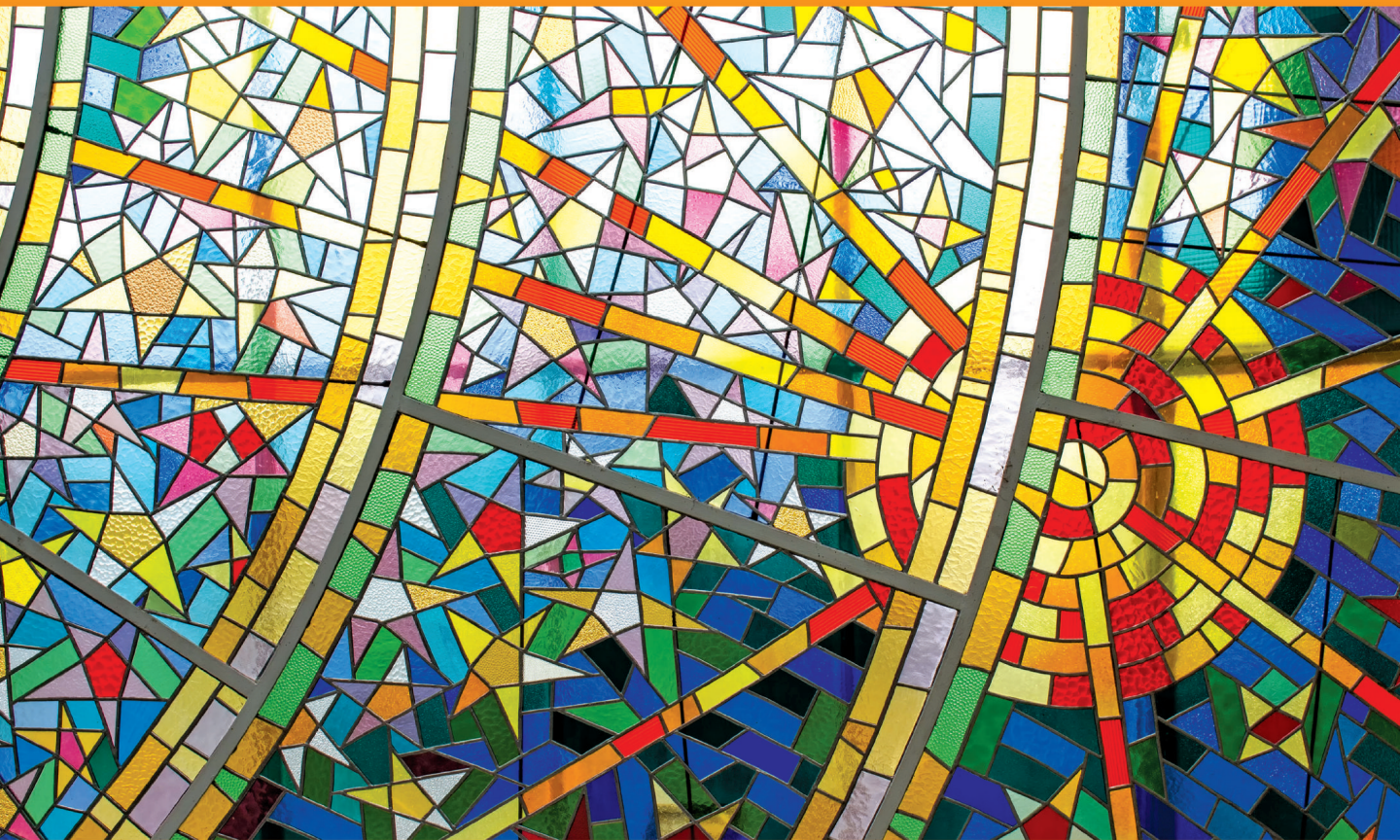


JIAWEI HAN ■ JIAN PEI ■ HANGHANG TONG



FOURTH EDITION

# DATA MINING

CONCEPTS AND TECHNIQUES

**MK**  
MORGAN KAUFMANN

# Data Mining

## Concepts and Techniques

This page intentionally left blank

# Data Mining

## Concepts and Techniques

Fourth Edition

**Jiawei Han**  
**Jian Pei**  
**Hanghang Tong**



**MK**

MORGAN KAUFMANN PUBLISHERS

ELSEVIER

AN IMPRINT OF ELSEVIER

Morgan Kaufmann is an imprint of Elsevier  
50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States

Copyright © 2023 Elsevier Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: [www.elsevier.com/permissions](http://www.elsevier.com/permissions).

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

#### Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

ISBN: 978-0-12-811760-6

For information on all Morgan Kaufmann publications  
visit our website at <https://www.elsevier.com/books-and-journals>

*Publisher:* Katey Birtcher  
*Acquisitions Editor:* Stephen Merken  
*Editorial Project Manager:* Beth LoGiudice  
*Publishing Services Manager:* Shereen Jameel  
*Production Project Manager:* Gayathri S  
*Designer:* Ryan Cook

Typeset by VTeX

Printed in the United States of America

Last digit is the print number: 9 8 7 6 5 4 3 2 1



*To Dora and Lawrence for your love and encouragement*

**J.H.**

*To Jennifer, Jacqueline, and Jasmine for your never-failing care,  
encouragement, and support*

**J.P.**

*To Jingrui, Emma, and Nathaniel for your endless love and inspiration*

**H.T.**

This page intentionally left blank

# Contents

Foreword	xvii
Foreword to second edition	xix
Preface	xxi
Acknowledgments	xxvii
About the authors	xxix
<b>CHAPTER 1 Introduction</b>	<b>1</b>
1.1 What is data mining?	1
1.2 Data mining: an essential step in knowledge discovery	2
1.3 Diversity of data types for data mining	4
1.4 Mining various kinds of knowledge	5
1.4.1 Multidimensional data summarization	6
1.4.2 Mining frequent patterns, associations, and correlations	6
1.4.3 Classification and regression for predictive analysis	7
1.4.4 Cluster analysis	9
1.4.5 Deep learning	9
1.4.6 Outlier analysis	10
1.4.7 Are all mining results interesting?	10
1.5 Data mining: confluence of multiple disciplines	12
1.5.1 Statistics and data mining	12
1.5.2 Machine learning and data mining	13
1.5.3 Database technology and data mining	15
1.5.4 Data mining and data science	15
1.5.5 Data mining and other disciplines	16
1.6 Data mining and applications	17
1.7 Data mining and society	19
1.8 Summary	19
1.9 Exercises	20
1.10 Bibliographic notes	21
<b>CHAPTER 2 Data, measurements, and data preprocessing</b>	<b>23</b>
2.1 Data types	24
2.1.1 Nominal attributes	24
2.1.2 Binary attributes	25
2.1.3 Ordinal attributes	25
2.1.4 Numeric attributes	26
2.1.5 Discrete vs. continuous attributes	27
2.2 Statistics of data	27
2.2.1 Measuring the central tendency	28
2.2.2 Measuring the dispersion of data	31



2.2.3	Covariance and correlation analysis . . . . .	34
2.2.4	Graphic displays of basic statistics of data . . . . .	38
<b>2.3</b>	<b>Similarity and distance measures . . . . .</b>	<b>43</b>
2.3.1	Data matrix vs. dissimilarity matrix . . . . .	43
2.3.2	Proximity measures for nominal attributes . . . . .	44
2.3.3	Proximity measures for binary attributes . . . . .	46
2.3.4	Dissimilarity of numeric data: Minkowski distance . . . . .	48
2.3.5	Proximity measures for ordinal attributes . . . . .	49
2.3.6	Dissimilarity for attributes of mixed types . . . . .	50
2.3.7	Cosine similarity . . . . .	52
2.3.8	Measuring similar distributions: the Kullback-Leibler divergence . . . . .	53
2.3.9	Capturing hidden semantics in similarity measures . . . . .	55
<b>2.4</b>	<b>Data quality, data cleaning, and data integration . . . . .</b>	<b>55</b>
2.4.1	Data quality measures . . . . .	55
2.4.2	Data cleaning . . . . .	56
2.4.3	Data integration . . . . .	62
<b>2.5</b>	<b>Data transformation . . . . .</b>	<b>63</b>
2.5.1	Normalization . . . . .	64
2.5.2	Discretization . . . . .	65
2.5.3	Data compression . . . . .	68
2.5.4	Sampling . . . . .	70
<b>2.6</b>	<b>Dimensionality reduction . . . . .</b>	<b>71</b>
2.6.1	Principal components analysis . . . . .	71
2.6.2	Attribute subset selection . . . . .	72
2.6.3	Nonlinear dimensionality reduction methods . . . . .	74
<b>2.7</b>	<b>Summary . . . . .</b>	<b>79</b>
<b>2.8</b>	<b>Exercises . . . . .</b>	<b>80</b>
<b>2.9</b>	<b>Bibliographic notes . . . . .</b>	<b>83</b>
<b>CHAPTER 3</b>	<b>Data warehousing and online analytical processing . . . . .</b>	<b>85</b>
<b>3.1</b>	<b>Data warehouse . . . . .</b>	<b>85</b>
3.1.1	Data warehouse: what and why? . . . . .	85
3.1.2	Architecture of data warehouses: enterprise data warehouses and data marts . . . . .	88
3.1.3	Data lakes . . . . .	93
<b>3.2</b>	<b>Data warehouse modeling: schema and measures . . . . .</b>	<b>96</b>
3.2.1	Data cube: a multidimensional data model . . . . .	97
3.2.2	Schemas for multidimensional data models: stars, snowflakes, and fact constellations . . . . .	99
3.2.3	Concept hierarchies . . . . .	103
3.2.4	Measures: categorization and computation . . . . .	105
<b>3.3</b>	<b>OLAP operations . . . . .</b>	<b>106</b>
3.3.1	Typical OLAP operations . . . . .	106
3.3.2	Indexing OLAP data: bitmap index and join index . . . . .	108
3.3.3	Storage implementation: column-based databases . . . . .	111

3.4	Data cube computation . . . . .	113
3.4.1	Terminology of data cube computation . . . . .	113
3.4.2	Data cube materialization: ideas . . . . .	115
3.4.3	OLAP server architectures: ROLAP vs. MOLAP vs. HOLAP . . . . .	117
3.4.4	General strategies for data cube computation . . . . .	119
3.5	Data cube computation methods . . . . .	120
3.5.1	Multiway array aggregation for full cube computation . . . . .	121
3.5.2	BUC: computing iceberg cubes from the apex cuboid downward . . . . .	125
3.5.3	Precomputing shell fragments for fast high-dimensional OLAP . . . . .	129
3.5.4	Efficient processing of OLAP queries using cuboids . . . . .	132
3.6	Summary . . . . .	133
3.7	Exercises . . . . .	135
3.8	Bibliographic notes . . . . .	142
<b>CHAPTER 4</b>	<b>Pattern mining: basic concepts and methods . . . . .</b>	<b>145</b>
4.1	Basic concepts . . . . .	145
4.1.1	Market basket analysis: a motivating example . . . . .	145
4.1.2	Frequent itemsets, closed itemsets, and association rules . . . . .	147
4.2	Frequent itemset mining methods . . . . .	149
4.2.1	Apriori algorithm: finding frequent itemsets by confined candidate generation . . . . .	150
4.2.2	Generating association rules from frequent itemsets . . . . .	153
4.2.3	Improving the efficiency of Apriori . . . . .	155
4.2.4	A pattern-growth approach for mining frequent itemsets . . . . .	157
4.2.5	Mining frequent itemsets using the vertical data format . . . . .	160
4.2.6	Mining closed and max patterns . . . . .	162
4.3	Which patterns are interesting?—Pattern evaluation methods . . . . .	163
4.3.1	Strong rules are not necessarily interesting . . . . .	163
4.3.2	From association analysis to correlation analysis . . . . .	164
4.3.3	A comparison of pattern evaluation measures . . . . .	165
4.4	Summary . . . . .	169
4.5	Exercises . . . . .	170
4.6	Bibliographic notes . . . . .	173
<b>CHAPTER 5</b>	<b>Pattern mining: advanced methods . . . . .</b>	<b>175</b>
5.1	Mining various kinds of patterns . . . . .	175
5.1.1	Mining multilevel associations . . . . .	175
5.1.2	Mining multidimensional associations . . . . .	179
5.1.3	Mining quantitative association rules . . . . .	180
5.1.4	Mining high-dimensional data . . . . .	183
5.1.5	Mining rare patterns and negative patterns . . . . .	185
5.2	Mining compressed or approximate patterns . . . . .	187
5.2.1	Mining compressed patterns by pattern clustering . . . . .	187
5.2.2	Extracting redundancy-aware top- $k$ patterns . . . . .	189

5.3	Constraint-based pattern mining . . . . .	191
5.3.1	Pruning pattern space with pattern pruning constraints . . . . .	193
5.3.2	Pruning data space with data pruning constraints . . . . .	196
5.3.3	Mining space pruning with succinctness constraints . . . . .	197
5.4	Mining sequential patterns . . . . .	198
5.4.1	Sequential pattern mining: concepts and primitives . . . . .	198
5.4.2	Scalable methods for mining sequential patterns . . . . .	200
5.4.3	Constraint-based mining of sequential patterns . . . . .	210
5.5	Mining subgraph patterns . . . . .	211
5.5.1	Methods for mining frequent subgraphs . . . . .	212
5.5.2	Mining variant and constrained substructure patterns . . . . .	219
5.6	Pattern mining: application examples . . . . .	223
5.6.1	Phrase mining in massive text data . . . . .	223
5.6.2	Mining copy and paste bugs in software programs . . . . .	230
5.7	Summary . . . . .	232
5.8	Exercises . . . . .	233
5.9	Bibliographic notes . . . . .	235
<b>CHAPTER 6</b>	<b>Classification: basic concepts and methods . . . . .</b>	<b>239</b>
6.1	Basic concepts . . . . .	239
6.1.1	What is classification? . . . . .	239
6.1.2	General approach to classification . . . . .	240
6.2	Decision tree induction . . . . .	243
6.2.1	Decision tree induction . . . . .	244
6.2.2	Attribute selection measures . . . . .	248
6.2.3	Tree pruning . . . . .	257
6.3	Bayes classification methods . . . . .	259
6.3.1	Bayes' theorem . . . . .	260
6.3.2	Naïve Bayesian classification . . . . .	262
6.4	Lazy learners (or learning from your neighbors) . . . . .	266
6.4.1	$k$ -nearest-neighbor classifiers . . . . .	266
6.4.2	Case-based reasoning . . . . .	269
6.5	Linear classifiers . . . . .	269
6.5.1	Linear regression . . . . .	270
6.5.2	Perceptron: turning linear regression to classification . . . . .	272
6.5.3	Logistic regression . . . . .	274
6.6	Model evaluation and selection . . . . .	278
6.6.1	Metrics for evaluating classifier performance . . . . .	278
6.6.2	Holdout method and random subsampling . . . . .	283
6.6.3	Cross-validation . . . . .	283
6.6.4	Bootstrap . . . . .	284
6.6.5	Model selection using statistical tests of significance . . . . .	285
6.6.6	Comparing classifiers based on cost-benefit and ROC curves . . . . .	286
6.7	Techniques to improve classification accuracy . . . . .	290
6.7.1	Introducing ensemble methods . . . . .	290

6.7.2	Bagging . . . . .	291
6.7.3	Boosting . . . . .	292
6.7.4	Random forests . . . . .	296
6.7.5	Improving classification accuracy of class-imbalanced data . . . . .	297
<b>6.8</b>	Summary . . . . .	298
<b>6.9</b>	Exercises . . . . .	299
<b>6.10</b>	Bibliographic notes . . . . .	302
<b>CHAPTER 7</b>	<b>Classification: advanced methods . . . . .</b>	<b>307</b>
<b>7.1</b>	Feature selection and engineering . . . . .	307
7.1.1	Filter methods . . . . .	308
7.1.2	Wrapper methods . . . . .	311
7.1.3	Embedded methods . . . . .	312
<b>7.2</b>	Bayesian belief networks . . . . .	315
7.2.1	Concepts and mechanisms . . . . .	315
7.2.2	Training Bayesian belief networks . . . . .	317
<b>7.3</b>	Support vector machines . . . . .	318
7.3.1	Linear support vector machines . . . . .	319
7.3.2	Nonlinear support vector machines . . . . .	324
<b>7.4</b>	Rule-based and pattern-based classification . . . . .	327
7.4.1	Using IF-THEN rules for classification . . . . .	328
7.4.2	Rule extraction from a decision tree . . . . .	330
7.4.3	Rule induction using a sequential covering algorithm . . . . .	331
7.4.4	Associative classification . . . . .	335
7.4.5	Discriminative frequent pattern-based classification . . . . .	338
<b>7.5</b>	Classification with weak supervision . . . . .	342
7.5.1	Semisupervised classification . . . . .	343
7.5.2	Active learning . . . . .	345
7.5.3	Transfer learning . . . . .	346
7.5.4	Distant supervision . . . . .	348
7.5.5	Zero-shot learning . . . . .	349
<b>7.6</b>	Classification with rich data type . . . . .	351
7.6.1	Stream data classification . . . . .	352
7.6.2	Sequence classification . . . . .	354
7.6.3	Graph data classification . . . . .	355
<b>7.7</b>	Potpourri: other related techniques . . . . .	359
7.7.1	Multiclass classification . . . . .	359
7.7.2	Distance metric learning . . . . .	362
7.7.3	Interpretability of classification . . . . .	364
7.7.4	Genetic algorithms . . . . .	367
7.7.5	Reinforcement learning . . . . .	367
<b>7.8</b>	Summary . . . . .	369
<b>7.9</b>	Exercises . . . . .	370
<b>7.10</b>	Bibliographic notes . . . . .	374

<b>CHAPTER 8</b>	<b>Cluster analysis: basic concepts and methods</b>	<b>379</b>
8.1	Cluster analysis	379
8.1.1	What is cluster analysis?	380
8.1.2	Requirements for cluster analysis	381
8.1.3	Overview of basic clustering methods	383
8.2	Partitioning methods	385
8.2.1	$k$ -Means: a centroid-based technique	386
8.2.2	Variations of $k$ -means	388
8.3	Hierarchical methods	394
8.3.1	Basic concepts of hierarchical clustering	394
8.3.2	Agglomerative hierarchical clustering	397
8.3.3	Divisive hierarchical clustering	400
8.3.4	BIRCH: scalable hierarchical clustering using clustering feature trees	402
8.3.5	Probabilistic hierarchical clustering	404
8.4	Density-based and grid-based methods	407
8.4.1	DBSCAN: density-based clustering based on connected regions with high density	408
8.4.2	DENCLUE: clustering based on density distribution functions	411
8.4.3	Grid-based methods	414
8.5	Evaluation of clustering	417
8.5.1	Assessing clustering tendency	417
8.5.2	Determining the number of clusters	419
8.5.3	Measuring clustering quality: extrinsic methods	420
8.5.4	Intrinsic methods	424
8.6	Summary	425
8.7	Exercises	427
8.8	Bibliographic notes	429
<b>CHAPTER 9</b>	<b>Cluster analysis: advanced methods</b>	<b>431</b>
9.1	Probabilistic model-based clustering	431
9.1.1	Fuzzy clusters	433
9.1.2	Probabilistic model-based clusters	435
9.1.3	Expectation-maximization algorithm	438
9.2	Clustering high-dimensional data	441
9.2.1	Why is clustering high-dimensional data challenging?	441
9.2.2	Axis-parallel subspace approaches	445
9.2.3	Arbitrarily oriented subspace approaches	447
9.3	Biclustering	447
9.3.1	Why and where is biclustering useful?	448
9.3.2	Types of biclusters	450
9.3.3	Biclustering methods	452
9.3.4	Enumerating all biclusters using MaPle	453
9.4	Dimensionality reduction for clustering	454
9.4.1	Linear dimensionality reduction methods for clustering	455
9.4.2	Nonnegative matrix factorization (NMF)	458

9.4.3	Spectral clustering . . . . .	460
<b>9.5</b>	<b>Clustering graph and network data . . . . .</b>	<b>463</b>
9.5.1	Applications and challenges . . . . .	463
9.5.2	Similarity measures . . . . .	465
9.5.3	Graph clustering methods . . . . .	470
<b>9.6</b>	<b>Semisupervised clustering . . . . .</b>	<b>475</b>
9.6.1	Semisupervised clustering on partially labeled data . . . . .	475
9.6.2	Semisupervised clustering on pairwise constraints . . . . .	476
9.6.3	Other types of background knowledge for semisupervised clustering . . . . .	477
<b>9.7</b>	<b>Summary . . . . .</b>	<b>479</b>
<b>9.8</b>	<b>Exercises . . . . .</b>	<b>480</b>
<b>9.9</b>	<b>Bibliographic notes . . . . .</b>	<b>482</b>
<b>CHAPTER 10</b>	<b>Deep learning . . . . .</b>	<b>485</b>
<b>10.1</b>	<b>Basic concepts . . . . .</b>	<b>485</b>
10.1.1	What is deep learning? . . . . .	485
10.1.2	Backpropagation algorithm . . . . .	489
10.1.3	Key challenges for training deep learning models . . . . .	498
10.1.4	Overview of deep learning architecture . . . . .	499
<b>10.2</b>	<b>Improve training of deep learning models . . . . .</b>	<b>500</b>
10.2.1	Responsive activation functions . . . . .	500
10.2.2	Adaptive learning rate . . . . .	501
10.2.3	Dropout . . . . .	504
10.2.4	Pretraining . . . . .	507
10.2.5	Cross-entropy . . . . .	509
10.2.6	Autoencoder: unsupervised deep learning . . . . .	511
10.2.7	Other techniques . . . . .	514
<b>10.3</b>	<b>Convolutional neural networks . . . . .</b>	<b>517</b>
10.3.1	Introducing convolution operation . . . . .	517
10.3.2	Multidimensional convolution . . . . .	519
10.3.3	Convolutional layer . . . . .	523
<b>10.4</b>	<b>Recurrent neural networks . . . . .</b>	<b>526</b>
10.4.1	Basic RNN models and applications . . . . .	526
10.4.2	Gated RNNs . . . . .	532
10.4.3	Other techniques for addressing long-term dependence . . . . .	536
<b>10.5</b>	<b>Graph neural networks . . . . .</b>	<b>539</b>
10.5.1	Basic concepts . . . . .	540
10.5.2	Graph convolutional networks . . . . .	541
10.5.3	Other types of GNNs . . . . .	545
<b>10.6</b>	<b>Summary . . . . .</b>	<b>547</b>
<b>10.7</b>	<b>Exercises . . . . .</b>	<b>548</b>
<b>10.8</b>	<b>Bibliographic notes . . . . .</b>	<b>552</b>
<b>CHAPTER 11</b>	<b>Outlier detection . . . . .</b>	<b>557</b>
<b>11.1</b>	<b>Basic concepts . . . . .</b>	<b>557</b>

11.1.1	What are outliers? . . . . .	558
11.1.2	Types of outliers . . . . .	559
11.1.3	Challenges of outlier detection . . . . .	561
11.1.4	An overview of outlier detection methods . . . . .	562
<b>11.2</b>	<b>Statistical approaches . . . . .</b>	<b>565</b>
11.2.1	Parametric methods . . . . .	565
11.2.2	Nonparametric methods . . . . .	569
<b>11.3</b>	<b>Proximity-based approaches . . . . .</b>	<b>572</b>
11.3.1	Distance-based outlier detection . . . . .	572
11.3.2	Density-based outlier detection . . . . .	573
<b>11.4</b>	<b>Reconstruction-based approaches . . . . .</b>	<b>576</b>
11.4.1	Matrix factorization–based methods for numerical data . . . . .	577
11.4.2	Pattern-based compression methods for categorical data . . . . .	582
<b>11.5</b>	<b>Clustering- vs. classification-based approaches . . . . .</b>	<b>585</b>
11.5.1	Clustering-based approaches . . . . .	585
11.5.2	Classification-based approaches . . . . .	588
<b>11.6</b>	<b>Mining contextual and collective outliers . . . . .</b>	<b>590</b>
11.6.1	Transforming contextual outlier detection to conventional outlier detection . . . . .	591
11.6.2	Modeling normal behavior with respect to contexts . . . . .	591
11.6.3	Mining collective outliers . . . . .	592
<b>11.7</b>	<b>Outlier detection in high-dimensional data . . . . .</b>	<b>593</b>
11.7.1	Extending conventional outlier detection . . . . .	594
11.7.2	Finding outliers in subspaces . . . . .	595
11.7.3	Outlier detection ensemble . . . . .	596
11.7.4	Taming high dimensionality by deep learning . . . . .	597
11.7.5	Modeling high-dimensional outliers . . . . .	599
<b>11.8</b>	<b>Summary . . . . .</b>	<b>600</b>
<b>11.9</b>	<b>Exercises . . . . .</b>	<b>601</b>
<b>11.10</b>	<b>Bibliographic notes . . . . .</b>	<b>602</b>
<b>CHAPTER 12</b>	<b>Data mining trends and research frontiers . . . . .</b>	<b>605</b>
<b>12.1</b>	<b>Mining rich data types . . . . .</b>	<b>605</b>
12.1.1	Mining text data . . . . .	605
12.1.2	Spatial-temporal data . . . . .	610
12.1.3	Graph and networks . . . . .	612
<b>12.2</b>	<b>Data mining applications . . . . .</b>	<b>617</b>
12.2.1	Data mining for sentiment and opinion . . . . .	617
12.2.2	Truth discovery and misinformation identification . . . . .	620
12.2.3	Information and disease propagation . . . . .	623
12.2.4	Productivity and team science . . . . .	626
<b>12.3</b>	<b>Data mining methodologies and systems . . . . .</b>	<b>629</b>
12.3.1	Structuring unstructured data for knowledge mining: a data-driven approach . . . . .	629
12.3.2	Data augmentation . . . . .	632

12.3.3	From correlation to causality . . . . .	635
12.3.4	Network as a context . . . . .	637
12.3.5	Auto-ML: methods and systems . . . . .	640
<b>12.4</b>	<b>Data mining, people, and society . . . . .</b>	<b>642</b>
12.4.1	Privacy-preserving data mining . . . . .	642
12.4.2	Human-algorithm interaction . . . . .	646
12.4.3	Mining beyond maximizing accuracy: fairness, interpretability, and robustness . . . . .	648
12.4.4	Data mining for social good . . . . .	652
<b>APPENDIX A</b>	<b>Mathematical background . . . . .</b>	<b>655</b>
<b>A.1</b>	<b>Probability and statistics . . . . .</b>	<b>655</b>
A.1.1	PDF of typical distributions . . . . .	655
A.1.2	MLE and MAP . . . . .	656
A.1.3	Significance test . . . . .	657
A.1.4	Density estimation . . . . .	658
A.1.5	Bias-variance tradeoff . . . . .	659
A.1.6	Cross-validation and Jackknife . . . . .	660
<b>A.2</b>	<b>Numerical optimization . . . . .</b>	<b>661</b>
A.2.1	Gradient descent . . . . .	661
A.2.2	Variants of gradient descent . . . . .	662
A.2.3	Newton’s method . . . . .	664
A.2.4	Coordinate descent . . . . .	666
A.2.5	Quadratic programming . . . . .	666
<b>A.3</b>	<b>Matrix and linear algebra . . . . .</b>	<b>668</b>
A.3.1	Linear system $\mathbf{Ax} = \mathbf{b}$ . . . . .	668
A.3.2	Norms of vectors and matrices . . . . .	669
A.3.3	Matrix decompositions . . . . .	669
A.3.4	Subspace . . . . .	671
A.3.5	Orthogonality . . . . .	672
<b>A.4</b>	<b>Concepts and tools from signal processing . . . . .</b>	<b>673</b>
A.4.1	Entropy . . . . .	673
A.4.2	Kullback-Leibler divergence (KL-divergence) . . . . .	674
A.4.3	Mutual information . . . . .	675
A.4.4	Discrete Fourier transform (DFT) and fast Fourier transform (FFT) . .	676
<b>A.5</b>	<b>Bibliographic notes . . . . .</b>	<b>678</b>
	Bibliography . . . . .	681
	Index . . . . .	735



This page intentionally left blank

# Foreword

Analyzing data is more important and prevalent than ever. Collecting and storing large datasets is easy; disks and “clouds” are well within budget of even small institutions. There is no excuse to not analyze the data to find patterns, trends, anomalies, and forecasts.

The 4th edition of *Data Mining: Concepts and Techniques* covers all the classics but adds significant material on recent developments. It has a whole chapter on deep learning, subchapters for vital topics like text mining (including one of my favorite algorithms, TopMine), frequent-subgraph discovery (covering gSpan and CloseGraph), and excellent summaries for explainability (LIME), genetic algorithms, reinforcement learning, misinformation detection, productivity and team science, causality, fairness, and social good.

The new appendix with mathematical background is extremely useful and convenient—it has all the fundamental formulas for data mining in one place, like gradient descent, Newton, and related material for optimization; SVD, eigenvalues and pseudo-inverse for matrix algebra; entropy and KL for information theory; and DFT and FFT for signal processing.

The book has an impressive, carefully chosen list of more than 800 citations, with more than 250 citations for papers after 2015. In short, this edition continues serving both as an excellent textbook and an encyclopedic reference book.

Christos Faloutsos  
Carnegie Mellon University  
Pittsburgh, June 2022

This page intentionally left blank

# Foreword to second edition

We are deluged by data—scientific data, medical data, demographic data, financial data, and marketing data. People have no time to look at this data. Human attention has become the precious resource. So, we must find ways to automatically analyze the data, to automatically classify it, to automatically summarize it, to automatically discover and characterize trends in it, and to automatically flag anomalies. This is one of the most active and exciting areas of the database research community. Researchers in areas including statistics, visualization, artificial intelligence, and machine learning are contributing to this field. The breadth of the field makes it difficult to grasp the extraordinary progress over the last few decades.

Six years ago, Jiawei Han’s and Micheline Kamber’s seminal textbook organized and presented Data Mining. It heralded a golden age of innovation in the field. This revision of their book reflects that progress; more than half of the references and historical notes are to recent work. The field has matured with many new and improved algorithms, and has broadened to include many more datatypes: streams, sequences, graphs, time-series, geospatial, audio, images, and video. We are certainly not at the end of the golden age—indeed research and commercial interest in data mining continues to grow—but we are all fortunate to have this modern compendium.

The book gives quick introductions to database and data mining concepts with particular emphasis on data analysis. It then covers in a chapter-by-chapter tour the concepts and techniques that underlie classification, prediction, association, and clustering. These topics are presented with examples, a tour of the best algorithms for each problem class, and with pragmatic rules of thumb about when to apply each technique. The Socratic presentation style is both very readable and very informative. I certainly learned a lot from reading the first edition and got re-educated and updated in reading the second edition.

Jiawei Han and Micheline Kamber have been leading contributors to data mining research. This is the text they use with their students to bring them up to speed on the field. The field is evolving very rapidly, but this book is a quick way to learn the basic ideas and to understand where the field is today. I found it very informative and stimulating, and believe you will too.

Jim Gray  
*In his memory*

This page intentionally left blank

# Preface

The computerization of our society has substantially enhanced our capabilities for both generating and collecting data from diverse sources. A tremendous amount of data has flooded almost every aspect of our lives. This explosive growth in stored or transient data has generated an urgent need for new techniques and automated tools that can intelligently assist us in transforming the vast amounts of data into useful information and knowledge. This has led to the generation of a promising and flourishing frontier in computer science called *data mining* and its various applications. Data mining, also popularly referred to as *knowledge discovery from data (KDD)*, is the automated or convenient extraction of patterns representing knowledge implicitly stored or captured in large databases, data warehouses, the Web, other massive information repositories, or data streams.

This book explores the concepts and techniques of *knowledge discovery* and *data mining*. As a multidisciplinary field, data mining draws on work from areas including statistics, machine learning, pattern recognition, database technology, information retrieval, natural language processing, network science, knowledge-based systems, artificial intelligence, high-performance computing, and data visualization. We focus on issues relating to the feasibility, usefulness, effectiveness, and scalability of techniques for the discovery of patterns hidden in *large data sets*. As a result, this book is not intended as an introduction to statistics, machine learning, database systems, or other such areas, although we do provide some background knowledge to facilitate the reader's comprehension of their respective roles in data mining. Rather, the book is a comprehensive introduction to data mining. It is useful for computer science students, application developers, and business professionals, as well as researchers involved in any of the disciplines listed above.

Data mining emerged during the late 1980s, made great strides during the 1990s, and continues to flourish into the new millennium. This book presents an overall picture of the field, introducing interesting data mining concepts and techniques and discussing applications and research directions. An important motivation for writing this book was the need to build an organized framework for the study of data mining—a challenging task, owing to the extensive multidisciplinary nature of this fast-developing field. We hope that this book will encourage people with different backgrounds and experiences to exchange their views regarding data mining to contribute toward the further promotion and shaping of this exciting and dynamic field.

## Organization of the book

Since the publication of the first three editions of this book, great progress has been made in the field of data mining. Many new data mining methodologies, systems, and applications have been developed, especially for handling new kinds of data, including information networks, graphs, complex structures, and data streams, as well as text, Web, multimedia, time-series, and spatiotemporal data. Such fast development and rich, new technical contents make it difficult to cover the full spectrum of the field in a single book. Instead of continuously expanding the coverage of this book, we have decided to cover the core material in sufficient scope and depth, and leave the handling of complex data types and their applications to the books dedicated to those specific topics.

The 4th edition substantially revises the first three editions of the book, with numerous enhancements and a reorganization of the technical contents. The core technical material, which handles different mining methodologies on general data types, is expanded and substantially enhanced. To keep the book concise and up-to-date, we have done the following major revisions: (1) Two chapters in the 3rd edition, “Getting to Know You Data” and “Data Preprocessing” are merged into one chapter “Data, Measurements and Data Preprocessing,” with the “Data Visualization” section removed since the concepts are easy to understand, the methods have been covered in multiple, dedicated data visualization books, and the software tools are widely available on the web; (2) two chapters in the 3rd edition, “Data Warehousing and Online Analytical Processing” and “Data Cube Technology” are merged into one chapter, with some not well-adopted data cube computation methods and data cube extensions omitted, but with a newer concept, “Data Lakes” introduced; (3) the key data mining method chapters in the 3rd edition on pattern discovery, classification, clustering and outlier analysis are retained with contents substantially enhanced and updated; (4) a new chapter “Deep Learning” is added, with a systematic introduction to neural network and deep learning methodologies; (5) the final chapter on “Data Mining Trends and Research Frontiers” is completely rewritten with many new advanced topics on data mining introduced in comprehensive and concise way; and finally, (6) a set of appendices that briefly introduce essential mathematical background needed to understand the contents of this book.

The chapters of this new edition are described briefly as follows, with emphasis on the new material.

**Chapter 1** provides an *introduction* to the multidisciplinary field of data mining. It discusses the evolutionary path of information technology, which has led to the need for data mining, and the importance of its applications. It examines various kinds of data to be mined, and presents a general classification of data mining tasks, based on the kinds of knowledge to be mined, the kinds of technologies used, and the kinds of applications that are targeted. It shows that data mining is a confluence of multiple disciplines, with broad applications. Finally, it discusses how data mining may impact society.

**Chapter 2** introduces the *data, measurements and data preprocessing*. It first discusses data objects and attribute types, and then introduces typical measures for basic statistical data descriptions. It also introduces ways to measure similarity and dissimilarity for various kinds of data. Then, the chapter moves to introduce techniques for data preprocessing. In particular, it introduces the concept of data quality and methods for data cleaning and data integration. It also discusses various methods for data transformation and dimensionality reduction.

**Chapter 3** provides a comprehensive introduction to *datawarehouses* and online analytical processing (*OLAP*). The chapter starts with a well-accepted definition of data warehouse, an introduction to the architecture, and the concept of data lake. Then it studies the logical design of a data warehouse as a multidimensional data model, and looks at OLAP operations and how to index OLAP data for efficient analytics. The chapter includes an in-depth treatment of the techniques of building data cube as a way of implementing a data warehouse.

Chapters 4 and 5 present methods for *mining frequent patterns, associations, and correlations* in large data sets. **Chapter 4** introduces fundamental concepts, such as market basket analysis, with many techniques for frequent itemset mining presented in an organized way. These range from the basic Apriori algorithm and its variations to more advanced methods that improve efficiency, including the frequent pattern growth approach, frequent pattern mining with vertical data format, and mining closed and max frequent itemsets. The chapter also discusses pattern evaluation methods and introduces measures for mining correlated patterns. **Chapter 5** is on advanced pattern mining methods. It discusses methods for pattern mining in multilevel and multidimensional space, mining quantitative association

rules, mining high-dimensional data, mining rare and negative patterns, mining compressed or approximate patterns, and constraint-based pattern mining. It then moves to advanced methods for mining sequential patterns and subgraph patterns. It also presents applications of pattern mining, including phrase mining in text data and mining copy and paste bugs in software programs.

Chapters 6 and 7 describe methods for *data classification*. Due to the importance and diversity of classification methods, the contents are partitioned into two chapters. **Chapter 6** introduces basic concepts and methods for classification, including decision tree induction, Bayes classification,  $k$ -nearest neighbor classifiers, and linear classifiers. It also discusses model evaluation and selection methods and methods for improving classification accuracy, including ensemble methods and how to handle imbalanced data. **Chapter 7** discusses advanced methods for classification, including feature selection, Bayesian belief networks, support vector machines, rule-based and pattern-based classification. Additional topics include classification with weak supervision, classification with rich data type, multiclass classification, distant metric learning, interpretation of classification, genetic algorithms and reinforcement learning.

*Cluster analysis* forms the topic of Chapters 8 and 9. **Chapter 8** introduces the basic concepts and methods for data clustering, including an overview of basic cluster analysis methods, partitioning methods, hierarchical methods, density-based and grid-based methods. It also introduces methods for the evaluation of clustering. **Chapter 9** discusses advanced methods for clustering, including probabilistic model-based clustering, clustering high-dimensional data, clustering graph and network data, and semisupervised clustering.

**Chapter 10** introduces *deep learning*, which is a powerful family of techniques based on artificial neural networks with broad applications in computer vision, natural language processing, machine translation, social network analysis, and so on. We start with the basic concepts and a foundational technique called backpropagation algorithm. Then, we introduce various techniques to improve the training of deep learning models, including responsive activation functions, adaptive learning rate, dropout, pretraining, cross-entropy, and autoencoder. We also introduce several commonly used deep learning architectures, ranging from feed-forward neural networks, convolutional neural networks, recurrent neural networks, and graph neural networks.

**Chapter 11** is dedicated to *outlier detection*. It introduces the basic concepts of outliers and outlier analysis and discusses various outlier detection methods from the view of degree of supervision (i.e., supervised, semisupervised, and unsupervised methods), as well as from the view of approaches (i.e., statistical methods, proximity-based methods, reconstruction-based methods, clustering-based methods, and classification-based methods). It also discusses methods for mining contextual and collective outliers, and for outlier detection in high-dimensional data.

Finally, in **Chapter 12**, we discuss *future trends* and *research frontiers* in data mining. We start with a brief coverage of mining complex data types, including text data, graphs and networks, and spatiotemporal data. After that, we introduce a few data mining applications, ranging from sentiment and opinion analysis, truth discovery and misinformation identification, information and disease propagation, to productivity and team science. The chapter then moves ahead to cover other data mining methodologies, including structuring unstructured data, data augmentation, causality analysis, network-as-a-context, and auto-ML. Finally, it discusses social impacts of data mining, including privacy-preserving data mining, human-algorithm interaction, fairness, interpretability and robustness, and data mining for social good.



Throughout the text, *italic* font is used to emphasize terms that are defined, and **bold** font is used to highlight or summarize main ideas. Sans serif font is used for reserved words. Bold italic font is used to represent multidimensional quantities.

This book has several strong features that set it apart from other textbooks on data mining. It presents a very broad yet in-depth coverage of the principles of data mining. The chapters are written to be as self-contained as possible, so they may be read in order of interest by the reader. Advanced chapters offer a larger-scale view and may be considered optional for interested readers. All of the major methods of data mining are presented. The book presents important topics in data mining regarding multidimensional OLAP analysis, which is often overlooked or minimally treated in other data mining books. The book also maintains web sites with a number of online resources to aid instructors, students, and professionals in the field. These are described further in the following.

## To the instructor

This book is designed to give a broad, yet detailed overview of the data mining field. First, it can be used to teach an introductory course on data mining at an advanced undergraduate level or at the first-year graduate level. Moreover, the book also provides essential materials for an advanced graduate course on data mining.

Depending on the length of the instruction period, the background of students, and your interests, you may select subsets of chapters to teach in various sequential orderings. For example, an introductory course may cover the following chapters.

- Chapter 1: Introduction
- Chapter 2: Data, measurements, and data preprocessing
- Chapter 3: Data warehousing and online analytical processing
- Chapter 4: Pattern mining: basic concepts and methods
- Chapter 6: Classification: basic concepts
- Chapter 8: Cluster analysis: basic concepts and methods

If time permits, some materials about deep learning (Chapter 10) or outlier detection (Chapter 11) may be chosen. In each chapter, the fundamental concepts should be covered, while some sections on advanced topics can be treated optionally.

As another example, for a place where a machine learning course is offered to cover supervised learning well, a data mining course can discuss in depth on clustering. Such a course can be based on the following chapters.

- Chapter 1: Introduction
- Chapter 2: Data, measurements, and data preprocessing
- Chapter 3: Data warehousing and online analytical processing
- Chapter 4: Pattern mining: basic concepts and methods
- Chapter 8: Cluster analysis: basic concepts and methods
- Chapter 9: Cluster analysis: advanced methods
- Chapter 11: Outlier detection

An instructor teaching an advanced data mining course may find Chapter 12 particularly informative, since it discusses an extensive spectrum of new topics in data mining that are under dynamic and fast development.

Alternatively, you may choose to teach the whole book in a two-course sequence that covers all of the chapters in the book, plus, when time permits, some advanced topics such as graph and network mining. Material for such advanced topics may be selected from the companion chapters available from the book's web site, accompanied with a set of selected research papers.

Individual chapters in this book can also be used for tutorials or for special topics in related courses, such as machine learning, pattern recognition, data warehousing, and intelligent data analysis.

Each chapter ends with a set of exercises, suitable as assigned homework. The exercises are either short questions that test basic mastery of the material covered, longer questions that require analytical thinking, or implementation projects. Some exercises can also be used as research discussion topics. The bibliographic notes at the end of each chapter can be used to find the research literature that contains the origin of the concepts and methods presented, in-depth treatment of related topics, and possible extensions.

## **To the student**

We hope that this textbook will spark your interest in the young yet fast-evolving field of data mining. We have attempted to present the material in a clear manner, with careful explanation of the topics covered. Each chapter ends with a summary describing the main points. We have included many figures and illustrations throughout the text to make the book more enjoyable and reader-friendly. Although this book was designed as a textbook, we have tried to organize it so that it will also be useful to you as a reference book or handbook, should you later decide to perform in-depth research in the related fields or pursue a career in data mining.

What do you need to know to read this book?

- You should have some knowledge of the concepts and terminology associated with statistics, database systems, and machine learning. However, we do try to provide enough background of the basics, so that if you are not so familiar with these fields or your memory is a bit rusty, you will not have trouble following the discussions in the book.
- You should have some programming experience. In particular, you should be able to read pseudocode and understand simple data structures such as multidimensional arrays and structures.

## **To the professional**

This book was designed to cover a wide range of topics in the data mining field. As a result, it is an excellent handbook on the subject. Because each chapter is designed to be as standalone as possible, you can focus on the topics that most interest you. The book can be used by application programmers, data scientists, and information service managers who wish to learn about the key ideas of data mining on their own. The book would also be useful for technical data analysis staff in banking, insurance, medicine, and retailing industries who are interested in applying data mining solutions to their businesses. Moreover, the book may serve as a comprehensive survey of the data mining field, which may also benefit researchers who would like to advance the state-of-the-art in data mining and extend the scope of data mining applications.

The techniques and algorithms presented are of practical utility. Rather than selecting algorithms that perform well on small “toy” data sets, the algorithms described in the book are geared for the discovery of patterns and knowledge hidden in large, real data sets. Algorithms presented in the book are illustrated in pseudocode. The pseudocode is similar to the C programming language, yet is designed so that it should be easy to follow by programmers unfamiliar with C or C++. If you wish to implement any of the algorithms, you should find the translation of our pseudocode into the programming language of your choice to be a fairly straightforward task.

### Book web site with resources

The book has a website with Elsevier at <https://educate.elsevier.com/book/details/9780128117606>. This website contains many supplemental materials for readers of the book or anyone else with an interest in data mining. The resources include the following:

- **Slide presentations for each chapter.** Lecture notes in Microsoft PowerPoint slides are available for each chapter.
- **Instructors’ manual.** This complete set of answers to the exercises in the book is available only to instructors from the publisher’s web site.
- **Figures from the book.** This may help you to make your own slides for your classroom teaching.
- **Table of Contents** of the book in PDF format.
- **Errata on the different printings of the book.** We encourage you to point out any errors in this book. Once the error is confirmed, we will update the errata list and include acknowledgment of your contribution.

Interested readers may also like to check **Authors’ course teaching websites**. All the authors are university professors in their respective universities. Please check their corresponding data mining course websites which may contain their undergraduate and/or graduate course materials for introductory and/or advanced courses on data mining, including updated course/chapter slides, syllabi, homeworks, programming assignments, research projects, errata, and other related information.

# Acknowledgments

## **Fourth edition of the book**

We would like to express our sincere thanks to Micheline Kamber, the co-author of the previous editions of this book. Micheline has contributed substantially to these editions. Due to her commitment of other duties, she will not be able to join us in this new edition. We really appreciate her long-term collaborations and contributions in the past many years.

We would also like to express our grateful thanks to the previous and current members, including faculty and students, of the Data and Information Systems (DAIS) Laboratory, the Data Mining Group, IDEA Lab and iSAIL Lab at UIUC and the Data Mining Group at SFU, and many friends and colleagues, whose constant support and encouragement have made our work on this edition a rewarding experience. Our thanks extend to the students and TAs in the many data mining courses we taught at UIUC and SFU, as well as those in summer schools and beyond, who carefully went through the early drafts and the early editions of this book, identified many errors, and suggested various improvements.

We also wish to thank Steve Merken and Beth LoGiudice at Elsevier, for their enthusiasm, patience, and support during our writing of this edition of the book. We thank Gayathri S, the Project Manager, and her team members, for keeping us on schedule. We are also grateful for the invaluable feedback from all of the reviewers.

We would like to thank the US National Science Foundation (NSF), US Defense Advanced Research Projects Agency (DARPA), US Army Research Laboratory (ARL), US National Institute of Health (NIH), US Defense Threat Reduction Agency (DTRA), and Natural Science and Engineering Research Council of Canada (NSERC), as well as Microsoft Research, Google Research, IBM Research, Amazon, Adobe, LinkedIn, Yahoo!, HP Labs, PayPal, Facebook, Visa Research, and other industry research labs for their support of our research in the form of research grants, contracts, and gifts. Such research support deepens our understanding of the subjects discussed in this book.

Finally, we thank our families for their wholehearted support throughout this project.

This page intentionally left blank

# About the authors

**Jiawei Han** is a Michael Aiken Chair Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has received numerous awards for his contributions on research into knowledge discovery and data mining, including ACM SIGKDD Innovation Award (2004), IEEE Computer Society Technical Achievement Award (2005), and IEEE W. Wallace McDowell Award (2009). He is a Fellow of ACM and a Fellow of IEEE. He served as founding Editor-in-Chief of *ACM Transactions on Knowledge Discovery from Data* (2006–2011) and as an editorial board member of several journals, including *IEEE Transactions on Knowledge and Data Engineering* and *Data Mining and Knowledge Discovery*.

**Jian Pei** is currently Professor of Computer Science, Biostatistics and Bioinformatics, and Electrical and Computer Engineering at Duke University. He received a Ph.D. degree in computing science from Simon Fraser University in 2002 under Dr. Jiawei Han's supervision. He has published prolifically in the premier academic forums on data mining, databases, Web searching, and information retrieval and actively served the academic community. He is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, ACM, and IEEE. He received the 2017 ACM SIGKDD Innovation Award and the 2015 ACM SIGKDD Service Award.

**Hanghang Tong** is currently an associate professor at Department of Computer Science at University of Illinois at Urbana-Champaign. He received his Ph.D. degree from Carnegie Mellon University in 2009. He has published over 200 refereed articles. His research is recognized by several prestigious awards and thousands of citations. He is the Editor-in-Chief of SIGKDD Explorations (ACM) and an associate editor of several journals.

This page intentionally left blank

# Introduction

**This book is an introduction** to the young and fast-growing field of *data mining* (also known as *knowledge discovery from data*, or *KDD* for short). The book focuses on fundamental data mining concepts and techniques for discovering interesting patterns from data in various applications. In particular, we emphasize prominent techniques for developing effective, efficient, and scalable data mining tools.

This chapter is organized as follows. In Section 1.1, we learn what is data mining and why data mining is in high demand. Section 1.2 links data mining with the overall knowledge discovery process. Next, we learn about data mining from multiple aspects, such as the kinds of data that can be mined (Section 1.3), the kinds of knowledge to be mined (Section 1.4), the relationship between data mining and other disciplines (Section 1.5), and data mining applications (Section 1.6). Finally, we discuss the impact of data mining to society (Section 1.7).

---

## 1.1 What is data mining?

*Necessity, who is the mother of invention.*  
– Plato

We live in a world where vast amounts of data are generated constantly and rapidly.

“*We are living in the information age*” is a popular saying; however, *we are actually living in the data age*. Terabytes or petabytes of data pour into our computer networks, the World Wide Web (WWW), and various kinds of devices every day from business, news agencies, society, science, engineering, medicine, and almost every other aspect of daily life. This explosive growth of available data volume is a result of the computerization of our society and the fast development of powerful computing, sensing, and data collection, storage, and publication tools.

Businesses worldwide generate gigantic data sets, including sales transactions, stock trading records, product descriptions, sales promotions, company profiles and performance, and customer feedback. Scientific and engineering practices generate high orders of petabytes of data in a continuous manner, from remote sensing, to process measuring, scientific experiments, system performance, engineering observations, and environment surveillance. Biomedical research and the health industry generate tremendous amounts of data from gene sequence machines, biomedical experiment and research reports, medical records, patient monitoring, and medical imaging. Billions of Web searches supported by search engines process tens of petabytes of data daily. Social media tools have become increasingly popular, producing a tremendous number of texts, pictures, and videos, generating various kinds of Web communities and social networks. The list of sources that generate huge amounts of data is endless.



This explosively growing, widely available, and gigantic body of data makes our time truly *the data age*. Powerful and versatile tools are badly needed to automatically uncover valuable information from the tremendous amounts of data and to transform such data into organized knowledge. This necessity has led to the birth of data mining.

Essentially, **data mining** is the process of discovering interesting patterns, models, and other kinds of knowledge in large data sets. The term, *data mining*, as a vivid view of searching for *gold nuggets* from data, appeared in 1990s. However, to refer to the mining of gold from rocks or sand, we say *gold mining* instead of rock or sand mining. Analogously, data mining should have been more appropriately named “knowledge mining from data,” which is unfortunately somewhat long. However, the shorter term, *knowledge mining* may not reflect the emphasis on mining from large amounts of data. Nevertheless, mining is a vivid term characterizing the process that finds a small set of precious nuggets from a great deal of raw material. Thus, such a misnomer carrying both “data” and “mining” became a popular choice. In addition, many other terms have a similar meaning to data mining—for example, *knowledge mining from data*, *KDD* (i.e., *Knowledge Discovery from Data*), *pattern discovery*, *knowledge extraction*, *data archaeology*, *data analytics*, and *information harvesting*.

Data mining is a young, dynamic, and promising field. It has made and will continue to make great strides in our journey from the data age toward the coming information age.

**Example 1.1. Data mining turns a large collection of data into knowledge.** A search engine (e.g., Google) receives billions of queries every day. What novel and useful knowledge can a search engine learn from such a huge collection of queries collected from users over time? Interestingly, some patterns found in user search queries can disclose invaluable knowledge that cannot be obtained by reading individual data items alone. For example, Google’s *Flu Trends* uses specific search terms as indicators of flu activity. It found a close relationship between the number of people who search for flu-related information and the number of people who actually have flu symptoms. A pattern emerges when all of the search queries related to flu are aggregated. Using aggregated Google search data, *Flu Trends* can estimate flu activity up to two weeks faster than what traditional systems can.<sup>1</sup> This example shows how data mining can turn a large collection of data into knowledge that can help meet a current global challenge. □

---

## 1.2 Data mining: an essential step in knowledge discovery

Many people treat data mining as a synonym for another popularly used term, **knowledge discovery from data**, or **KDD**, whereas others view data mining as merely an essential step in the overall process of knowledge discovery. The overall knowledge discovery process is shown in Fig. 1.1 as an iterative sequence of the following steps:

### 1. Data preparation

- a. **Data cleaning** (to remove noise and inconsistent data)
- b. **Data integration** (where multiple data sources may be combined)<sup>2</sup>

---

<sup>1</sup> This is reported in [GMP<sup>+</sup>09]. The *Flu Trend* reporting stopped in 2015.

<sup>2</sup> A popular trend in the information industry is to perform data cleaning and data integration as a preprocessing step, where the resulting data are stored in a data warehouse.

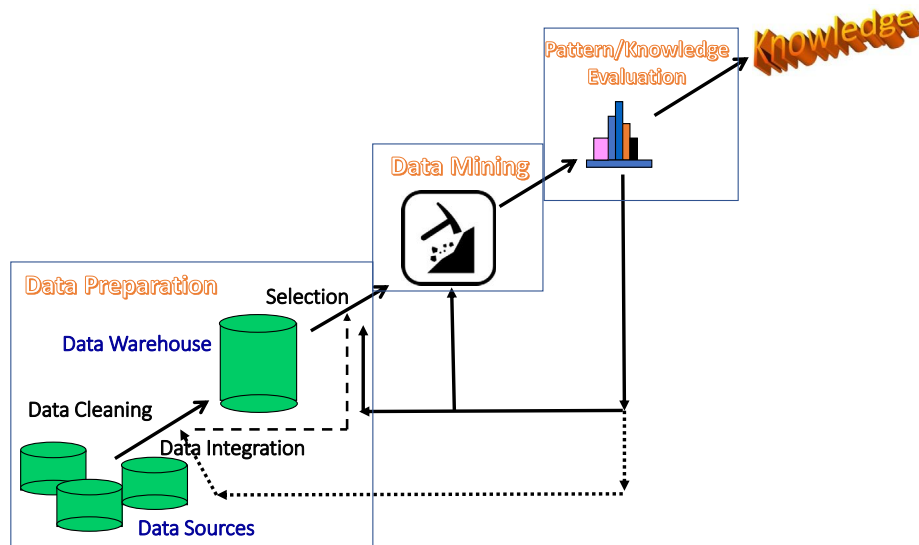


FIGURE 1.1

Data mining: An essential step in the process of knowledge discovery.

- c. **Data transformation** (where data are transformed and consolidated into forms appropriate for mining by performing summary or aggregation operations)<sup>3</sup>
- d. **Data selection** (where data relevant to the analysis task are retrieved from the database)
2. **Data mining** (an essential process where intelligent methods are applied to extract patterns or construct models)
3. **Pattern/model evaluation** (to identify the truly interesting patterns or models representing knowledge based on *interestingness measures*—see Section 1.4.7)
4. **Knowledge presentation** (where visualization and knowledge representation techniques are used to present mined knowledge to users)

Steps 1(a) through 1(d) are different forms of data preprocessing, where data are prepared for mining. The data mining step may interact with a user or a knowledge base. The interesting patterns are presented to the user and may be stored as new knowledge in the knowledge base.

The preceding view shows data mining as one step in the knowledge discovery process, albeit an essential one because it uncovers hidden patterns or models for evaluation. However, in industry, in media, and in the research milieu, the term *data mining* is often used to refer to the entire knowledge discovery process (perhaps because the term is shorter than *knowledge discovery from data*). Therefore, we adopt a broad view of data mining functionality: *Data mining is the process of discovering inter-*

<sup>3</sup> Data transformation and consolidation are often performed before the data selection process, particularly in the case of data warehousing. *Data reduction* may also be performed to obtain a smaller representation of the original data without sacrificing its integrity.

*esting patterns and knowledge from large amounts of data.* The data sources can include databases, data warehouses, the Web, other information repositories, or data that are streamed into the system dynamically.

---

### 1.3 Diversity of data types for data mining

As a general technology, data mining can be applied to any kind of data as long as the data are meaningful for a target application. However, different kinds of data may need rather different data mining methodologies, from simple to rather sophisticated, making data mining a rich and diverse field.

#### Structured vs. unstructured data

Based on whether data have clear structures, we can categorize data as *structured* vs. *unstructured data*.

Data stored in *relational databases*, *data cubes*, *data matrices*, and many *data warehouses* have uniform, record- or table-like structures, defined by their data dictionaries, with a fixed set of attributes (or fields, columns), each with a fixed set of value ranges and semantic meaning. These data sets are typical examples of highly structured data. In many real applications, such strict structural requirement can be relaxed in multiple ways to accommodate *semistructured* nature of the data, such as to allow a data object to contain a set value, a small set of heterogeneous typed values, or nested structures or to allow the structure of objects or subobjects to be defined flexibly and dynamically (e.g., XML structures).

There are many data sets that may not be as structured as relational tables or data matrices. However, they do have certain structures with clearly defined semantic meaning. For example, a *transactional data set* may contain a large set of transactions each containing a set of items. A *sequence data set* may contain a large set of sequences each containing an ordered set of elements that can in turn contain a set of items. Many application data sets, such as shopping transaction data, time-series data, gene or protein data, or Weblog data, belong to this category.

A more sophisticated type of semistructured data set is *graph or network data*, where a set of nodes are connected by a set of edges (also called links); and each node/link may have its own semantic description or substructures.

Each of such categories of structured and semistructured data sets may have special kinds of patterns or knowledge to be mined and many dedicated data mining methods, such as sequential pattern mining, graph pattern mining, and information network mining methods, have been developed to analyze such data sets.

Beyond such structured or semistructured data, there are also large amounts of unstructured data, such as text data and multimedia (e.g., audio, image, video) data. Although some studies treat them as one-dimensional or multidimensional byte streams, they do carry a lot of interesting semantics. Domain-specific methods have been developed to analyze such data in the fields of natural language understanding, text mining, computer vision, and pattern recognition. Moreover, recent advances on deep learning have made tremendous progress on processing text, image, and video data. Nevertheless, mining hidden structures from unstructured data may greatly help understand and make good use of such data.

The real-world data can often be a mixture of structured data, semistructured data, and unstructured data. For example, an online shopping website may host information for a large set of products, which

can be essentially structured data stored in a relational database, with a fixed set of fields on product name, price, specifications, and so on. However, some fields may essentially be text, image, and video data, such as product introduction, expert or user reviews, product images, and advertisement videos. Data mining methods are often developed for mining some particular type of data, and their results can be integrated and coordinated to serve the overall goal.

#### **Data associated with different applications**

Different applications may generate or need to handle very different data sets and require rather different data analysis methods. Thus when categorizing data sets for data mining, we should take specific applications into consideration.

Take sequence data as an example. *Biological sequences* such as DNA or protein sequences may have very different semantic meaning from *shopping transaction sequences* or *Web click streams*, calling for rather different sequence mining methods. A special kind of sequence data is time-series data where a *time-series* may contain an ordered set of numerical values with equal time interval, which is also rather different from shopping transaction sequences, which may not have fixed time gaps (a customer may shop at anytime she likes).

Data in some applications can be associated with spatial information, time information, or both, forming *spatial*, *temporal*, and *spatiotemporal data*, respectively. Special data mining methods, such as spatial data mining, temporal data mining, spatiotemporal data mining, or trajectory pattern mining, should be developed for mining such data sets as well.

For graph and network data, different applications may also need rather different data mining methods. For example, social networks (e.g., Facebook or LinkedIn data), computer communication networks, biological networks, and information networks (e.g., authors linking with keywords) may carry rather different semantics and require different mining methods.

Even for the same data set, finding different kinds of patterns or knowledge may require different data mining methods. For example, for the same set of software (source) programs, finding plagiarized subprogram modules or finding copy-and-paste bugs may need rather different data mining techniques.

Rich data types and diverse application requirements call for very diverse data mining methods. Thus data mining is a rich and fascinating research domain, with lots of new methods waiting to be studied and developed.

#### **Stored vs. streaming data**

Usually, data mining handles finite, stored data sets, such as those stored in various kinds of large data repositories. However, in some applications such as video surveillance or remote sensing, data may stream in dynamically and constantly, as infinite *data streams*. Mining stream data will require rather different methods than stored data, which may form another interesting theme in our study.

---

## **1.4 Mining various kinds of knowledge**

Different kinds of patterns and knowledge can be uncovered via data mining. In general, data mining tasks can be put into two categories: **descriptive data mining** and **predictive data mining**. Descriptive mining characterizes properties of the interested set of data, whereas predictive mining performs induction on the data set in order to make predictions.

In this section, we introduce different data mining tasks. These include multidimensional data summarization (Section 1.4.1); the mining of frequent patterns, associations, and correlations (Section 1.4.2); classification and regression (Section 1.4.3); cluster analysis (Section 1.4.4); and outlier analysis (Section 1.4.6). Different data mining functionalities generate different kinds of results that are often called patterns, models, or knowledge. In Section 1.4.7, we will also introduce the interestingness of a pattern or a model. In many cases, only interesting patterns or models will be considered as *knowledge*.

### 1.4.1 Multidimensional data summarization

It is often tedious for a user to go over the details of a large set of data. Thus it is desirable to automatically summarize an interested set of data and compare it with the contrasting sets at some high levels. Such summaritive description of an *interested set of data* is called **data summarization**. Data summarization can often be conducted in a multidimensional space. If the multidimensional space is well defined and frequently used, such as product category, producer, location, or time, massive amounts of data can be aggregated in the form of **data cubes** to facilitate user's drill-down or roll-up of the summarization space with mouse clicking. The output of such multidimensional summarization can be presented in various forms, such as **pie charts, bar charts, curves, multidimensional data cubes, and multidimensional tables**, including crosstabs.

For structured data, multidimensional aggregation methods have been developed to facilitate such precomputation or online computation of multidimensional aggregations using data cube technology, which will be discussed in Chapter 3. For unstructured data, such as text, this task becomes challenging. We will give a brief discussion of such research frontiers in our last chapter.

### 1.4.2 Mining frequent patterns, associations, and correlations

**Frequent patterns**, as the name suggests, are patterns that occur frequently in data. There are many kinds of frequent patterns, including frequent itemsets, frequent subsequences (also known as sequential patterns), and frequent substructures. A *frequent itemset* typically refers to a set of items that often appear together in a transactional data set—for example, milk and bread, which are frequently bought together in grocery stores by many customers. A frequently occurring subsequence, such as the pattern that customers tend to purchase first a laptop, followed by a computer bag, and then other accessories, is a (*frequent*) *sequential pattern*. A substructure can refer to different structural forms (e.g., graphs, trees, or lattices) that may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.

**Example 1.2. Association analysis.** Suppose that, a webstore manager wants to know which items are frequently purchased together (i.e., in the same transaction). An example of such a rule, mined from the transactional database, is

$$\text{buys}(X, \text{"computer"}) \Rightarrow \text{buys}(X, \text{"webcam"}) \quad [\text{support} = 1\%, \text{confidence} = 50\%],$$

where  $X$  is a variable representing a customer. A **confidence**, or certainty, of 50% means that if a customer buys a computer, there is a 50% chance that she will buy webcam as well. A 1% **support** means

that 1% of all the transactions under analysis show that computer and webcam are purchased together. This association rule involves a single attribute or predicate (i.e., *buys*) that repeats. Association rules that contain a single predicate are referred to as **single-dimensional association rules**. Dropping the predicate notation, the rule can be written simply as “*computer*  $\Rightarrow$  *webcam* [1%, 50%].”

Suppose, mining the same database generates another association rule:

$$\text{age}(X, \text{“20..29”}) \wedge \text{income}(X, \text{“40K..49K”}) \Rightarrow \text{buys}(X, \text{“laptop”}) \\ [\text{support} = 0.5\%, \text{confidence} = 60\%].$$

The rule indicates that of all its customers under study, 0.5% are 20 to 29 years old with an income of \$40,000 to \$49,000 and have purchased a laptop (computer). There is a 60% probability that a customer in this age and income group will purchase a laptop. Note that this is an association involving more than one attribute or predicate (i.e., *age*, *income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a **multidimensional association rule**.  $\square$

Typically, association rules are discarded as uninteresting if they do not satisfy both a **minimum support threshold** and a **minimum confidence threshold**. Additional analysis can be performed to uncover interesting statistical **correlations** between associated attribute–value pairs.

*Frequent itemset mining* is a fundamental form of frequent pattern mining. Mining frequent itemsets, associations, and correlations will be discussed in Chapter 4. Mining diverse kinds of frequent pattern, as well as mining sequential patterns and structured patterns, will be covered in Chapter 5.

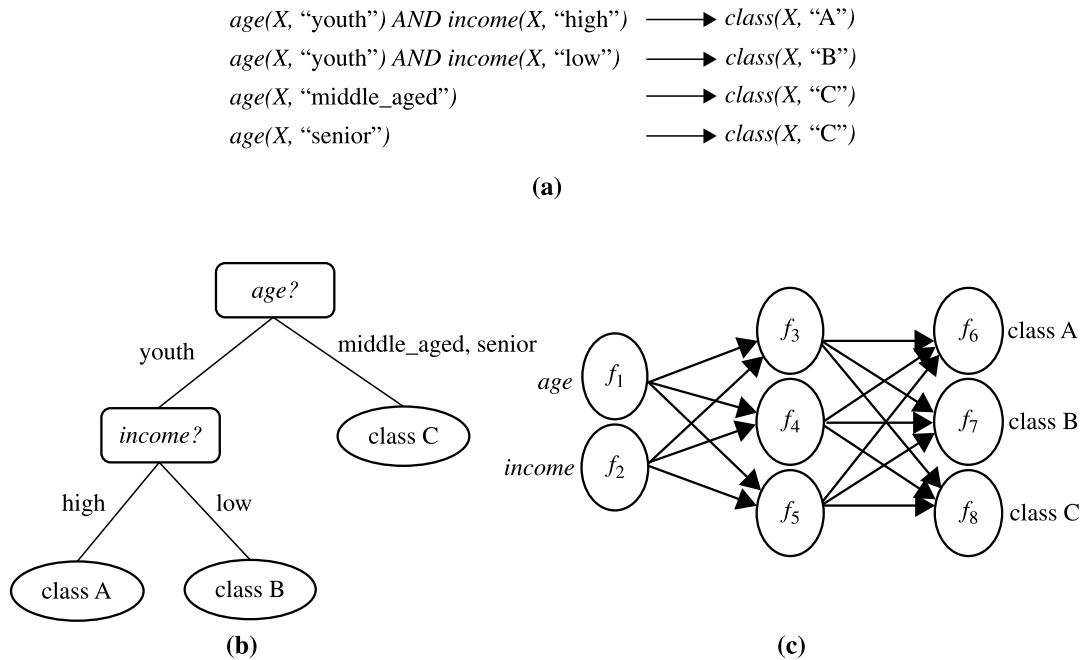
### 1.4.3 Classification and regression for predictive analysis

**Classification** is the process of finding a **model** (or function) that describes and distinguishes data classes or concepts. The model is derived based on the analysis of a set of **training data** (i.e., data objects for which the class labels are known). The model is used to predict the class labels of objects for which the class labels are unknown.

Depending on the classification methods, a derived model can be in various forms, such as a set of *classification rules* (i.e., *IF-THEN rules*), a *decision tree*, a *mathematical formula*, or a learned *neural network* (Fig. 1.2). A **decision tree** is a flowchart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can easily be converted to classification rules. A **neural network**, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and *k*-nearest-neighbor classification.

Whereas classification predicts categorical (discrete, unordered) labels, **regression** models continuous-valued functions. That is, regression is used to predict missing or unavailable *numerical data values* rather than (discrete) class labels. The term *prediction* refers to both numeric prediction and class label prediction. **Regression analysis** is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Regression also encompasses the identification of distribution *trends* based on the available data.

Classification and regression may need to be preceded by **feature selection** or **relevance analysis**, which attempts to identify attributes (often called *features*) that are significantly relevant to the clas-



**FIGURE 1.2**

A classification model can be represented in various forms: (a) IF-THEN rules, (b) a decision tree, or (c) a neural network.

sification and regression process. Such attributes will be selected for the classification and regression process. Other attributes, which are irrelevant, can then be excluded from consideration.

**Example 1.3. Classification and regression.** Suppose a webstore sales manager wants to classify a large set of items in the store, based on three kinds of responses to a sales campaign: *good response*, *mild response*, and *no response*. You want to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand*, *place\_made*, *type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set.

Suppose that the resulting classification is expressed as a decision tree. The decision tree, for instance, may identify *price* as being the first important factor that best distinguishes the three classes. Other features that help further distinguish objects of each class from one another include *brand* and *place\_made*. Such a decision tree may help the manager understand the impact of the given sales campaign and design a more effective campaign in the future.

Suppose instead, that rather than predicting categorical response labels for each store item, you would like to predict the amount of revenue that each item will generate during an upcoming sale, based on the previous sales data. This is an example of regression analysis because the regression model constructed will predict a continuous function (or ordered value.) □

Chapters 6 and 7 discuss classification in further detail. Regression analysis is covered lightly in these chapters since it is typically introduced in statistics courses. Sources for further information are given in the bibliographic notes.

### 1.4.4 Cluster analysis

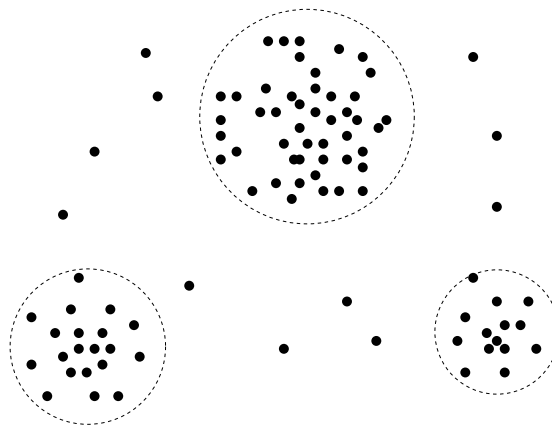
Unlike classification and regression, which analyze class-labeled (training) data sets, **cluster analysis** (also called **clustering**) groups data objects without consulting class labels. In many cases, class-labeled data may simply not exist at the beginning. Clustering can be used to generate class labels for a group of data. The objects are clustered or grouped based on the principle of *maximizing the intraclass similarity and minimizing the interclass similarity*. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are rather dissimilar to objects in other clusters. Each cluster so formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate **taxonomy formation**, that is, the organization of observations into a hierarchy of classes that group similar events together.

**Example 1.4. Cluster analysis.** Cluster analysis can be performed on the webstore customer data to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. Fig. 1.3 shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident. □

Cluster analysis forms the topic of Chapters 8 and 9.

### 1.4.5 Deep learning

For many data mining tasks, such as classification and clustering, a key step often lies in finding “good features,” which is a vector representation of each input data tuple. For example, in order to predict



**FIGURE 1.3**

A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters.



whether a regional disease outbreak will occur, one might have collected a large number of features from the health surveillance data, including the number of daily positive cases, number of daily tests, number of daily hospitalization, etc. Traditionally, this step (called feature engineering) often heavily relies on domain knowledge. Deep learning techniques provide an automatic way for feature engineering, which is capable of generating semantically meaningful features (e.g., weekly positive rate) from the initial input features. The generated features often significantly improve the mining performance (e.g., classification accuracy).

Deep learning is based on *neural networks*. A neural network is a set of connected input-output units where each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights to be able to predict the correct target values (e.g., class labels) of the input tuples. The core algorithm to learn such weights is called *backpropagation*, which searches for a set of weights and bias values that can model the data to minimize the loss function between the network's prediction and the actual target output of data tuples. Various forms (called architectures) of neural networks have been developed, including feed-forward neural networks, convolutional neural networks, recurrent neural networks, graph neural networks, and many more.

Deep learning has broad applications in computer vision, natural language processing, machine translation, social network analysis, and so on. It has been used in a variety of data mining tasks, including classification, clustering, outlier detection, and reinforcement learning.

Deep learning is the topic of Chapter 10.

### 1.4.6 Outlier analysis

A data set may contain objects that do not comply with the general behavior or model of the data. These data objects are **outliers**. Many data mining methods discard outliers as noise or exceptions. However, in some applications (e.g., fraud detection) the rare events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as **outlier analysis** or **anomaly mining**.

Outliers may be detected using statistical tests that assume a distribution or probability model for the data, or using distance measures where objects that are remote from any other cluster are considered outliers. Rather than using statistical or distance measures, density-based methods may identify outliers in a local region, although they look normal from a global statistical distribution view.

**Example 1.5. Outlier analysis.** Outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of unusually large amounts for a given account number in comparison to regular charges incurred by the same account. Outlier values may also be detected with respect to the locations and types of purchase, or the purchase frequency. □

Outlier analysis is discussed in Chapter 11.

### 1.4.7 Are all mining results interesting?

Data mining has the potential to generate a lot of results. A question can be, “*Are all of the mining results interesting?*”

This is a great question. Each type of data mining functions has its own measures on the evaluation of the mining quality. Nevertheless, there are some shared philosophy and principles.

Take pattern mining as an example. Pattern mining may generate thousands or even millions of patterns, or rules. You may wonder, “*What makes a pattern interesting? Can a data mining system generate all of the interesting patterns? Or, can the system generate only the interesting ones?*”

To answer the first question, a pattern is **interesting** if it is (1) *easily understood* by humans, (2) *valid* on new or test data with some degree of *certainty*, (3) *potentially useful*, and (4) *novel*. A pattern is also interesting if it validates a hypothesis that the user *sought to confirm*.

Several **objective measures of pattern interestingness** exist. These are based on the structure of discovered patterns and the statistics underlying them. An objective measure for association rules of the form  $X \Rightarrow Y$  is rule **support**, representing the percentage of transactions from a transaction database that the given rule satisfies. This is taken to be the probability  $P(X \cup Y)$ , where  $X \cup Y$  indicates that a transaction contains both  $X$  and  $Y$ , that is, the union of itemsets  $X$  and  $Y$ . Another objective measure for association rules is **confidence**, which assesses the degree of certainty of the detected association. This is taken to be the conditional probability  $P(Y|X)$ , that is, the probability that a transaction containing  $X$  also contains  $Y$ . More formally, support and confidence are defined as

$$\begin{aligned} \text{support}(X \Rightarrow Y) &= P(X \cup Y), \\ \text{confidence}(X \Rightarrow Y) &= P(Y|X). \end{aligned}$$

In general, each interestingness measure is associated with a threshold, which may be controlled by the user. For example, rules that do not satisfy a confidence threshold of, say, 50% can be considered uninteresting. Rules below the threshold likely reflect noise, exceptions, or minority cases and are probably of less value.

There are also other objective measures. For example, one may like set of items to be strongly correlated in an association rule. We will discuss such measures in the corresponding chapter.

Although objective measures help identify interesting patterns, they are often insufficient unless combined with subjective measures that reflect a particular user’s needs and interests. For example, patterns describing the characteristics of customers who shop frequently online should be interesting to the marketing manager, but may be of little interest to other analysts studying the same database for patterns on employee performance. Furthermore, many patterns that are interesting by objective standards may represent common sense and, therefore, are actually uninteresting.

**Subjective interestingness measures** are based on user beliefs in the data. These measures find patterns interesting if the patterns are **unexpected** (contradicting a user’s belief) or offer strategic information on which the user can act. In the latter case, such patterns are referred to as **actionable**. For example, patterns like “a large earthquake often follows a cluster of small quakes” may be highly actionable if users can act on the information to save lives. Patterns that are **expected** can be interesting if they confirm a hypothesis that the user wishes to validate or they resemble a user’s hunch.

The second question—“*Can a data mining system generate all of the interesting patterns?*”—refers to the **completeness** of a data mining algorithm. It is often unrealistic and inefficient for a pattern mining system to generate all possible patterns since there could be a very large number of them. However, one may also worry whether one may miss some important ones if the system stops short. To solve this dilemma, user-provided constraints and interestingness measures should be used to focus the search. With well-defined interesting measures and user-provided constraints, it is quite realistic to ensure the completeness of pattern mining. The methods involved are examined in detail in Chapter 4.

Finally, the third question—“*Can a data mining system generate only interesting patterns?*”—is an optimization problem in data mining. It is highly desirable for a data mining system to generate only

interesting patterns. This would be efficient for both the data mining system and the user because the system may spend much less time to generate much fewer but interesting patterns, whereas the user will not need to sift through a large number of patterns to identify the truly interesting ones. Constraint-based pattern mining described in Chapter 5 is a good example in this direction.

Methods to assess the quality or interestingness of data mining results, and how to use them to improve data mining efficiency, are discussed throughout the book.

## 1.5 Data mining: confluence of multiple disciplines

As a discipline that studies efficient and effective methods for uncovering patterns and knowledge from various kinds of massive data sets for many applications, data mining naturally serves a confluence of multiple disciplines including machine learning, statistics, pattern recognition, natural language processing, database technology, visualization and human computer interaction (HCI), algorithms, high-performance computing, social sciences, and many application domains (Fig. 1.4). The interdisciplinary nature of data mining research and development contributes significantly to the success of data mining and its extensive applications. On the other hand, data mining is not only nurtured from the knowledge and development of these disciplines, the dedicated research, development, and applications of data mining on various kinds of big data may have substantially impacted the development of these disciplines in recent years as well. In this section, we discuss several disciplines that strongly impact and actively interact with the research, development, and applications of data mining.

### 1.5.1 Statistics and data mining

**Statistics** studies the collection, analysis, interpretation or explanation, and presentation of data. Data mining has an inherent connection with statistics.

A **statistical model** is a set of mathematical functions that describe the behavior of the objects in a target class in terms of random variables and their associated probability distributions. Statistical

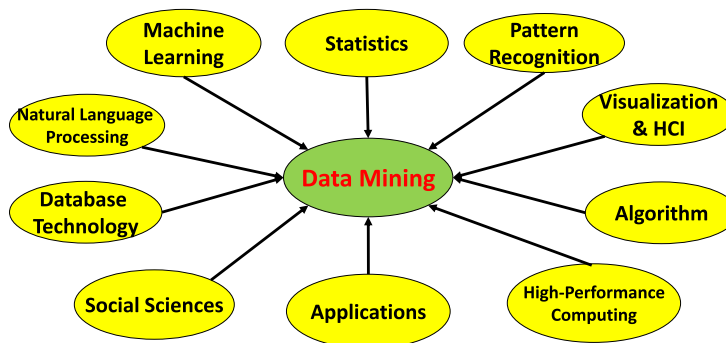


FIGURE 1.4

Data mining: Confluence of multiple disciplines.

models are widely used to model data and data classes. For example, in data mining tasks such as data characterization and classification, statistical models of target classes can be built. In other words, such statistical models can be the outcome of a data mining task. Alternatively, data mining tasks can be built on top of statistical models. For example, we can use statistics to model noise and missing data values. Then, when mining patterns in a large data set, the data mining process can use the model to help identify and handle noisy or missing values in the data.

Statistics research develops tools for prediction and forecasting using data and statistical models. Statistical methods can be used to summarize or describe a collection of data. Basic **statistical descriptions** of data are introduced in Chapter 2. Statistics is useful for mining various patterns from data and for understanding the underlying mechanisms generating and affecting the patterns. **Inferential statistics** (or **predictive statistics**) models data in a way that accounts for randomness and uncertainty in the observations and is used to draw inferences about the process or population under investigation.

Statistical methods can also be used to verify data mining results. For example, after a classification or prediction model is mined, the model should be verified by statistical hypothesis testing. A **statistical hypothesis test** (sometimes called *confirmatory data analysis*) makes statistical decisions using experimental data. A result is called *statistically significant* if it is unlikely to have occurred by chance. If the classification or prediction model holds, then the descriptive statistics of the model increases the soundness of the model.

Applying statistical methods in data mining is far from trivial. Often, a serious challenge is how to scale up a statistical method over a large data set. Many statistical methods have high complexity in computation. When such methods are applied on large data sets that are also distributed on multiple logical or physical sites, algorithms should be carefully designed and tuned to reduce the computational cost. This challenge becomes even tougher for online applications, such as online query suggestions in search engines, where data mining is required to continuously handle fast, real-time data streams.

Data mining research has developed many scalable and effective solutions for the analysis of massive data sets and data streams. Moreover, different kinds of data sets and different applications may require rather different analysis methods. Effective solutions have been proposed and tested, which leads to many new, scalable data mining-based statistical analysis methods.

## 1.5.2 Machine learning and data mining

**Machine learning** investigates how computers can learn (or improve their performance) based on data. Machine learning is a fast-growing discipline, with many new methodologies and applications developed in recent years, from support vector machines to probabilistic graphical models and deep learning, which we will cover in this book.

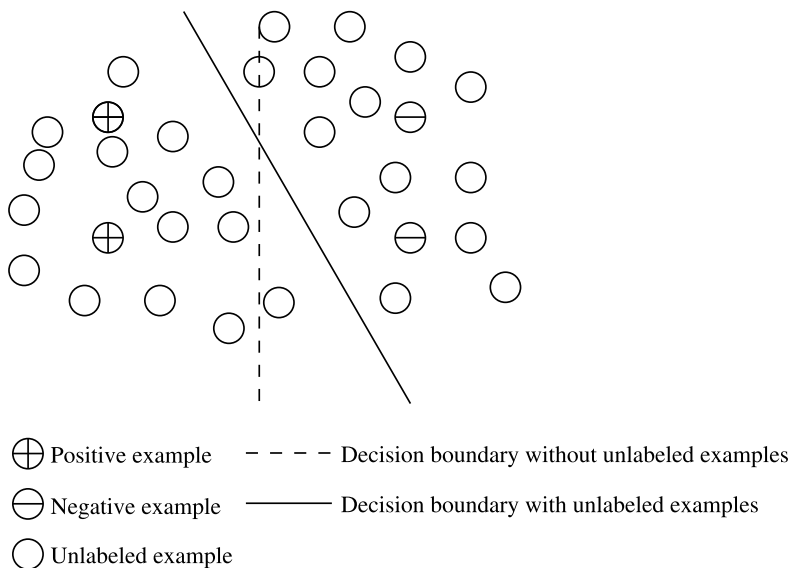
In general, machine learning addresses two classical problems: *supervised learning* and *unsupervised learning*.

- **Supervised learning:** A classic example of supervised learning is classification. The supervision in the learning comes from the labeled examples in the training data set. For example, to automatically recognize handwritten postal codes on mails, the learning system takes a set of handwritten postal code images and their corresponding machine-readable translations as the training examples, and learns (i.e., computes) a classification model.
- **Unsupervised learning:** A classic example of unsupervised learning is clustering. The learning process is unsupervised since the input examples are not class-labeled. Typically, we may use clustering

to discover groups within the data. For example, an unsupervised learning method can take, as input, a set of images of handwritten digits. Suppose that it finds 10 clusters of data. These clusters may hopefully correspond to the 10 distinct digits of 0 to 9, respectively. However, since the training data are not labeled, the learned model cannot tell us the semantic meaning of the clusters found.

As to these two basic problems, data mining and machine learning do share many similarities. However, data mining differs from machine learning in several major aspects. First, even on similar tasks like classification and clustering, data mining often works on very large data sets, or even on infinite data streams, scalability can be an important concern, and many efficient and highly scalable data mining algorithms or stream mining algorithms have to be developed to accomplish such tasks.

Second, in many data mining problems, the data sets are usually large, but the training data can still be rather small since it is expensive for experts to provide quality labels for many examples. Therefore, data mining has to put a lot of effort on developing *weakly supervised methods*. These include methodologies like *semisupervised learning* with a small set of labeled data but a large set of unlabeled data (with the idea sketched in Fig. 1.5), *integration or ensemble of multiple weak models* obtained from nonexperts (e.g., those obtained by crowd-sourcing), *distant supervision*, such as using popularly available and general (but distantly relevant to the problem to be solved) knowledge-bases (e.g., wikipedia, DBpedia), *actively learning* by carefully selecting examples to ask human experts, or *transfer learning* by integrating models learned from similar problem domains. Data mining has been extending such weakly supervised methods for constructing quality classification models on large data sets with a very limited set of high quality training data.



**FIGURE 1.5**

Semisupervised learning.

Third, machine learning methods may not be able to handle many kinds of knowledge discovery problems on big data. On the other hand, data mining, developing effective solutions for concrete application problems, goes deep in the problem domain, and expands far beyond the scope covered by machine learning. For example, many application problems, such as business transaction data analysis, software program execution sequence analysis, and chemical and biological structural analysis, need effective methods for mining frequent patterns, sequential patterns, and structured patterns. Data mining research has generated many scalable, effective, and diverse mining methods for such tasks. As another example, the analysis of large-scale social and information networks poses many challenging problems that may not fit the typical scope of many machine learning methods due to the information interaction across links and nodes in such networks. Data mining has developed a lot of interesting solutions to such problems.

From this point of view, data mining and machine learning are two different but closely related disciplines. Data mining dives deep into concrete, data-intensive application domains, does not confine itself to a single problem-solving methodology, and develops concrete (sometimes rather novel), effective and scalable solutions for many challenging application problems. It is a young, broad, and promising research discipline for many researchers and practitioners to study and work on.

### 1.5.3 Database technology and data mining

**Database system research** focuses on the creation, maintenance, and use of databases for organizations and end-users. Particularly, database system researchers have established well-recognized principles in data models, query languages, query processing and optimization, data storage, and indexing methods. Database technology is well known for its scalability in processing very large, relatively structured data sets.

Many data mining tasks need to handle large data sets or even real-time, fast streaming data. Data mining can make good use of scalable database technologies to achieve high efficiency and scalability on large data sets. Moreover, data mining tasks can be used to extend the capability of existing database systems to satisfy users' sophisticated data analysis requirements.

Recent database systems have built systematic data analysis capabilities on database data using data warehousing and data mining facilities. A **data warehouse** integrates data originated from multiple sources and various timeframes. It consolidates data in multidimensional space to form partially materialized data cubes. The data cube model not only facilitates online analytical processing (OLAP) in multidimensional databases but also promotes *multidimensional data mining*, which will be further discussed in future chapters.

### 1.5.4 Data mining and data science

With the tremendous amount of data in almost every discipline and various kinds of applications, big data and data science have become buzzwords in recent years. **Big data** generally refers to huge amounts of structured and unstructured data of various forms, and **data science** is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from massive data of various forms. Clearly, data mining plays an essential role in data science.

For most people, data science is a concept that unifies statistics, machine learning, data mining, and their related methods in order to understand and analyze massive data. It employs techniques and theories drawn from many fields within the context of mathematics, statistics, information science, and

computer science. For many industry people, the term “data science” often refers to business analytics, business intelligence, predictive modeling, or any meaningful use of data, and is being taken as a glamorized term to re-brand statistics, data mining, machine learning, or any kind of data analytics. So far, there exists no consensus on a definition or suitable curriculum contents in data science degree programs of many universities. Nonetheless, most universities take basic knowledge generated in statistics, machine learning, data mining, database, and human computer interaction as the core curriculum in data science education.

In 1990s, the late Turing award winner Jim Gray envisioned data science as the “fourth paradigm” of science (i.e., from empirical to theoretical, computational, and now data-driven) and asserted that “everything about science is changing because of the impact of information technology” and the emergence of massive data. So there is no wonder that data science, big data, and data mining are closely interrelated and represent an inevitable trend in science and technology developments.

### 1.5.5 Data mining and other disciplines

Besides statistics, machine learning, and database technology, data mining has close relationships with many other disciplines as well.

The majority of the real-world data are unstructured, in the form of natural language text, images, or audio-video data. Therefore, natural language processing, computer vision, pattern recognition, audio-video signal processing, and information retrieval will offer critical help at handling such data. Actually, handling any special kinds of data will need a lot of domain knowledge to be integrated into the data mining algorithm design. For example, mining biomedical data will need the integration of knowledge from biological sciences, medical sciences, and bioinformatics. Mining geospatial data will need much knowledge and techniques from geography and geospatial data sciences. Mining software bugs in large software programs will need to integrate software engineering with data mining. Mining social media and social networks will need knowledge and skills from social sciences and network sciences. Such examples can go on and on since data mining will penetrate almost every application domain.

One major challenge in data mining is efficiency and scalability since we often have to handle huge amounts of data with critical time and resource constraints. Data mining is critically connected with efficient algorithm design such as low-complexity, incremental, and streaming data mining algorithms. It often needs to explore high performance computation, parallel computation, and distributed computation, with advanced hardware and cloud computing or cluster computing environment.

Data mining is also closely tied with human-computer interaction. Users need to interact with a data mining system or process in an effective way, telling the system what to mine, how to incorporate background knowledge, how to mine, and how to present the mining results in an easy-to-understand (e.g., by interpretation and visualization) and easy-to-interact way (e.g., with friendly graphic user interface and interactive mining).

Actually, nowadays, there are not only many interactive data mining systems but also many more data mining functions hidden in various kinds of application programs. It is unrealistic to expect everyone in our society to understand and master data mining techniques. It is also forbidden for industries to expose their large data sets. Many systems have data mining functions built within so that people can perform data mining or use data mining results simply by mouse clicking. For example, intelligent search engines and online retails perform such **invisible data mining** by collecting their data and user’s search or purchase history, incorporating data mining into their components to improve their performance, functionality, and user satisfaction. When your grandma shops online, she may be surprised

when receiving some smart recommendations. This could likely be resulted from such invisible data mining.

---

## 1.6 Data mining and applications

*Where there are data, there are data mining applications*

As a highly application-driven discipline, data mining has seen great successes in many applications. It is impossible to enumerate all applications where data mining plays a critical role. Presentations of data mining in knowledge-intensive application domains, such as bioinformatics and software engineering, require more in-depth treatment and are beyond the scope of this book. To demonstrate the importance of applications of data mining, we briefly discuss a few highly successful and popular application examples of data mining: *business intelligence; search engines; social media and social networks; and biology, medical science, and health care.*

### **Business intelligence**

It is critical for businesses to acquire a better understanding of the commercial context of their organization, such as their customers, the market, supply and resources, and competitors. **Business intelligence (BI)** technologies provide historical, current, and predictive views of business operations. Examples include reporting, online analytical processing, business performance management, competitive intelligence, benchmarking, and predictive analytics.

*“How important is data mining in business intelligence?”* Without data mining, many businesses may not be able to perform effective market analysis, compare customer feedback on similar products, discover the strengths and weaknesses of their competitors, retain highly valuable customers, and make smart business decisions.

Clearly, data mining is the core of business intelligence. Online analytical processing tools in business intelligence rely on data warehousing and multidimensional data mining. Classification and prediction techniques are the core of predictive analytics in business intelligence, for which there are many applications in analyzing markets, supplies, and sales. Moreover, clustering plays a central role in customer relationship management, which groups customers based on their similarities. Using multidimensional summarization techniques, we can better understand features of each customer group and develop customized customer reward programs.

### **Web search engines**

A **Web search engine** is a specialized computer server that searches for information on the Web. The search results of a user query are often returned as a list (sometimes called *hits*). The hits may consist of web pages, images, and other types of files. Some search engines also search and return data available in public databases or open directories. Search engines differ from **web directories** in that web directories are maintained by human editors, whereas search engines operate algorithmically or by a mixture of algorithmic and human input.

Search engines pose grand challenges to data mining. First, they have to handle a huge and ever-growing amount of data. Typically, such data cannot be processed using one or a few machines. Instead, search engines often need to use *computer clouds*, which consist of thousands or even hundreds of thou-



sands of computers that collaboratively mine the huge amount of data. Scaling up data mining methods over computer clouds and large distributed data sets is an area of active research and development.

Second, Web search engines often have to deal with online data. A search engine may be able to afford constructing a model offline on huge datasets. To do this, it may construct a query classifier that assigns a search query to predefined categories based on the query topic (i.e., whether the search query “apple” is meant to retrieve information about a fruit or a brand of computers). Even if a model is constructed offline, the adaptation of the model online must be fast enough to answer user queries in real time.

Another challenge is maintaining and incrementally updating a model on fast-growing data streams. For example, a query classifier may need to be incrementally maintained continuously since new queries keep emerging and predefined categories and the data distribution may change. Most of the existing model training methods are offline and static and thus cannot be used in such a scenario.

Third, Web search engines often have to deal with queries that are asked only a very small number of times. Suppose a search engine wants to provide *context-aware* query recommendations. That is, when a user poses a query, the search engine tries to infer the context of the query using the user’s profile and his query history in order to return more customized answers within a small fraction of a second. However, although the total number of queries asked can be huge, many queries may be asked only once or a few times. Such severely skewed data are challenging for many data mining and machine learning methods.

### ***Social media and social networks***

The prevalence of social media and social networks has fundamentally changed our life and the way we exchange information and socialize nowadays. With tremendous amounts of social media and social network data available, it is critical to analyze such data to extract actionable patterns and trends from social media and social network data.

Social media mining is to sift through massive amounts of social media data (e.g., on social media usage, online social behaviors, connections between individuals, online shopping behavior, content exchange, etc.) in order to discern patterns and trends. These patterns and trends have been used for social event detection, public health monitoring and surveillance, sentiment analysis in social media, recommendation in social media, information provenance, social media trustability analysis, and social spammer detection.

Social network mining is to investigate social network structures and the information associated with such networks through the use of networks and graph theory and data mining methods. The social network structures are characterized in terms of nodes (individual actors, people, or things within the network) and the ties, edges, or links (relationships or interactions) that connect them. Examples of social structures commonly visualized through social network analysis include social media networks, memes spread, friendship and acquaintance networks, collaboration graphs, kinship, disease transmission, and sexual relationships. These networks are often visualized through sociograms in which nodes are represented as points and ties are represented as lines.

Social network mining has been used to detect hidden communities, uncover the evolution and dynamics of social networks, compute network measures (e.g., centrality, transitivity, reciprocity, balance, status, and similarity), analyze how information propagates in social media sites, measure and model node/substructure influence and homophily, and conduct location-based social network analysis.

Social media mining and social network mining are important applications of data mining.

### ***Biology, medical science, and health care***

Biology, medical science and health care have also been generating massive data at exponential scale. Biomedical data take many forms, from “omics” to imaging, mobile health, and electronic health records. With the availability of more efficient digital collection methods, biomedical scientists and clinicians now find themselves confronting ever larger sets of data and trying to devise creative ways to sift through this mountain of data and make sense of it. Indeed, data that used to be considered large now seems small as the amount of data now being collected in a single day by an investigator can surpass what might have been generated over his/her career even a decade ago. This deluge of biomedical information requires new thinking about how data can be managed and analyzed to further scientific understanding and for improving healthcare.

Biomedical data mining involves many challenging data mining tasks, including mining massive genomic and proteomic sequence data, mining frequent subgraph patterns for classifying biological data, mining regulatory networks, characterization and prediction of protein-protein interactions, classification and predictive analysis of medical images, biological text mining, biological information network construction from biotext data, mining electronic health records, and mining biomedical networks.

---

## **1.7 Data mining and society**

With data mining penetrating our everyday lives, it is important to study the impact of data mining on society. How can we use data mining technology to benefit society? How can we guard against its misuse? The improper disclosure or use of data and the potential violation of individual privacy and data protection rights are areas of concern that need to be addressed.

Data mining will help scientific discovery, business management, economy recovery, and security protection (e.g., the real-time discovery of intruders and cyberattacks). However, it also poses the risk of unintentionally disclosing some confidential business or government information and disclosing an individual’s personal information. Studies on data security in data mining and privacy-preserving data publishing and data mining are important, ongoing research theme. The philosophy is to observe data sensitivity and preserve data security and people’s privacy while performing successful data mining.

These issues and many additional ones relating to the research, development, and application of data mining will be discussed throughout the book.

---

## **1.8 Summary**

- *Necessity is the mother of invention.* With the mounting growth of data in every application, data mining meets the imminent need for effective, scalable, and flexible data analysis in our society. Data mining can be considered as a natural evolution of information technology and a confluence of several related disciplines and application domains.
- **Data mining** is the process of discovering interesting patterns and knowledge from massive amounts of data. As a *knowledge discovery process*, it typically involves data cleaning, data integration, data selection, data transformation, pattern and model discovery, pattern or model evaluation, and knowledge presentation.

- A pattern or model is *interesting* if it is valid on test data with some degree of certainty, novel, potentially useful (e.g., can be acted on or validates a hunch about which the user was curious), and easily understood by humans. Interesting patterns represent knowledge. Measures of **pattern interestingness**, either *objective* or *subjective*, can be used to guide the discovery process.
- Data mining can be conducted on any kind of **data** as long as the data are meaningful for a target application, such as structured data (e.g., relational database, transaction data) and unstructured data (e.g., text and multimedia data), as well as data associated with different applications. Data can also be categorized as stored vs. stream data, whereas the latter may need to explore special stream mining algorithms.
- **Data mining functionalities** are used to specify the kinds of patterns or **knowledge** to be found in data mining tasks. The functionalities include characterization and discrimination; the mining of frequent patterns, associations, and correlations; classification and regression; deep learning; cluster analysis; and outlier detection. As new types of data, new applications, and new analysis demands continue to emerge, there is no doubt we will see more and more novel data mining tasks in the future.
- Data mining, is a confluence of multiple disciplines but it has its unique research focus, dedicated to many advanced applications. We study the close relationships of data mining with statistics, machine learning, database technology, and many other disciplines.
- Data mining has many successful **applications**, such as business intelligence, Web search, bioinformatics, health informatics, finance, digital libraries, and digital governments.
- Data mining may already have its strong impact on the society and the study of such impact, such as how to ensure the effectiveness of data mining and in the meantime ensure the data privacy and security, has become an important issue in research.

---

## 1.9 Exercises

- 1.1. What is *data mining*? In your answer, address the following:
  - a. Is it a simple transformation or application of technology developed from *databases*, *statistics*, *machine learning*, and *pattern recognition*?
  - b. Someone believes that data mining is an inevitable result of the evolution of information technology. If you are a database researcher, show data mining is resulted from a nature evolution of database technology. What about if you are a machine learner researcher, or a statistician?
  - c. Describe the steps involved in data mining when viewed as a process of knowledge discovery.
- 1.2. Define each of the following *data mining functionalities*: association and correlation analysis, classification, regression, clustering, deep learning, and outlier analysis. Give examples of each data mining functionality, using a real-life database that you are familiar with.
- 1.3. Present an example where data mining is crucial to the success of a business. What *data mining functionalities* does this business need (e.g., think of the kinds of patterns that could be mined)? Can such patterns be generated alternatively by data query processing or simple statistical analysis?
- 1.4. Explain the difference and similarity between correlation analysis and classification, between classification and clustering, and between classification and regression.

- 1.5. Based on your observations, describe another possible kind of knowledge that needs to be discovered by data mining methods but has not been listed in this chapter. Does it require a mining methodology that is quite different from those outlined in this chapter?
- 1.6. *Outliers* are often discarded as noise. However, one person's garbage could be another's treasure. For example, exceptions in credit card transactions can help us detect the fraudulent use of credit cards. Using fraud detection as an example, propose two methods that can be used to detect outliers and discuss which one is more reliable.
- 1.7. What are the major challenges of mining a huge amount of data (e.g., billions of tuples) in comparison with mining a small amount of data (e.g., data set of a few hundred tuples)?
- 1.8. Outline the major research challenges of data mining in one specific application domain, such as stream/sensor data analysis, spatiotemporal data analysis, or bioinformatics.

---

## 1.10 Bibliographic notes

The book *Knowledge Discovery in Databases*, edited by Piatetsky-Shapiro and Frawley [PSF91], is an early collection of research papers on knowledge discovery from data. The book *Advances in Knowledge Discovery and Data Mining*, edited by Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy [FPSSe96], is another early collection of research results on knowledge discovery and data mining. There have been many data mining textbook or research books published since then. Some popular ones include *Data Mining: Practical Machine Learning Tools and Techniques* (4th ed.) by Witten, Frank, Hall and Pal [WFHP16]; *Data Mining: Concepts and Techniques* (3rd ed.) by Han and Kamber and Pei [HKP11], *Introduction to Data Mining* (2nd ed.) by Tan, Steinbach, Karpatne, and Kumar [TSKK18]; *Data Mining: The Textbook* [Agg15b]; *Data Mining and Machine Learning: Fundamental Concepts and Algorithms* (2nd ed.) by Zaki and Meira [ZJ20]; *Mining of Massive Datasets* (3rd ed.) by Leskovec, Rajaraman and Ullman [ZJ20]; *The Elements of Statistical Learning* (2nd ed.) by Hastie, Tibshirani, and Friedman [HTF09]; *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management* (3rd ed.) by Linoff and Berry [LB11]; *Principles of Data Mining (Adaptive Computation and Machine Learning)* by Hand, Mannila, and Smyth [HMS01]; *Mining the Web: Discovering Knowledge from Hypertext Data* by Chakrabarti [Cha03]; *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data* by Liu [Liu06]; and *Data Mining: Multimedia, Soft Computing, and Bioinformatics* by Mitra and Acharya [MA03].

There are also numerous books that contain collections of papers or chapters on particular aspects of knowledge discovery, such as cluster analysis, outlier detection, classification, association mining, and mining particular kinds of data, such as mining text data, multimedia data, relational data, geospatial data, social and information network data, and social media data. However, this list has gone very long over the years and we will not list them individually. There are numerous tutorial notes on data mining in major data mining, database, machine learning, statistics, and Web technology conferences.

*KDNuggets* is a regular electronic newsletter containing information relevant to knowledge discovery and data mining, moderated by Piatetsky-Shapiro since 1991. The Internet site *KDNuggets* (<https://www.kdnuggets.com>) contains a good collection of KDD-related information.

The data mining community started its first international conference on knowledge discovery and data mining in 1995. The conference evolved from the four international workshops on knowledge discovery in databases, held from 1989 to 1994. ACM-SIGKDD, a Special Interest Group on Knowl-

edge Discovery in Databases was set up under ACM in 1998 and has been organizing the international conferences on knowledge discovery and data mining since 1999. IEEE Computer Science Society has organized its annual data mining conference, International Conference on Data Mining (ICDM), since 2001. SIAM (Society on Industrial and Applied Mathematics) has organized its annual data mining conference, SIAM Data Mining Conference (SDM), since 2002. A dedicated journal, *Data Mining and Knowledge Discovery*, published by Springer, has been available since 1997. An ACM journal, *ACM Transactions on Knowledge Discovery from Data*, published its first volume in 2007.

ACM-SIGKDD also publishes a bi-annual newsletter, *SIGKDD Explorations*. There are a few other international or regional conferences on data mining, such as the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD), the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), and the International Conference on Web Search and Data Mining (WSDM).

Research in data mining has also been popularly published in many textbooks, research books, conferences, and journals on data mining, database, statistics, machine learning, and data visualization.

# Data, measurements, and data preprocessing

**To conduct successful data mining, the first important thing is to get familiar with your data.** You may want to know the following: What are the types of *attributes* or fields that make up your data? What kind of values does each attribute have? How are the values distributed? How can we measure the similarity of some data objects with respect to others? Gaining such insights into the data will help with the subsequent analysis. Moreover, real-world data are typically noisy, enormous in volume (often several gigabytes or more), and may originate from a hodgepodge of heterogeneous sources. How can we measure the quality of data? How can we clean and integrate data from multiple heterogeneous sources? How can we normalize, compress, or transform the data? How can we reduce the dimensionality of data to help subsequent analysis? These are the tasks of this chapter.

We begin in Section 2.1 by studying the various attribute types. These include nominal attributes, binary attributes, ordinal attributes, and numeric attributes. Basic *statistical descriptions* can be used to learn more about each attribute's values, as described in Section 2.2. Given a *temperature* attribute, for example, we can determine its *mean* (average value), *median* (middle value), and *mode* (most common value). These are *measures of central tendency*, which give us an idea of the “middle” or center of a distribution. Knowing such basic statistics regarding each attribute makes it easier to fill in missing values, smooth noisy values, and spot outliers during data preprocessing. Knowledge of the attributes and attribute values can also help in fixing inconsistencies incurred during data integration. Plotting the measures of central tendency shows us if the data are symmetric or skewed. Quantile plots, histograms, and scatter plots are other graphic displays of basic statistical descriptions. These can all be useful during data preprocessing and can provide insight into areas for mining.

We may also want to examine how *similar* (or *dissimilar*) data objects are. For example, suppose we have a database where the data objects are patients, described by their symptoms. We may want to find the similarity or dissimilarity between individual patients. Such information can allow us to find clusters of like patients within the data set. The similarity (or dissimilarity) between objects may also be used to detect outliers in the data, or to perform nearest-neighbor classification. There are many measures for assessing similarity and dissimilarity. In general, such measures are referred to as *proximity measures*. Think of the proximity of two objects as a function of the *distance* between their attribute values, although proximity can also be calculated based on probabilities rather than actual distance. Measures of data proximity are described in Section 2.3.

Finally, we will discuss data preprocessing, which is to address today's real-world challenges: data sets are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size and their likely origin from multiple, heterogeneous sources. Low-quality data will lead to low-quality mining results. Huge efforts need to be paid to preprocess the data to enhance the quality of data for effective mining. Section 2.4 is on *data cleaning* and *data integration*. The former is to remove noise and correct inconsistencies in data, whereas the latter is to merge data from multiple sources into a

coherent data store such as a data warehouse. Section 2.5 is on *data transformation*, which transforms or consolidates data into forms appropriate for mining. That is, it can make the resulting mining process be more efficient, and the patterns found be easier to understand. Various strategies for data transformation have been developed. For example, *data normalization* scales the attribute data to fall within a smaller range, like 0.0 to 1.0; *data discretization* replaces the raw values of a numeric attribute by interval labels or conceptual labels; and *data reduction* techniques (e.g., *compression* and *sampling*) transform the input data to a reduced representation and can improve the accuracy and efficiency of mining algorithms involving distance measurements. Last, Section 2.6 is on *dimensionality reduction*, which is the process of reducing the number of random variables or attributes under consideration. Please note that various data preprocessing techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a *date* field to a common format.

---

## 2.1 Data types

Data sets are made up of data objects. A **data object** represents an entity—in a sales database, the objects may be customers, store items, and sales; in a medical database, the objects may be patients; in a university database, the objects may be students, professors, and courses. Data objects are typically described by attributes. Data objects can also be referred to as *samples*, *examples*, *instances*, *data points*, or *objects*. If the data objects are stored in a database, they are *data tuples*. That is, the rows of a database correspond to the data objects, and the columns correspond to the attributes. In this section, we define attributes and look at the various attribute types.

*What is an attribute?* An **attribute** is a data field, representing a characteristic or feature of a data object. The nouns *attribute*, *dimension*, *feature*, and *variable* are often used interchangeably in the literature. The term *dimension* is commonly used in data warehousing. Machine learning literature tends to use the term *feature*, whereas statisticians prefer the term *variable*. Data mining and database professionals commonly use the term *attribute*, and we do here as well. Attributes describing a customer object can include, for example, *customer\_ID*, *name*, and *address*. Observed values for a given attribute are known as *observations*. A set of attributes used to describe a given object is called an *attribute vector* (or *feature vector*). The distribution of data involving one attribute (or variable) is called *univariate*. A *bivariate* distribution involves two attributes, and so on.

The **type** of an attribute is determined by the set of possible values—nominal, binary, ordinal, or numeric—the attribute can have. In the following subsections, we introduce each type.

### 2.1.1 Nominal attributes

Nominal means “relating to names.” The values of a **nominal attribute** are symbols or *names of things*. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as **categorical**. The values do not have any meaningful order. In computer science, the values are also known as *enumerations*.

**Example 2.1. Nominal attributes.** Suppose that *hair\_color* and *marital\_status* are two attributes describing *person* objects. In our application, possible values for *hair\_color* are *black*, *brown*, *blond*, *red*, *auburn*, *gray*, and *white*. The attribute *marital\_status* can take on the values *single*, *married*, *divorced*,

and *widowed*. Both *hair\_color* and *marital\_status* are nominal attributes. Another example of a nominal attribute is *occupation*, with the values *teacher*, *dentist*, *programmer*, *farmer*, and so on. □

Although we said that the values of a nominal attribute are symbols or “names of things,” it is possible to represent such symbols or “names” with numbers. With *hair\_color*, for instance, we can assign a code of 0 for *black*, 1 for *brown*, and so on. Another example is *customer\_ID*, with possible values that are all numeric. However, in such cases, the numbers are not intended to be used quantitatively. That is, mathematical operations on values of nominal attributes are not meaningful. It makes no sense to subtract one customer ID number from another, unlike, say, subtracting an age value from another (where *age* is a numeric attribute). Even though a nominal attribute may have integers as values, it is not considered a numeric attribute because the integers are not meant to be used quantitatively. We will say more on numeric attributes in Section 2.1.4.

Because nominal attribute values do not have any meaningful order about them and are not quantitative, it makes no sense to find the mean (average) value or median (middle) value for such an attribute, given a set of objects. One thing that is of interest, however, is the attribute’s most commonly occurring value. This value, known as the *mode*, is one of the measures of central tendency. You will learn about measures of central tendency in Section 2.2.

### 2.1.2 Binary attributes

A **binary attribute** is a nominal attribute with only two categories or states: 0 or 1, where 0 typically means that the attribute is absent, and 1 means that it is present. Binary attributes are referred to as **Boolean** if the two states correspond to *true* and *false*.

**Example 2.2. Binary attributes.** Given the attribute *smoker* describing a *patient* object, 1 indicates that the patient smokes, whereas 0 indicates that the patient does not. Similarly, suppose the patient undergoes a medical test that has two possible outcomes. The attribute *medical\_test* is binary, where a value of 1 means the result of the test for the patient is positive, whereas 0 means the result is negative. □

A binary attribute is **symmetric** if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute *gender* having the states *male* and *female*.

A binary attribute is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a medical test for HIV. By convention, we code the most important outcome, which is usually the rarer one, by 1 (e.g., *HIV positive*) and the other by 0 (e.g., *HIV negative*).

Computing similarities between objects involving symmetric and asymmetric binary attributes will be discussed in a later section of this chapter.

### 2.1.3 Ordinal attributes

An **ordinal attribute** is an attribute with possible values that have a meaningful order or *ranking* among them, but the magnitude between successive values is not known.

**Example 2.3. Ordinal attributes.** Suppose that *drink\_size* corresponds to the size of drinks available at a fast-food restaurant. This nominal attribute has three possible values: *small*, *medium*, and *large*.



The values have a meaningful sequence (which corresponds to increasing drink size); however, we cannot tell from the values *how much* bigger, say, a large is than a medium. Other examples of ordinal attributes include *grade* (e.g., *A+*, *A*, *A-*, *B+*, and so on) and *professional\_rank*. Professional ranks can be enumerated in a sequential order: for example, *assistant*, *associate*, and *full* for professors, and *private*, *private second class*, *private first class*, *specialist*, *corporal*, *sergeant*, . . . for army ranks.

Ordinal attributes are useful for registering subjective assessments of qualities that cannot be measured objectively; thus ordinal attributes are often used in surveys for ratings. In one survey, participants were asked to rate how satisfied they were as customers. Customer satisfaction had the following ordinal categories: 1: *very dissatisfied*, 2: *dissatisfied*, 3: *neutral*, 4: *satisfied*, and 5: *very satisfied*. □

Ordinal attributes may also be obtained from the discretization of numeric quantities by splitting the value range into a finite number of ordered categories as described in a later section on data reduction.

The central tendency of an ordinal attribute can be represented by its mode and its median (the middle value in an ordered sequence), but the mean cannot be defined.

Note that nominal, binary, and ordinal attributes are *qualitative*. That is, they *describe* a feature of an object without giving an actual size or quantity. The values of such qualitative attributes are typically words representing categories. If integers are used, they represent computer codes for the categories, as opposed to measurable quantities (e.g., 0 for *small* drink size, 1 for *medium*, and 2 for *large*). In the following subsection we look at numeric attributes, which provide *quantitative* measurements of an object.

### 2.1.4 Numeric attributes

A **numeric attribute** is *quantitative*; that is, it is a measurable quantity, represented in integer or real values. Numeric attributes can be *interval-scaled* or *ratio-scaled*.

#### *Interval-scaled attributes*

**Interval-scaled attributes** are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allow us to compare and quantify the *difference* between values.

**Example 2.4. Interval-scaled attributes.** A *temperature* attribute is interval-scaled. Suppose that we have the outdoor *temperature* values for a number of different days, where each day is an object. By ordering the values, we obtain a ranking of the objects with respect to *temperature*. In addition, we can quantify the difference between values. For example, a temperature of 20°C is five degrees higher than a temperature of 15°C. Calendar dates are another example. For instance, the years 2012 and 2020 are eight years apart. □

Temperatures in Celsius and Fahrenheit do not have a true zero-point, that is, neither 0°C nor 0°F indicates “no temperature.” (On the Celsius scale, for example, the unit of measurement is 1/100 of the difference between the melting temperature and the boiling temperature of water in atmospheric pressure.) Although we can compute the *difference* between temperature values, we cannot talk of one temperature value as being a *multiple* of another. Without a true zero, we cannot say, for instance, that 10°C is twice as warm as 5°C. That is, we cannot speak of the values in terms of ratios. Similarly, there is no true zero-point for calendar dates. (The year 0 does not correspond to the beginning of time.) This brings us to ratio-scaled attributes, for which a true zero-point exists.

Because interval-scaled attributes are numeric, we can compute their mean value, in addition to the median and mode measures of central tendency.

### **Ratio-scaled attributes**

A **ratio-scaled attribute** is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value. In addition, the values are ordered, and we can also compute the difference between values, as well as the mean, median, and mode.

**Example 2.5. Ratio-scaled attributes.** Unlike temperatures in Celsius and Fahrenheit, the Kelvin (K) temperature scale has what is considered a true zero-point ( $0\text{ K} = -273.15^\circ\text{C}$ ): It is the point at which all thermal motion ceases in the classical description of thermodynamics. Other examples of ratio-scaled attributes include *count* attributes such as *years\_of\_experience* (e.g., the objects are employees) and *number\_of\_words* (e.g., the objects are documents). Additional examples include attributes to measure weight, height, and speed, and monetary quantities (e.g., you are 100 times richer with \$100 than with \$1).  $\square$

## 2.1.5 Discrete vs. continuous attributes

In our presentation, we have organized attributes into nominal, binary, ordinal, and numeric types. There are many ways to organize attribute types. The types are not mutually exclusive.

Classification algorithms developed from the field of machine learning often consider attributes as being either *discrete* or *continuous*. Each type may be processed differently. A **discrete attribute** has a finite or countably infinite set of values, which may or may not be represented as integers. The attributes *hair\_color*, *smoker*, *medical\_test*, and *drink\_size* each have a finite number of values, and so are discrete. Note that discrete attributes may have numeric values, such as 0 and 1 for binary attributes or, the values 0 to 110 for the attribute *age*. An attribute is *countably infinite* if the set of possible values is infinite but the values can be put in a one-to-one correspondence with natural numbers. For example, the attribute *customer\_ID* is countably infinite. The number of customers can grow to infinity, but in reality, the actual set of values is countable (where the values can be put in one-to-one correspondence with the set of integers). Zip codes are another example.

If an attribute is not discrete, it is **continuous**. The terms *numeric attribute* and *continuous attribute* are often used interchangeably in the literature. (This can be confusing because, in the classic sense, continuous values are real numbers, whereas numeric values can be either integers or real numbers.) In practice, real values are represented using a finite number of digits. Continuous attributes are typically represented as floating-point variables.

---

## 2.2 Statistics of data

For data preprocessing to be successful, it is essential to have an overall picture of your data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.

This section discusses three areas of basic statistical descriptions. We start with *measures of central tendency* (Section 2.2.1), which measure the location of the middle or center of a data distribution.

Intuitively speaking, given an attribute, where do most of its values fall? In particular, we discuss the mean, median, mode, and midrange.

In addition to assessing the central tendency of our data set, we also would like to have an idea of the *dispersion of the data*. That is, how are the data spread out? The most common data dispersion measures are the *range*, *quartiles* (e.g.,  $Q_1$ , which is the first quartile, i.e., the 25th percentile), and *interquartile range*; the *five-number summary* and *boxplots*; and the *variance* and *standard deviation* of the data. These measures are useful for identifying outliers and are described in Section 2.2.2.

To facilitate the description of relations among multiple variables, the concepts of *co-variance* and *correlation coefficient* for numerical data and  $\chi^2$  *correlation test* for nominal data are introduced in Section 2.2.3.

Finally, we can use many graphic displays of basic statistical descriptions to visually inspect our data (Section 2.2.4). Most statistical or graphical data presentation software packages include bar charts, pie charts, and line graphs. Other popular displays of data summaries and distributions include *quantile plots*, *quantile-quantile plots*, *histograms*, and *scatter plots*.

## 2.2.1 Measuring the central tendency

In this section, we look at various ways to measure the central tendency of data. Suppose that we have some attribute  $X$ , like *salary*, which has been recorded for a set of objects. Let  $x_1, x_2, \dots, x_N$  be the set of  $N$  observed values or *observations* for  $X$ . Here, these values may also be referred to as the data set (for  $X$ ). If we were to plot the observations for *salary*, where would most of the values fall? This gives us an idea of the central tendency of the data. Measures of central tendency include the mean, median, mode, and midrange.

The most common and effective numeric measure of the “center” of a set of data is the (*arithmetic*) *mean*. Let  $x_1, x_2, \dots, x_N$  be a set of  $N$  values or *observations*, such as for some numeric attribute  $X$ , like *salary*. The **mean** of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N} = \frac{x_1 + x_2 + \dots + x_N}{N}. \quad (2.1)$$

This corresponds to the built-in aggregate function, *average* (`avg()` in SQL), provided in relational database systems.

**Example 2.6. Mean.** Suppose we have the following values for *salary* (in thousands of dollars), shown in ascending order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110. Using Eq. (2.1), we have

$$\begin{aligned} \bar{x} &= \frac{30 + 36 + 47 + 50 + 52 + 52 + 56 + 60 + 63 + 70 + 70 + 110}{12} \\ &= \frac{696}{12} = 58. \end{aligned}$$

Thus, the mean salary is \$58,000. □

Sometimes, each value  $x_i$  in a set may be associated with a weight  $w_i$  for  $i = 1, \dots, N$ . The weights reflect the significance, importance, or occurrence frequency attached to their respective values. In this

case, we can compute

$$\bar{x} = \frac{\sum_{i=1}^N w_i x_i}{\sum_{i=1}^N w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}. \quad (2.2)$$

This is called the **weighted arithmetic mean** or the **weighted average**.

Although the mean is the single most useful quantity for describing a data set, it is not always the best way of measuring the center of the data. A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the mean score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number of extreme values, we can instead use the **trimmed mean**, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for *salary* and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends, as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the **median**, which is the middle value in a set of ordered data values. It is the value that separates the higher half of a data set from the lower half.

In probability and statistics, the median generally applies to numeric data; however, we may extend the concept to ordinal data. Suppose that a given data set of  $N$  values for an attribute  $X$  is sorted in ascending order. If  $N$  is odd, then the median is the *middle value* of the ordered set. If  $N$  is even, then the median is not unique; it is the two middlemost values and any value in between. If  $X$  is a numeric attribute in this case, by convention, the median is taken as the average of the two middlemost values.

**Example 2.7. Median.** Let's find the median of the data from Example 2.6. The data are already sorted in ascending order. There is an even number of observations (i.e., 12); therefore, the median is not unique. It can be any value within the two middlemost values of 52 and 56 (that is, within the sixth and seventh values in the list). By convention, we assign the average of the two middlemost values as the median; that is,  $\frac{52+56}{2} = \frac{108}{2} = 54$ . Thus, the median is \$54,000.

Suppose that we had only the first 11 values in the list. Given an odd number of values, the median is the middlemost value. This is the sixth value in this list, which has a value of \$52,000.  $\square$

The median is expensive to compute when we have a large number of observations. For numeric attributes, however, we can easily *approximate* the value. Assume that data are grouped in intervals according to their  $x_i$  data values and that the frequency (i.e., number of data values) of each interval is known. For example, employees may be grouped according to their annual salary in intervals such as \$10,001–20,000, \$20,001–50,000, and so on. (A similar, concrete example can be seen in the data table of Exercise 2.3.) Let the interval that contains the median frequency be the *median interval*. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the

formula

$$median \approx L_1 + \left( \frac{N/2 - (\sum freq)_l}{freq_{median}} \right) \times width, \quad (2.3)$$

where  $L_1$  is the lower boundary of the median interval,  $N$  is the number of values in the entire data set,  $(\sum freq)_l$  is the sum of the frequencies of all of the intervals that are lower than the median interval,  $freq_{median}$  is the frequency of the median interval, and  $width$  is the width of the median interval.

*Mode* is another measure of central tendency. The **mode** for a set of data is the value that occurs most frequently compared to all neighboring values in the set. Therefore, it can be determined for qualitative and quantitative attributes. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. Data sets with one, two, or three modes are respectively called **unimodal**, **bimodal**, and **trimodal**. In general, a data set with two or more modes is **multimodal**.

**Example 2.8. Mode.** The data from Example 2.6 are bimodal. The two modes are \$52,000 and \$70,000. □

For unimodal numeric data that are moderately skewed (asymmetrical), we have the following empirical relation:

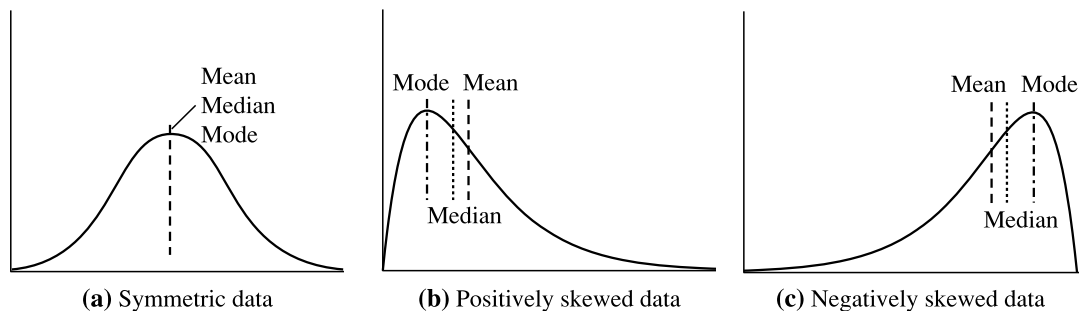
$$mean - mode \approx 3 \times (mean - median). \quad (2.4)$$

This implies that the mode for unimodal frequency curves that are moderately skewed can easily be approximated if the mean and median values are known.

The **midrange** can also be used to assess the central tendency of a numeric data set. It is the average of the largest and smallest values in the set. This measure is easy to compute using the SQL aggregate functions, `max()` and `min()`.

**Example 2.9. Midrange.** The midrange of the data of Example 2.6 is  $\frac{30,000+110,000}{2} = \$70,000$ . □

In a unimodal frequency curve with perfect **symmetric** data distribution, the mean, median, and mode are all at the same center value, as shown in Fig. 2.1a.



**FIGURE 2.1**

Mean, median, and mode of symmetric vs. positively and negatively skewed data.

Data in most real applications are not symmetric. They may instead be either **positively skewed**, where the mode occurs at a value that is smaller than the median (Fig. 2.1b), or **negatively skewed**, where the mode occurs at a value greater than the median (Fig. 2.1c).

### 2.2.2 Measuring the dispersion of data

We now look at measures to assess the dispersion or spread of numeric data. The measures include range, quantiles, quartiles, percentiles, and the interquartile range. The five-number summary, which can be displayed as a boxplot, is useful in identifying outliers. Variance and standard deviation also indicate the spread of a data distribution.

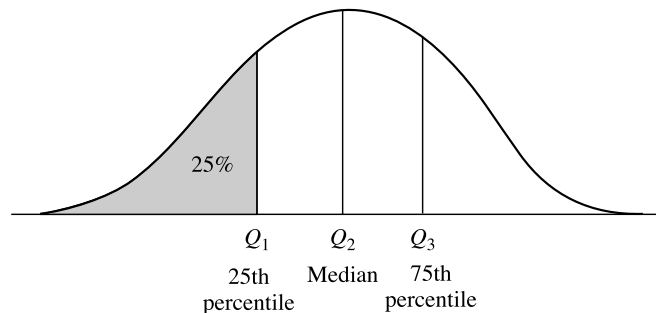
#### *Range, quartiles, and interquartile range*

To start off, let's study the *range*, *quantiles*, *quartiles*, *percentiles*, and the *interquartile range* as measures of data dispersion.

Let  $x_1, x_2, \dots, x_N$  be a set of observations for some numeric attribute,  $X$ . The **range** of the set is the difference between the largest ( $\max()$ ) and smallest ( $\min()$ ) values.

Suppose that the data for attribute  $X$  are sorted in ascending numeric order. Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets, as in Fig. 2.2. These data points are called *quantiles*. **Quantiles** are points taken at regular intervals of a data distribution, dividing it into essentially equal-size consecutive sets. (We say “essentially” because there may not be data values of  $X$  that divide the data into exactly equal-size subsets. For readability, we will refer to them as equal.) The  $k$ th  $q$ -quantile for a given data distribution is the value  $x$  such that at most  $k/q$  of the data values are less than  $x$  and at most  $(q - k)/q$  of the data values are more than  $x$ , where  $k$  is an integer such that  $0 < k < q$ . There are  $q - 1$   $q$ -quantiles.

The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as **quartiles**. The 100-quantiles are more commonly referred to as **percentiles**; they divide



**FIGURE 2.2**

A plot of the data distribution for some attribute  $X$ . The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

the data distribution into 100 equal-size consecutive sets. The median, quartiles, and percentiles are the most widely used forms of quantiles.

The quartiles give an indication of a distribution's center, spread, and shape. The **first quartile**, denoted by  $Q_1$ , is the 25th percentile. It cuts off the lowest 25% of the data. The **third quartile**, denoted by  $Q_3$ , is the 75th percentile—it cuts off the lowest 75% (or highest 25%) of the data. The second quartile is the 50th percentile. As the median, it gives the center of the data distribution.

The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range (IQR)** and is defined as

$$IQR = Q_3 - Q_1. \quad (2.5)$$

**Example 2.10. Interquartile range.** The quartiles are the three values that split the sorted data set into four equal parts. The data of Example 2.6 contain 12 observations, already sorted in ascending order. Since there are even number of elements on this list, the median of the list should be the mean of the center two elements, that is  $(\$52,000 + \$56,000)/2 = \$54,000$ . Then the first quartile should be the mean of the 3rd and 4th elements, that is,  $(\$47,000 + \$50,000)/2 = \$48,500$ , whereas the 3rd quartile should be the mean of the 9th and 10th elements, that is,  $(\$63,000 + \$70,000)/2 = \$66,500$ . Thus the interquartile range is  $IQR = \$66,500 - \$48,500 = \$18,000$ .  $\square$

### ***Five-number summary, boxplots, and outliers***

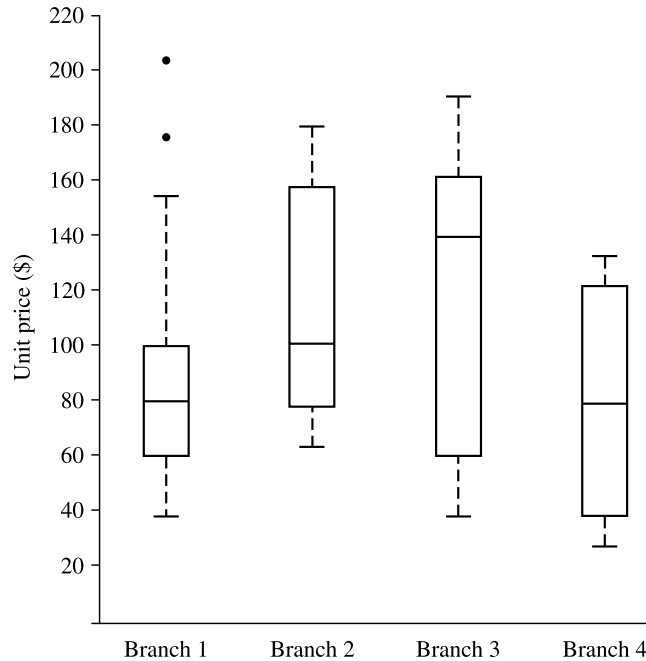
No single numeric measure of spread (e.g.,  $IQR$ ) is very useful for describing skewed distributions. Have a look at the symmetric and skewed data distributions of Fig. 2.1. In the symmetric distribution, the median (and other measures of central tendency) splits the data into equal-size halves. This does not occur for skewed distributions. Therefore it is more informative to also provide the two quartiles  $Q_1$  and  $Q_3$ , along with the median. A common rule of thumb for identifying suspected **outliers** is to single out values falling at least  $1.5 \times IQR$  above the third quartile or below the first quartile.

Because  $Q_1$ , the median, and  $Q_3$  together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The **five-number summary** of a distribution consists of the median ( $Q_2$ ), the quartiles  $Q_1$  and  $Q_3$ , and the smallest and largest individual observations, written in the order of *Minimum*,  $Q_1$ , *Median*,  $Q_3$ , *Maximum*.

**Boxplots** are a popular way of visualizing a distribution. A boxplot incorporates the five-number summary as follows:

- Typically, the ends of the box are at the quartiles so that the box length is the interquartile range.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

When dealing with a moderate number of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme low and high observations *only if* these values are less than  $1.5 \times IQR$  beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within  $1.5 \times IQR$  of the quartiles. The remaining cases are plotted individually. Boxplots can be used in the comparisons of several sets of compatible data.



**FIGURE 2.3**

Boxplot for the unit price data for items sold at four branches of an online store during a given time period.

**Example 2.11. Boxplot.** Fig. 2.3 shows boxplots for unit price data for items sold at four branches of an online store during a given time period. For branch 1, we see that the median price of items sold is \$80,  $Q_1$  is \$60, and  $Q_3$  is \$100. Notice that two outlying observations for this branch were plotted individually, as their values of 175 and 202 are more than 1.5 times the  $IQR$  here of 40.  $\square$

### Variance and standard deviation

Variance and standard deviation are measures of data dispersion. They indicate how spread out a data distribution is. A low standard deviation means that the data observations tend to be very close to the mean, whereas a high standard deviation indicates that the data are spread out over a large range of values.

The **variance** of  $N$  observations,  $x_1, x_2, \dots, x_N$  (when  $N$  is large), for a numeric attribute  $X$  is

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \left( \frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2, \quad (2.6)$$

where  $\bar{x}$  is the mean value of the observations, as defined in Eq. (2.1). The **standard deviation**,  $\sigma$ , of the observations is the square root of the variance,  $\sigma^2$ .



**Example 2.12. Variance and standard deviation.** In Example 2.6, we found  $\bar{x} = \$58,000$  using Eq. (2.1) for the mean. To determine the variance and standard deviation of the data from that example, we set  $N = 12$  and use Eq. (2.6) to obtain

$$\begin{aligned}\sigma^2 &= \frac{1}{12}(30^2 + 36^2 + 47^2 \dots + 110^2) - 58^2 \\ &\approx 379.17 \\ \sigma &\approx \sqrt{379.17} \approx 19.47.\end{aligned}$$

□

The basic properties of the standard deviation,  $\sigma$ , as a measure of spread are as follows:

- $\sigma$  measures spread about the mean and should be considered only when the mean is chosen as the measure of center.
- $\sigma = 0$  only when there is no spread, that is, when all observations have the same value. Otherwise,  $\sigma > 0$ .

Importantly, an observation is unlikely to be more than several standard deviations away from the mean. Mathematically, using Chebyshev's inequality, it can be shown that at least  $\left(1 - \frac{1}{k^2}\right) \times 100\%$  of the observations are no more than  $k$  standard deviations from the mean. Therefore, the standard deviation is a good indicator of the spread of a data set.

The computation of the variance and standard deviation is scalable in large data sets.

## 2.2.3 Covariance and correlation analysis

### *Covariance of numeric data*

In probability theory and statistics, correlation and covariance are two similar measures for assessing how much two attributes change together. Consider two numeric attributes  $A$  and  $B$  and a set of  $n$  real-valued observations  $\{(a_1, b_1), \dots, (a_n, b_n)\}$ . The mean values of  $A$  and  $B$ , respectively, are also known as the **expected values** on  $A$  and  $B$ , that is,

$$E(A) = \bar{A} = \frac{\sum_{i=1}^n a_i}{n}$$

and

$$E(B) = \bar{B} = \frac{\sum_{i=1}^n b_i}{n}.$$

The **covariance** between  $A$  and  $B$  is defined as

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^n (a_i - \bar{A})(b_i - \bar{B})}{n}. \quad (2.7)$$

Mathematically, it can also be shown that

$$Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B}. \quad (2.8)$$

This equation may simplify calculations.

For two attributes  $A$  and  $B$  that tend to change together, if a value  $a_i$  of  $A$  is larger than  $\bar{A}$  (the expected value of  $A$ ), then the corresponding value of  $b_i$  of attribute  $B$  is likely to be larger than  $\bar{B}$  (the expected value of  $B$ ). Therefore the covariance between  $A$  and  $B$  is *positive*. On the other hand, if one of the attributes tends to be above its expected value when the other attribute is below its expected value, then the covariance of  $A$  and  $B$  is *negative*.

If  $A$  and  $B$  are *independent* (i.e., they do not have correlation), then  $E(A \cdot B) = E(A) \cdot E(B)$ . Therefore the covariance is  $Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B} = E(A) \cdot E(B) - \bar{A}\bar{B} = 0$ . However, the converse is not true. Some pairs of random variables (attributes) may have a covariance of 0 but are not independent. Only under some additional assumptions (e.g., the data follow multivariate normal distributions) does a covariance of 0 imply independence.

**Example 2.13. Covariance analysis of numeric attributes.** Consider Table 2.1, which presents a simplified example of stock prices observed at five time points for *AllElectronics* and *HighTech*, a high-tech company. If the stocks are affected by the same industry trends, will their prices rise or fall together?

$$E(\text{AllElectronics}) = \frac{6 + 5 + 4 + 3 + 2}{5} = \frac{20}{5} = \$4$$

and

$$E(\text{HighTech}) = \frac{20 + 10 + 14 + 5 + 5}{5} = \frac{54}{5} = \$10.80.$$

Thus, using Eq. (2.7), we compute

$$\begin{aligned} Cov(\text{AllElectronics}, \text{HighTech}) &= \frac{6 \times 20 + 5 \times 10 + 4 \times 14 + 3 \times 5 + 2 \times 5}{5} - 4 \times 10.80 \\ &= 50.2 - 43.2 = 7. \end{aligned}$$

Therefore, given the positive covariance we can say that stock prices for both companies rise together.  $\square$

*Variance* is a special case of covariance, where the two attributes are identical (i.e., the covariance of an attribute with itself).

Time point	AllElectronics	HighTech
t1	6	20
t2	5	10
t3	4	14
t4	3	5
t5	2	5

### Correlation coefficient for numeric data

For numeric attributes, we can evaluate the correlation between two attributes,  $A$  and  $B$ , by computing the **correlation coefficient** (also known as **Pearson's product moment coefficient**, named after its inventor, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^n (a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^n (a_i b_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}, \quad (2.9)$$

where  $n$  is the number of tuples,  $a_i$  and  $b_i$  are the respective values of  $A$  and  $B$  in tuple  $i$ ,  $\bar{A}$  and  $\bar{B}$  are the respective mean values of  $A$  and  $B$ ,  $\sigma_A$  and  $\sigma_B$  are the respective standard deviations of  $A$  and  $B$  (as defined in Section 2.2.2), and  $\sum(a_i b_i)$  is the sum of the  $AB$  cross-product (i.e., for each tuple, the value for  $A$  is multiplied by the value for  $B$  in that tuple). Note that  $-1 \leq r_{A,B} \leq +1$ . If  $r_{A,B}$  is greater than 0, then  $A$  and  $B$  are *positively correlated*, meaning that the values of  $A$  increase as the values of  $B$  increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that  $A$  (or  $B$ ) may be removed as a redundancy.

If the resulting value is equal to 0, then  $A$  and  $B$  are *independent*, and there is no correlation between them. If the resulting value is less than 0, then  $A$  and  $B$  are *negatively correlated*, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes (Section 2.2.3). For example, Fig. 2.8's scatter plots, respectively, show positively correlated data and negatively correlated data, whereas Fig. 2.9 displays uncorrelated data.

Note that correlation does not imply causality. That is, if  $A$  and  $B$  are correlated, this does not necessarily imply that  $A$  causes  $B$  or that  $B$  causes  $A$ . For example, in analyzing a demographic database, we may find that attributes representing the number of hospitals and the number of car thefts in a region are correlated. This does not mean that one causes the other. Both are actually causally linked to a third attribute, namely, *population*.

### $\chi^2$ correlation test for nominal data

For nominal data, a correlation relationship between two attributes,  $A$  and  $B$ , can be discovered by a  $\chi^2$  (**chi-square**) test. Suppose  $A$  has  $c$  distinct values, namely,  $a_1, a_2, \dots, a_c$ , and  $B$  has  $r$  distinct values, namely,  $b_1, b_2, \dots, b_r$ . The data tuples described by  $A$  and  $B$  can be shown as a **contingency table**, with the  $c$  values of  $A$  making up the columns and the  $r$  values of  $B$  making up the rows. Let  $(A_i, B_j)$  denote the joint event that attribute  $A$  takes on value  $a_i$  and attribute  $B$  takes on value  $b_j$ , that is, where  $(A = a_i, B = b_j)$ . Each and every possible  $(A_i, B_j)$  joint event has its own cell (or slot) in the table. The  $\chi^2$  value (also known as the *Pearson  $\chi^2$  statistic*) is computed as

$$\chi^2 = \sum_{i=1}^c \sum_{j=1}^r \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \quad (2.10)$$

where  $o_{ij}$  is the *observed frequency* (i.e., actual count) of the joint event  $(A_i, B_j)$  and  $e_{ij}$  is the *expected frequency* of  $(A_i, B_j)$ , which can be computed as

$$e_{ij} = \frac{\text{count}(A = a_i) \times \text{count}(B = b_j)}{n}, \quad (2.11)$$

where  $n$  is the number of data tuples,  $count(A = a_i)$  is the number of tuples having value  $a_i$  for  $A$ , and  $count(B = b_j)$  is the number of tuples having value  $b_j$  for  $B$ . The sum in Eq. (2.10) is computed over all of the  $r \times c$  cells. Note that the cells that contribute the most to the  $\chi^2$  value are those for which the actual count is very different from that expected.

The  $\chi^2$  statistic tests the hypothesis that  $A$  and  $B$  are *independent*, that is, there is no correlation between them. The test is based on a significance level, with  $(r - 1) \times (c - 1)$  degrees of freedom. We illustrate the use of this statistic in Example 2.14. If the hypothesis can be rejected, then we say that  $A$  and  $B$  are statistically correlated.

**Example 2.14. Correlation analysis of nominal attributes using  $\chi^2$ .** Suppose that a group of 1500 people was surveyed. The gender of each person was noted. Each person was polled as to whether his or her preferred type of reading material was fiction or nonfiction. Thus, we have two attributes, *gender* and *preferred\_reading*. The observed frequency (or count) of each possible joint event is summarized in the contingency table shown in Table 2.2, where the numbers in parentheses are the expected frequencies. The expected frequencies are calculated based on the data distribution for both attributes using Eq. (2.11).

Using Eq. (2.11), we can verify the expected frequencies for each cell. For example, the expected frequency for the cell (*male, fiction*) is

$$e_{11} = \frac{count(male) \times count(fiction)}{n} = \frac{300 \times 450}{1500} = 90,$$

and so on. Notice that in any row, the sum of the expected frequencies must equal the total observed frequency for that row, and the sum of the expected frequencies in any column must also equal the total observed frequency for that column.

Using Eq. (2.10) for  $\chi^2$  computation, we get

$$\begin{aligned} \chi^2 &= \frac{(250 - 90)^2}{90} + \frac{(50 - 210)^2}{210} + \frac{(200 - 360)^2}{360} + \frac{(1000 - 840)^2}{840} \\ &= 284.44 + 121.90 + 71.11 + 30.48 = 507.93. \end{aligned}$$

For this  $2 \times 2$  table, the degrees of freedom are  $(2 - 1) \times (2 - 1) = 1$ . For 1 degree of freedom, the  $\chi^2$  value needed to reject the hypothesis at the 0.001 significance level is 10.828 (taken from the table of upper percentage points of the  $\chi^2$  distribution, typically available from any textbook on statistics). Since our computed value is above this, we can reject the hypothesis that *gender* and *preferred\_reading*

	Male	Female	Total
<i>fiction</i>	250 (90)	200 (360)	450
<i>non_fiction</i>	50 (210)	1000 (840)	1050
Total	300	1200	1500

*Note: Are gender and preferred\_reading correlated?*

are independent and conclude that the two attributes are (strongly) correlated for the given group of people.  $\square$

### 2.2.4 Graphic displays of basic statistics of data

In this section, we study graphic displays of basic statistical descriptions. These include *quantile plots*, *quantile-quantile plots*, *histograms*, and *scatter plots*. Such graphs are helpful for the visual inspection of data, which is useful for data preprocessing. The first three of these show univariate distributions (i.e., data for one attribute), whereas scatter plots show bivariate distributions (i.e., involving two attributes).

#### Quantile plot

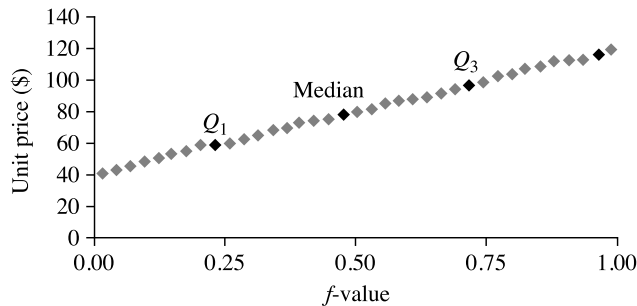
A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing a user to assess both the overall behavior and unusual occurrences). Second, it plots quantile information (see Section 2.2.2). Let  $x_i$ , for  $i = 1$  to  $N$ , be the data sorted in ascending order so that  $x_1$  is the smallest observation and  $x_N$  is the largest for some ordinal or numeric attribute  $X$ . Each observation,  $x_i$ , is paired with a percentage,  $f_i$ , which indicates that approximately  $f_i \times 100\%$  of the data are below the value,  $x_i$ . We say “approximately” because there may not be a value with exactly a fraction,  $f_i$ , of the data below  $x_i$ . Note that the 0.25 quantile corresponds to quartile  $Q_1$ , the 0.50 quantile is the median, and the 0.75 quantile is  $Q_3$ .

Let

$$f_i = \frac{i - 0.5}{N}. \quad (2.12)$$

These numbers increase in equal steps of  $1/N$ , ranging from  $\frac{1}{2N}$  (which is slightly above 0) to  $1 - \frac{1}{2N}$  (which is slightly below 1). On a quantile plot,  $x_i$  is graphed against  $f_i$ . This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their  $Q_1$ , median,  $Q_3$ , and other  $f_i$  values at a glance.

**Example 2.15. Quantile plot.** Fig. 2.4 shows a quantile plot for the *unit price* data of Table 2.3.  $\square$



**FIGURE 2.4**

A quantile plot for the unit price data of Table 2.3.

**Table 2.3 A set of unit price data for items sold at a branch of the online store.**

Unit price (\$)	Count of items sold
40	275
43	300
47	250
⋮	⋮
74	360
75	515
78	540
⋮	⋮
115	320
117	270
120	350

### Quantile-quantile plot

A **quantile-quantile plot**, or **q-q plot**, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Suppose that we have two sets of observations for the attribute or variable *unit price*, taken from two different branch locations. Let  $x_1, \dots, x_N$  be the data from the first branch, and  $y_1, \dots, y_M$  be the data from the second, where each data set is sorted in ascending order. If  $M = N$  (i.e., the number of points in each set is the same), then we simply plot  $y_i$  against  $x_i$ , where  $y_i$  and  $x_i$  are both  $(i - 0.5)/N$  quantiles of their respective data sets. If  $M < N$  (i.e., the second branch has fewer observations than the first), there can be only  $M$  points on the q-q plot. Here,  $y_i$  is the  $(i - 0.5)/M$  quantile of the  $y$  data, which is plotted against the  $(i - 0.5)/M$  quantile of the  $x$  data. This computation typically involves interpolation.

**Example 2.16. Quantile-quantile plot.** Fig. 2.5 shows a quantile-quantile plot for *unit price* data of items sold at two branches of the online store during a given time period. Each point corresponds to the same quantile for each data set and shows the unit price of items sold at branch 1 vs. branch 2 for that quantile. (To aid comparison, the straight line represents the case where, for each given quantile, the unit price at each branch is the same. The darker points correspond to the data for  $Q_1$ , the median, and  $Q_3$ , respectively.)

We see, for example, that at  $Q_1$ , the unit price of items sold at branch 1 was slightly less than that at branch 2. In other words, 25% of items sold at branch 1 were less than or equal to \$60, whereas 25% of items sold at branch 2 were less than or equal to \$64. At the 50th percentile (marked by the median, which is also  $Q_2$ ), we see that 50% of items sold at branch 1 were less than \$78, whereas 50% of items at branch 2 were less than \$85. In general, we note that there is a shift in the distribution of branch 1 with respect to branch 2 in that the unit prices of items sold at branch 1 tend to be lower than those at branch 2.  $\square$

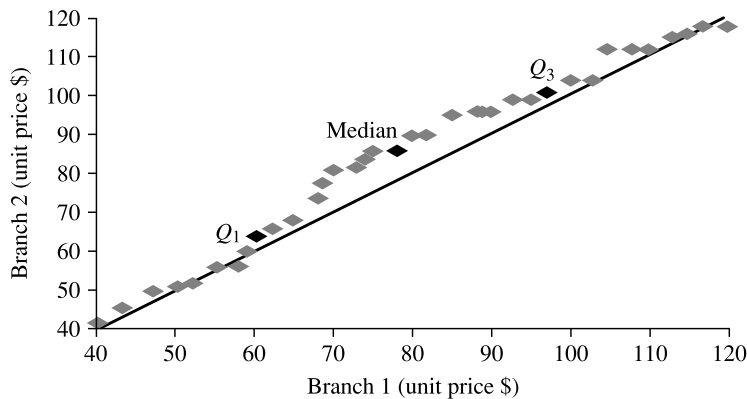


FIGURE 2.5

A q-q plot for unit price data from two branches of the online store.

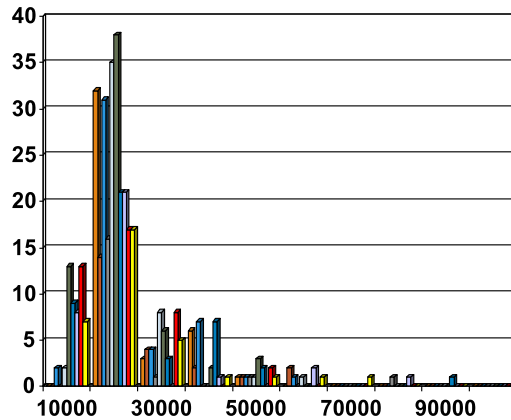
### Histograms

**Histograms** (or **frequency histograms**) are at least a century old and are widely used. “Histos” means pole or mast, and “gram” means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute,  $X$ . According to the number of poles desired in the chart, the range of values for  $X$  is partitioned into a set of disjoint consecutive subranges. The subranges, referred to as *buckets* or *bins*, are disjoint subsets of the data distribution for  $X$ . The range of a bucket is known as the **width**. Typically, the buckets are of equal width. For example, a *price* attribute with a value range of \$1–\$200 (rounded up to the nearest dollar) can be partitioned into subranges 1–20, 21–40, 41–60, and so on. For each subrange, a bar is drawn with a height that represents the total count of items observed within the subrange.

Please note that histogram is different from another popularly used graph representation called **bar chart**. Bar chart uses a set of bars (often separated with space) with  $X$  representing a set of categorical data, such as *automobile\_model* or *item\_type*, and the height of the bar (column) indicates the size of the group defined by the categories. On the other hand, histogram plots quantitative data with a range of  $X$  values grouped into bins or intervals. Histograms are used to show distributions (along  $X$  axis) while bar charts are used to compare categories. It is always appropriate to talk about the skewness of a histogram; that is, the tendency of the observations to fall more on the low end or the high end of the  $X$  axis. However, bar chart’s  $X$  axis does not have a low end or a high end; because the labels on the  $X$  axis are categorical—not quantitative. Thus, bars can be reordered in bar charts but not in histograms.

**Example 2.17. Histogram.** Fig. 2.6 shows a histogram for a data set on research award distribution for a region, where buckets (or bins) are defined by equal-width ranges representing \$1000 increments, and the frequency is the number of research awards in the corresponding buckets. □

Although histograms are widely used, they may not be as effective as the quantile plot, q-q plot, and boxplot methods in comparing groups of univariate observations.



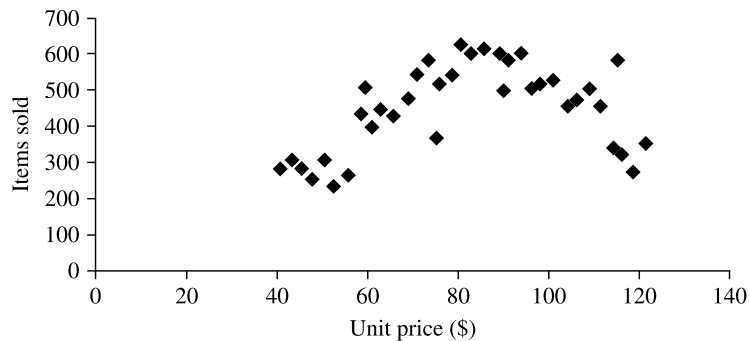
**FIGURE 2.6**

A histogram on research award distribution for a region.

### *Scatter plots and data correlation*

A **scatter plot** is one of the most effective graphical methods for determining whether there appears to be a relationship, pattern, or trend between two numeric attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. Fig. 2.7 shows a scatter plot for the set of data in Table 2.3.

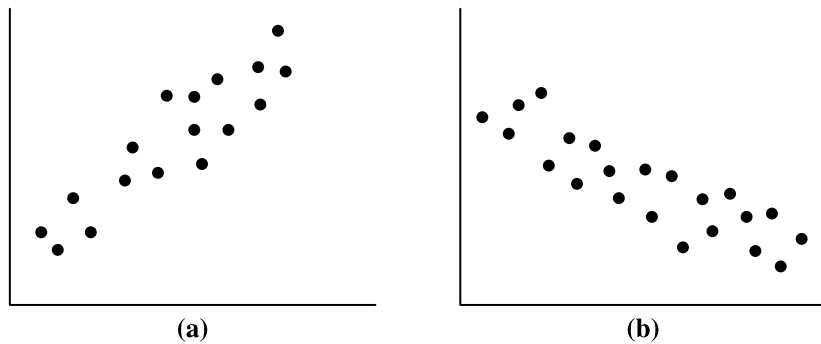
The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships. Two attributes,  $X$  and  $Y$ , are **correlated** if the knowledge of one attribute enables to predict the other with some accuracy. Correlations can be positive, negative, or null (uncorrelated). Fig. 2.8 shows examples of positive and negative correlations between two attributes.



**FIGURE 2.7**

A scatter plot for Table 2.3 data set.

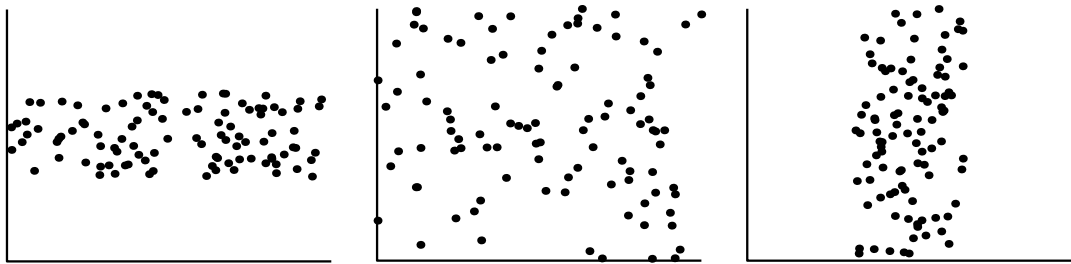





---

**FIGURE 2.8**

Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.




---

**FIGURE 2.9**

Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

If the plotted points pattern slopes from lower left to upper right, this means that the values of  $X$  increase as the values of  $Y$  increase, suggesting a *positive correlation* (Fig. 2.8a). If the pattern of plotted points slopes from upper left to lower right, the values of  $X$  increase as the values of  $Y$  decrease, suggesting a *negative correlation* (Fig. 2.8b). A line of best fit can be drawn to study the correlation between the variables. Statistical tests for correlation are introduced in Appendix A.

Fig. 2.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets. Scatter plots can also be extended to  $n$  attributes, resulting in a *scatter-plot matrix*.

In summary, basic data descriptions (e.g., measures of central tendency and measures of dispersion) and graphic statistical displays (e.g., quantile plots, histograms, and scatter plots) provide valuable insight into the overall behavior of your data. By helping to identify noise and outliers, they are especially useful for data cleaning.

---

## 2.3 Similarity and distance measures

In data mining applications, such as clustering, outlier analysis, and nearest-neighbor classification, we need ways to assess how alike or unlike objects are in comparison to one another. For example, a store may want to search for clusters of *customer* objects, resulting in groups of customers with similar characteristics (e.g., similar income, area of residence, and age). Such information can then be used for marketing. A **cluster** is a collection of data objects such that the objects within a cluster are *similar* to one another and *dissimilar* to the objects in other clusters. Outlier analysis also employs clustering-based techniques to identify potential outliers as objects that are highly dissimilar to others. Knowledge of object similarities can also be used in nearest-neighbor classification schemes where a given object (e.g., a *patient*) is assigned a class label (relating to, say, a *diagnosis*) based on its similarity toward other objects in the model.

This section presents similarity and dissimilarity measures, which are referred to as measures of *proximity*. Similarity and dissimilarity are related. A similarity measure for two objects,  $i$  and  $j$ , will typically return value 0 if the objects are completely unlike. The higher the similarity value, the greater the similarity between objects. (Typically, a value of 1 indicates complete similarity, that is, the objects are identical.) A dissimilarity measure works the opposite way. It returns a value of 0 if the objects are the same (and therefore, far from being dissimilar). The higher the dissimilarity value, the more dissimilar the two objects are.

In Section 2.3.1 we present two data structures that are commonly used in the above types of applications: the *data matrix* (used to store the data objects) and the *dissimilarity matrix* (used to store dissimilarity values for pairs of objects). We also switch to a different notation for data objects than previously used in this chapter since now we are dealing with objects described by more than one attribute. We then discuss how object dissimilarity can be computed for objects described by *nominal* attributes (Section 2.3.2), by *binary* attributes (Section 2.3.3), by *numeric* attributes (Section 2.3.4), by *ordinal* attributes (Section 2.3.5), or by combinations of these attribute types (Section 2.3.6). Section 2.3.7 provides similarity measures for very long and sparse data vectors, such as term-frequency vectors representing documents in information retrieval. Finally, Section 2.3.8 discusses how to measure the difference between two probability distributions over the same variable  $x$ , and introduces a measure, called the *Kullback-Leibler divergence*, or simply, the *KL divergence*, which has been popularly used in the data mining literature.

Knowing how to compute dissimilarity is useful in studying attributes and will also be referenced in later topics on clustering (Chapters 8 and 9), outlier analysis (Chapter 11), and nearest-neighbor classification (Chapter 6).

### 2.3.1 Data matrix vs. dissimilarity matrix

In Section 2.2, we looked at ways of studying the central tendency, dispersion, and spread of observed values for some attribute  $X$ . Our objects there were one-dimensional, that is, described by a single attribute. In this section, we talk about objects described by *multiple* attributes. Therefore we need a change in notation. Suppose that we have  $n$  objects (e.g., persons, items, or courses) described by  $p$  attributes (also called *measurements* or *features*, such as age, height, weight, or gender). The objects are  $x_1 = (x_{11}, x_{12}, \dots, x_{1p})$ ,  $x_2 = (x_{21}, x_{22}, \dots, x_{2p})$ , and so on, where  $x_{ij}$  is the value for object  $x_i$  of the  $j$ th attribute. For brevity, we hereafter refer to object  $x_i$  as object  $i$ . The objects may be tuples in a relational database and are also referred to as *data samples* or *feature vectors*.

Main memory-based clustering and nearest-neighbor algorithms typically operate on either of the following two data structures:

- **Data matrix** (or *object-by-attribute structure*): This structure stores the  $n$  data objects in the form of a relational table or an  $n$ -by- $p$  matrix ( $n$  objects  $\times$   $p$  attributes):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}. \quad (2.13)$$

Each row corresponds to an object. As part of our notation, we may use  $f$  to index through the  $p$  attributes.

- **Dissimilarity matrix** (or *object-by-object structure*): This structure stores a collection of proximities that are available for all pairs of  $n$  objects. It is often represented by an  $n$ -by- $n$  table:

$$\begin{bmatrix} 0 & & & & & \\ d(2, 1) & 0 & & & & \\ d(3, 1) & d(3, 2) & 0 & & & \\ \vdots & \vdots & \vdots & & & \\ d(n, 1) & d(n, 2) & \cdots & \cdots & 0 & \end{bmatrix}, \quad (2.14)$$

where  $d(i, j)$  is the measured **dissimilarity** or “difference” between objects  $i$  and  $j$ . In general,  $d(i, j)$  is a nonnegative number that is close to 0 when objects  $i$  and  $j$  are highly similar or “near” each other, and becomes larger the more they differ. Note that  $d(i, i) = 0$ ; that is, the difference between an object and itself is 0. Furthermore,  $d(i, j) = d(j, i)$ . (For readability, we do not show the  $d(j, i)$  entries since the matrix is symmetric.) Measures of dissimilarity are discussed throughout the remainder of this chapter.

Measures of similarity can often be expressed as a function of measures of dissimilarity. For example, for nominal data,

$$\text{sim}(i, j) = 1 - d(i, j), \quad (2.15)$$

where  $\text{sim}(i, j)$  is the similarity between objects  $i$  and  $j$ . Throughout the rest of this chapter, we will also comment on measures of similarity.

A data matrix is made up of two entities or “things,” namely rows (for objects) and columns (for attributes). Therefore, the data matrix is often called a **two-mode** matrix. The dissimilarity matrix contains one kind of entity (dissimilarities) and so is called a **one-mode** matrix. Many clustering and nearest-neighbor algorithms operate on a dissimilarity matrix. Data in the form of a data matrix can be transformed into a dissimilarity matrix before applying such algorithms.

### 2.3.2 Proximity measures for nominal attributes

A nominal attribute can take on two or more states (Section 2.1.1). For example, *map\_color* is a nominal attribute that may have, say, five states: *red*, *yellow*, *green*, *pink*, and *blue*.

Let the number of states of a nominal attribute be  $M$ . The states can be denoted by letters, symbols, or a set of integers, such as  $1, 2, \dots, M$ . Notice that such integers are used just for data handling and do not represent any specific ordering.

“How is dissimilarity computed between objects described by nominal attributes?” The dissimilarity between two objects  $i$  and  $j$  can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p}, \tag{2.16}$$

where  $m$  is the number of *matches* (i.e., the number of attributes for which  $i$  and  $j$  are in the same state), and  $p$  is the total number of attributes describing the objects. Weights can be assigned to increase the effect of  $m$  or to assign greater weight to the matches in attributes having a larger number of states.

**Example 2.18. Dissimilarity between nominal attributes.** Suppose that we have the sample data of Table 2.4, except that only the *object-identifier* and the attribute *test-1* are available, where *test-1* is nominal. (We will use *test-2* and *test-3* in later examples.) Let’s compute the dissimilarity matrix Eq. (2.14), that is,

$$\begin{bmatrix} 0 & & & & \\ d(2, 1) & 0 & & & \\ d(3, 1) & d(3, 2) & 0 & & \\ d(4, 1) & d(4, 2) & d(4, 3) & 0 & \end{bmatrix}.$$

Since here we have one nominal attribute, *test-1*, we set  $p = 1$  in Eq. (2.16) so that  $d(i, j)$  evaluates to 0 if objects  $i$  and  $j$  match, and 1 if the objects differ. Thus, we get

$$\begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 1 & 0 & & \\ 0 & 1 & 1 & 0 & \end{bmatrix}.$$

From this, we see that all objects are dissimilar except objects 1 and 4 (i.e.,  $d(4, 1) = 0$ ). □

Alternatively, similarity can be computed as

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}. \tag{2.17}$$

Object Identifier	Test-1 (nominal)	Test-2 (ordinal)	Test-3 (numeric)
1	code A	excellent	45
2	code B	fair	22
3	code C	good	64
4	code A	excellent	28

Proximity between objects described by nominal attributes can be computed using an alternative encoding scheme. Nominal attributes can be encoded using asymmetric binary attributes by creating a new binary attribute for each of the  $M$  states. For an object with a given state value, the binary attribute representing that state is set to 1, whereas the remaining binary attributes are set to 0. For example, to encode the nominal attribute *map\_color*, a binary attribute can be created for each of the five colors previously listed. For an object having the color *yellow*, the *yellow* attribute is set to 1, whereas the remaining four attributes are set to 0. Proximity measures for this form of encoding can be calculated using the methods discussed in the next subsection.

### 2.3.3 Proximity measures for binary attributes

Let's look at dissimilarity and similarity measures for objects described by either *symmetric* or *asymmetric binary attributes*.

Recall that a binary attribute has only one of two states, 0 and 1, where 0 means that the attribute is absent, and 1 means that it is present (Section 2.1.2). Given the attribute *smoker* describing a patient, for instance, 1 indicates that the patient smokes, whereas 0 indicates that the patient does not. Treating binary attributes as if they are other numeric attributes can be misleading. Therefore methods specific to binary data are necessary for computing dissimilarity.

“So, how can we compute the dissimilarity between two binary attributes?” One approach involves computing a dissimilarity matrix from the given binary data. If all binary attributes are thought of as having the same weight, we have the  $2 \times 2$  contingency table of Table 2.5, where  $q$  is the number of attributes that equal 1 for both objects  $i$  and  $j$ ,  $r$  is the number of attributes that equal 1 for object  $i$  but equal 0 for object  $j$ ,  $s$  is the number of attributes that equal 0 for object  $i$  but equal 1 for object  $j$ , and  $t$  is the number of attributes that equal 0 for both objects  $i$  and  $j$ . The total number of attributes is  $p$ , where  $p = q + r + s + t$ .

Recall that for symmetric binary attributes, each state is equally valuable. Dissimilarity that is based on symmetric binary attributes is called **symmetric binary dissimilarity**. If objects  $i$  and  $j$  are described by symmetric binary attributes, then the dissimilarity between  $i$  and  $j$  is

$$d(i, j) = \frac{r + s}{q + r + s + t}. \quad (2.18)$$

For asymmetric binary attributes, the two states are not equally important, such as the *positive* (1) and *negative* (0) outcomes of a disease test. Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore such binary attributes are often considered “monary” (having one state). The dissimilarity

**Table 2.5** Contingency table for binary attributes.

		Object $j$		sum
		1	0	
Object $i$	1	$q$	$r$	$q + r$
	0	$s$	$t$	$s + t$
sum		$q + s$	$r + t$	$p$



### 2.3.4 Dissimilarity of numeric data: Minkowski distance

In this section, we describe distance measures that are commonly used for computing the dissimilarity of objects described by numeric attributes. These measures include the *Euclidean*, *Manhattan*, and *Minkowski distances*.

In some cases, the data are normalized before applying distance calculations. This involves transforming the data to fall within a smaller or common range, such as  $[-1.0, 1.0]$  or  $[0.0, 1.0]$ . Consider a *height* attribute, for example, which could be measured in either meters or inches. In general, expressing an attribute in smaller units will lead to a larger range for that attribute and thus tend to give such attributes greater effect or “weight.” Normalizing the data attempts to give all attributes an equal weight. It may or may not be useful in a particular application. Methods for normalizing data are discussed in detail in Section 2.5 on data transformation.

The most popular distance measure is **Euclidean distance** (i.e., straight line or “as the crow flies”). Let  $i = (x_{i1}, x_{i2}, \dots, x_{ip})$  and  $j = (x_{j1}, x_{j2}, \dots, x_{jp})$  be two objects described by  $p$  numeric attributes. The Euclidean distance between objects  $i$  and  $j$  is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ip} - x_{jp})^2}. \quad (2.21)$$

Another well-known measure is the **Manhattan (or city block) distance**, named so because it is the distance in blocks between any two points in a city (such as 2 blocks down and 3 blocks over for a total of 5 blocks). It is defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|. \quad (2.22)$$

Both the Euclidean and the Manhattan distance satisfy the following mathematical properties:

**Nonnegativity:**  $d(i, j) \geq 0$ : Distance is a nonnegative number.

**Identity of indiscernibles:**  $d(i, i) = 0$ : The distance of an object to itself is 0.

**Symmetry:**  $d(i, j) = d(j, i)$ : Distance is a symmetric function.

**Triangle inequality:**  $d(i, j) \leq d(i, k) + d(k, j)$ : Going directly from object  $i$  to object  $j$  in space is no more than making a detour over any other object  $k$ .

A measure that satisfies these conditions is known as **metric**. Please note that the nonnegativity property is implied by the other three properties.

**Example 2.20. Euclidean distance and Manhattan distance.** Let  $x_1 = (1, 2)$  and  $x_2 = (3, 5)$  represent two objects as shown in Fig. 2.10. The Euclidean distance between the two is  $\sqrt{2^2 + 3^2} = 3.61$ . The Manhattan distance between the two is  $2 + 3 = 5$ .  $\square$

**Minkowski distance** is a generalization of the Euclidean and Manhattan distances. It is defined as

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \dots + |x_{ip} - x_{jp}|^h}, \quad (2.23)$$

where  $h$  is a real number such that  $h \geq 1$ . (Such a distance is also called  $L_p$  **norm** in some literature, where the symbol  $p$  refers to our notation of  $h$ . We have kept  $p$  as the number of attributes to be consistent with the rest of this chapter.) It represents the Manhattan distance when  $h = 1$  (i.e.,  $L_1$  norm) and Euclidean distance when  $h = 2$  (i.e.,  $L_2$  norm).

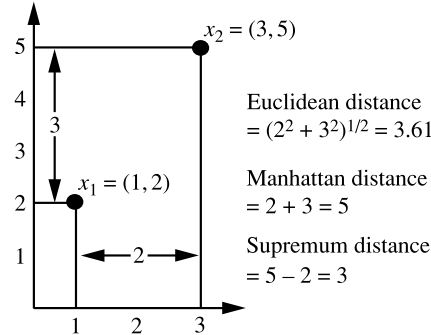


FIGURE 2.10

Euclidean, Manhattan, and supremum distances between two objects.

The **supremum distance** (also referred to as  $L_{max}$ ,  $L_{\infty}$  **norm**, and the **Chebyshev distance**) is a generalization of the Minkowski distance for  $h \rightarrow \infty$ . To compute it, we find the attribute  $f$  that gives the maximum difference in values between the two objects. This difference is the supremum distance, defined more formally as:

$$d(i, j) = \lim_{h \rightarrow \infty} \left( \sum_{f=1}^p |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_f |x_{if} - x_{jf}|. \quad (2.24)$$

The  $L_{\infty}$  norm is also known as the *uniform norm*.

**Example 2.21. Supremum distance.** Let's use the same two objects,  $x_1 = (1, 2)$  and  $x_2 = (3, 5)$ , as in Fig. 2.10. The second attribute gives the greatest difference between the values for the objects. That is,  $\max\{|3 - 1|, |5 - 2|\} = 3$ . This is the supremum distance between the two objects.  $\square$

If each attribute is assigned a weight according to its perceived importance, the **weighted Euclidean distance** can be computed as

$$d(i, j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_m|x_{ip} - x_{jp}|^2}. \quad (2.25)$$

Weighting can also be applied to other distance measures as well.

### 2.3.5 Proximity measures for ordinal attributes

The values of an ordinal attribute have a meaningful order or ranking about them, yet the magnitude between successive values is unknown (Section 2.1.3). An example includes the sequence *small*, *medium*, *large* for a *size* attribute. Ordinal attributes may also be obtained from the discretization of numeric attributes by splitting the value range into a finite number of categories. These categories are organized into ranks. That is, the range of a numeric attribute can be mapped to an ordinal attribute  $f$  having  $M_f$



states. For example, the range of the interval-scaled attribute *temperature* (in Celsius) can be organized into the following states:  $-30$  to  $-10$ ,  $-10$  to  $10$ , and  $10$  to  $30$ , representing the categories *cold temperature*, *moderate temperature*, and *warm temperature*, respectively. Let  $M_f$  represent the number of possible states that an ordinal attribute can have. These ordered states define the ranking  $1, \dots, M_f$ .

“How are ordinal attributes handled?” The treatment of ordinal attributes is quite similar to that of numeric attributes when computing dissimilarity between objects. Suppose that  $f$  is an attribute from a set of ordinal attributes describing  $n$  objects. The dissimilarity computation with respect to  $f$  involves the following steps:

1. The value of  $f$  for the  $i$ th object is  $x_{if}$ , and  $f$  has  $M_f$  ordered states, representing the ranking  $1, \dots, M_f$ . Replace each  $x_{if}$  by its corresponding rank,  $r_{if} \in \{1, \dots, M_f\}$ .
2. Since each ordinal attribute can have a different number of states, it is often necessary to map the range of each attribute onto  $[0.0, 1.0]$  so that each attribute has equal weight. We perform such data normalization by replacing the rank  $r_{if}$  of the  $i$ th object in the  $f$ th attribute by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \quad (2.26)$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 2.3.4 for numeric attributes, using  $z_{if}$  to represent the  $f$  value for the  $i$ th object.

**Example 2.22. Dissimilarity between ordinal attributes.** Suppose that we have the sample data shown earlier in Table 2.4, except that this time only the *object-identifier* and the continuous ordinal attribute, *test-2*, are available. There are three states for *test-2*: *fair*, *good*, and *excellent*, that is,  $M_f = 3$ . For step 1, if we replace each value for *test-2* by its rank, the four objects are assigned the ranks 3, 1, 2, and 3, respectively. Step 2 normalizes the ranking by mapping rank 1 to 0.0, rank 2 to 0.5, and rank 3 to 1.0. For step 3, we can use, say, the Euclidean distance defined in Eq. (2.21), which results in the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1.0 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}.$$

Therefore objects 1 and 2 are the most dissimilar, as are objects 2 and 4 (i.e.,  $d(2, 1) = 1.0$  and  $d(4, 2) = 1.0$ ). This makes intuitive sense since objects 1 and 4 are both *excellent*. Object 2 is *fair*, which is at the opposite end of the range of values for *test-2*.  $\square$

Similarity values for ordinal attributes can be interpreted from dissimilarity as  $sim(i, j) = 1 - d(i, j)$ .

### 2.3.6 Dissimilarity for attributes of mixed types

Sections 2.3.2 through 2.3.5 discussed how to compute the dissimilarity between objects described by attributes of the same type, where these types may be either *nominal*, *symmetric binary*, *asymmetric binary*, *numeric*, or *ordinal*. However, in many real databases, objects are described by a *mixture* of attribute types. In general, a database can contain all of these attribute types.

“So, how can we compute the dissimilarity between objects of mixed attribute types?” One approach is to group each type of attributes together, performing separate data mining (e.g., clustering) analysis for each type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate analysis per attribute type will generate compatible results.

A more preferable approach is to process all attribute types together, performing a single analysis. One such technique combines the different attributes into a single dissimilarity matrix, bringing all of the meaningful attributes onto a common scale of the interval [0.0, 1.0].

Suppose that the data set contains  $p$  attributes of mixed types. The dissimilarity  $d(i, j)$  between objects  $i$  and  $j$  is defined as

$$d(i, j) = \frac{\sum_{f=1}^p \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^p \delta_{ij}^{(f)}} \tag{2.27}$$

where the indicator  $\delta_{ij}^{(f)} = 0$  if either (1)  $x_{if}$  or  $x_{jf}$  is missing (i.e., there is no measurement of attribute  $f$  for object  $i$  or object  $j$ ), or (2)  $x_{if} = x_{jf} = 0$  and attribute  $f$  is asymmetric binary; otherwise,  $\delta_{ij}^{(f)} = 1$ . The contribution of attribute  $f$  to the dissimilarity between  $i$  and  $j$  (i.e.,  $d_{ij}^{(f)}$ ) is computed dependent on its type:

- If  $f$  is numeric:  $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_f - min_f}$ , where  $max_f$  and  $min_f$  are the maximum and minimum values of attribute  $f$ , respectively;
- If  $f$  is nominal or binary:  $d_{ij}^{(f)} = 0$  if  $x_{if} = x_{jf}$ ; otherwise,  $d_{ij}^{(f)} = 1$ ; and
- If  $f$  is ordinal: compute the ranks  $r_{if}$  and  $z_{if} = \frac{r_{if} - 1}{M_f - 1}$ , and treat  $z_{if}$  as numeric.

These steps are identical to what we have already seen for each of the individual attribute types. The only difference is for numeric attributes, where we normalize so that the values map to the interval [0.0, 1.0]. Thus the dissimilarity between objects can be computed even when the attributes describing the objects are of different types.

**Example 2.23. Dissimilarity between attributes of mixed types.** Let’s compute a dissimilarity matrix for the objects in Table 2.4. Now we will consider *all* of the attributes, which are of different types. In Examples 2.18 and 2.22, we worked out the dissimilarity matrices for each of the individual attributes. The procedures we followed for *test-1* (which is nominal) and *test-2* (which is ordinal) are the same as outlined earlier for processing attributes of mixed types. Therefore we can use the dissimilarity matrices obtained for *test-1* and *test-2* later when we compute Eq. (2.27). First, however, we need to compute the dissimilarity matrix for the third attribute, *test-3* (which is numeric). That is, we must compute  $d_{ij}^{(3)}$ . Following the case for numeric attributes, we let  $max_{hx_h} = 64$  and  $min_{hx_h} = 22$ . The difference between the two is used in Eq. (2.27) to normalize the values of the dissimilarity matrix. The resulting dissimilarity matrix for *test-3* is

$$\begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1.00 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}.$$

We can now use the dissimilarity matrices for the three attributes in our computation of Eq. (2.27). The indicator  $\delta_{ij}^{(f)} = 1$  for each of the three attributes,  $f$ . We get, for example,  $d(3, 1) = \frac{1(1) + 1(0.50) + 1(0.45)}{3} = 0.65$ . The resulting dissimilarity matrix obtained for the data described by the three attributes of mixed types is:

$$\begin{bmatrix} 0 & & & & \\ 0.85 & 0 & & & \\ 0.65 & 0.83 & 0 & & \\ 0.13 & 0.71 & 0.79 & 0 & \end{bmatrix}.$$

From Table 2.4, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for *test-1* and *test-2*. This is confirmed by the dissimilarity matrix, where  $d(4, 1)$  is the lowest value for any pair of different objects. Similarly, the matrix indicates that objects 1 and 2 are the least similar.  $\square$

### 2.3.7 Cosine similarity

**Cosine similarity** measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus each document is an object represented by what is called a *term-frequency vector*. For example, in Table 2.7, we see that *Document1* contains five instances of the word *team*, whereas *hockey* occurs three times. The word *coach* is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Term-frequency vectors are typically very long and **sparse** (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, and biological data analysis. The traditional distance measures that we have studied in this chapter do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents *do* have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

**Cosine similarity** is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let  $\mathbf{x}$  and  $\mathbf{y}$  be two vectors for

Document	Team	Coach	Hockey	Baseball	Soccer	Penalty	Score	Win	Loss	Season
<i>Document1</i>	5	0	3	0	2	0	0	2	0	0
<i>Document2</i>	3	0	2	0	1	1	0	1	0	1
<i>Document3</i>	0	7	0	2	1	0	0	3	0	0
<i>Document4</i>	0	1	0	0	1	2	2	0	3	0

comparison. Using the cosine measure as a similarity function, we have

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}, \quad (2.28)$$

where  $\|\mathbf{x}\|$  is the Euclidean norm of vector  $\mathbf{x} = (x_1, x_2, \dots, x_p)$ , defined as  $\sqrt{x_1^2 + x_2^2 + \dots + x_p^2}$ . Conceptually, it is the length of the vector. Similarly,  $\|\mathbf{y}\|$  is the Euclidean norm of vector  $\mathbf{y}$ . The measure computes the cosine of the angle between vectors  $\mathbf{x}$  and  $\mathbf{y}$ . A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors. Note that because the cosine similarity measure does not obey all of the properties of Section 2.3.4 defining metric measures, it is referred to as a *nonmetric measure*.

**Example 2.24. Cosine similarity between two term-frequency vectors.** Suppose that  $\mathbf{x}$  and  $\mathbf{y}$  are the first two term-frequency vectors in Table 2.7. That is,  $\mathbf{x} = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0)$  and  $\mathbf{y} = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1)$ . How similar are  $\mathbf{x}$  and  $\mathbf{y}$ ? Using Eq. (2.28) to compute the cosine similarity between the two vectors, we get:

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y} &= 5 \times 3 + 0 \times 0 + 3 \times 2 + 0 \times 0 + 2 \times 1 + 0 \times 1 + 0 \times 0 + 2 \times 1 \\ &\quad + 0 \times 0 + 0 \times 1 = 25 \\ \|\mathbf{x}\| &= \sqrt{5^2 + 0^2 + 3^2 + 0^2 + 2^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2} = 6.48 \\ \|\mathbf{y}\| &= \sqrt{3^2 + 0^2 + 2^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2} = 4.12 \\ \text{sim}(\mathbf{x}, \mathbf{y}) &= 0.94. \end{aligned}$$

Therefore if we were using the cosine similarity measure to compare these documents, they would be considered quite similar.  $\square$

When attributes are binary-valued, the cosine similarity function can be interpreted in terms of shared features or attributes. Suppose an object  $\mathbf{x}$  possesses the  $i$ th attribute if  $x_i = 1$ . Then  $\mathbf{x} \cdot \mathbf{y}$  is the number of attributes possessed (i.e., shared) by both  $\mathbf{x}$  and  $\mathbf{y}$ , and  $\|\mathbf{x}\|$  and  $\|\mathbf{y}\|$  are the *geometric mean* of the number of attributes possessed by  $\mathbf{x}$  and that by  $\mathbf{y}$  respectively. Thus,  $\text{sim}(\mathbf{x}, \mathbf{y})$  is a measure of relative possession of common attributes.

A simple variation of cosine similarity for the preceding scenario is

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\mathbf{x} \cdot \mathbf{x} + \mathbf{y} \cdot \mathbf{y} - \mathbf{x} \cdot \mathbf{y}}, \quad (2.29)$$

which is the ratio of the number of attributes shared by  $\mathbf{x}$  and  $\mathbf{y}$  to the number of attributes possessed by  $\mathbf{x}$  or  $\mathbf{y}$ . This function, known as the **Tanimoto coefficient** or **Tanimoto distance**, is frequently used in information retrieval and biology taxonomy.

### 2.3.8 Measuring similar distributions: the Kullback-Leibler divergence

Finally, we introduce *Kullback-Leibler divergence*, or simply, the *KL divergence*, a measure that has been popularly used in the data mining literature to measure the difference between two probability

distributions over the same variable  $x$ . This concept was originated in probability theory and information theory.

The KL divergence, which is closely related to *relative entropy*, *information divergence*, and *information for discrimination*, is a nonsymmetric measure of the difference between two probability distributions  $p(x)$  and  $q(x)$ . Specifically, the KL divergence of  $q(x)$  from  $p(x)$ , denoted  $D_{KL}(p(x)||q(x))$ , is a measure of the information loss when  $q(x)$  is used to approximate  $p(x)$ .

Let  $p(x)$  and  $q(x)$  be two probability distributions of a discrete random variable  $x$ . That is, both  $p(x)$  and  $q(x)$  sum up to 1, and  $p(x) > 0$  and  $q(x) > 0$  for any  $x$  in  $X$ .  $D_{KL}(p(x)||q(x))$  is defined in Eq. (2.30).

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)} \quad (2.30)$$

The KL divergence measures the expected number of extra bits required to code samples from  $p(x)$  when using a code based on  $q(x)$  rather than using a code based on  $p(x)$ . Typically  $p(x)$  represents the “true” distribution of data, observations, or a precisely calculated theoretical distribution. The measure  $q(x)$  typically represents a theory, model, description, or approximation of  $p(x)$ .

The continuous version of the KL divergence is

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx. \quad (2.31)$$

Although the KL divergence measures the “distance” between two distributions, it is not a distance measure. This is because that the KL divergence is not a metric measure. It is not symmetric: the KL from  $p(x)$  to  $q(x)$  is generally not the same as the KL from  $q(x)$  to  $p(x)$ . Furthermore, it need not satisfy triangular inequality. Nevertheless,  $D_{KL}(p(x)||q(x))$  is a nonnegative measure.  $D_{KL}(p(x)||q(x)) \geq 0$  and  $D_{KL}(p(x)||q(x)) = 0$  if and only if  $p(x) = q(x)$ .

Notice that attention should be paid when computing the KL divergence. We know  $\lim_{p(x) \rightarrow 0} p(x) \log p(x) = 0$ . However, when  $p(x) \neq 0$  but  $q(x) = 0$ ,  $D_{KL}(p(x)||q(x))$  is defined as  $\infty$ . This means that if one event  $e$  is possible (i.e.,  $p(e) > 0$ ), and the other predicts it is absolutely impossible (i.e.,  $q(e) = 0$ ), then the two distributions are absolutely different. However, in practice, two distributions  $P$  and  $Q$  are derived from observations and sample counting, that is, from frequency distributions. It is unreasonable to predict in the derived probability distribution that an event is completely impossible since we must take into account the possibility of unseen events. A *smoothing* method can be used to derive the probability distribution from an observed frequency distribution, as illustrated in the following example.

**Example 2.25. Computing the KL divergence by smoothing.** Suppose there are two sample distributions  $P$  and  $Q$  as follows:  $P : (a : 3/5, b : 1/5, c : 1/5)$  and  $Q : (a : 5/9, b : 3/9, d : 1/9)$ . To compute the KL divergence  $D_{KL}(P||Q)$ , we introduce a small constant  $\epsilon$ , for example  $\epsilon = 10^{-3}$ , and define a smoothed version of  $P$  and  $Q$ ,  $P'$  and  $Q'$ , as follows.

The sample set observed in  $P$ ,  $SP = \{a, b, c\}$ . Similarly,  $SQ = \{a, b, d\}$ . The union set is  $SU = \{a, b, c, d\}$ . By smoothing, the missing symbols can be added to each distribution accordingly, with the small probability  $\epsilon$ . Thus we have  $P' : (a : 3/5 - \epsilon/3, b : 1/5 - \epsilon/3, c : 1/5 - \epsilon/3, d : \epsilon)$  and  $Q' : (a : 5/9 - \epsilon/3, b : 3/9 - \epsilon/3, c : \epsilon, d : 1/9 - \epsilon/3)$ .  $D_{KL}(P', Q')$  can be computed easily.  $\square$

### 2.3.9 Capturing hidden semantics in similarity measures

Similarity measure is a fundamental concept in data mining. We have introduced multiple measures for computing similarities among objects consisting of numerical attribute, symmetric and asymmetric binary attribute, ordinal attribute, and nominal attribute. We have also introduced how to compute document similarity using the vector space model, and how to compare two distributions using the notion of KL divergence. These notions and measures on object similarity will be used substantially in our subsequent studies on methods for pattern discovery, classification, clustering, and outlier analysis.

In real-life applications, we may encounter the notion of object similarity beyond what we have discussed in this chapter. Even for simple objects, similarities among objects are often closely related to their semantic meanings, which cannot be captured based on the above defined similarity measures. For example, people often consider *geometry* and *algebra* are more similar than *geometry* vs. *music* or *politics*, even all are subjects studied in schools. Moreover, documents that consist of similar frequency distributions of words (or similar *bags* of words) may express rather different meanings (e.g., considering “The cat bites a mouse” vs. “The mouse bites a cat”). This goes beyond what a *vector space model* (i.e., expressing words as a set of vectors in a high-dimensional vector space as shown in Section 2.3.7) can handle. Furthermore, objects can be composed of rather complex structures and connections. Similarity measures for graphs and networks may need to be introduced, which is beyond the notions of object similarity introduced here.

In the upcoming chapters, we will introduce additional similarity measures when encountered along with the problems and methods to be discussed. In particular, in Chapter 12, we will briefly introduce the notion of distributive representation and representation learning, where text embedding and deep learning will be used to compute such advanced notion of similarities.

---

## 2.4 Data quality, data cleaning, and data integration

In this section, we start with a discussion of data quality measures (Section 2.4.1). Then, we introduce common techniques for data cleaning (Section 2.4.2) and data integration (Section 2.4.3).

### 2.4.1 Data quality measures

Data have quality if they satisfy the requirements of the intended use. There are many factors comprising **data quality**, including *accuracy*, *completeness*, *consistency*, *timeliness*, *believability*, and *interpretability*.

Imagine that you are a manager at an online webstore and have been charged with analyzing the company’s data with respect to your branch’s sales. You immediately set out to perform this task. You carefully inspect the company’s database and data warehouse, identifying and selecting the attributes or dimensions (e.g., *item*, *price*, and *units\_sold*) to be included in your analysis. Alas! You notice that several of the attributes for various tuples have no recorded values. For your analysis, you would like to include information as to whether each item purchased was advertised as on sale, yet you discover that this information has not been recorded. Furthermore, users of your database system have reported errors, unusual values, and inconsistencies in the data recorded for some transactions. In other words, the data you wish to analyze by data mining techniques are *incomplete* (lacking attribute values or certain attributes of interest, or containing only aggregate data); *inaccurate* or *noisy* (containing errors, or val-

ues that deviate from the expected); and *inconsistent* (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

This scenario illustrates three of the elements defining data quality: **accuracy**, **completeness**, and **consistency**. Inaccurate, incomplete, and inconsistent data are commonplace properties of large real-world databases and data warehouses. There are many possible reasons for inaccurate data (i.e., having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Users may purposely submit incorrect data values for mandatory fields when they do not wish to submit personal information (e.g., by choosing the default value “January 1” displayed for birthday). This is known as *disguised missing data*. Errors in data transmission can also occur. There may be technology limitations such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes or inconsistent formats for input fields (e.g., *date*). Duplicate tuples also require data cleaning.

Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because they were not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding or because of equipment malfunctions. Data that were inconsistent with other recorded data may have been deleted. Furthermore, the recording of the data history or modifications may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

Recall that data quality depends on the intended use of the data. Two different users may have very different assessments of the quality of a given database. For example, a marketing analyst may need to access the database mentioned before for a list of customer addresses. Some of the addresses are outdated or incorrect, yet overall, 80% of the addresses are accurate. The marketing analyst considers this to be a large customer database for target marketing purposes and is pleased with the database’s accuracy, although as sales manager, you found the data inaccurate.

**Timeliness** also affects data quality. Suppose that you are overseeing the distribution of monthly sales bonuses to the top sales representatives in a company. Several sales representatives, however, fail to submit their sales records on time at the month-end. There are also a number of corrections and adjustments that flow in after the month-end. For a period of time following each month, the data stored in the database are incomplete. However, once all of the data are received, it is correct. The fact that the month-end data are not updated in a timely fashion has a negative impact on the data quality.

Two other factors affecting data quality are believability and interpretability. **Believability** reflects how much the data are trusted by users, whereas **interpretability** reflects how easily the data are understood. Suppose that a database, at one point, had several errors, all of which have since been corrected. The past errors, however, had caused many problems for sales department users, and so they no longer trust the data. The data also use many accounting codes, which the sales department does not know how to interpret. Even though the database is now accurate, complete, consistent, and timely, sales department users may regard it as of low quality due to poor believability and interpretability.

## 2.4.2 Data cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* (or *data cleansing*) routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. In this section, you will study basic methods for data cleaning. First, we look

at ways of handling missing values. Then, we explain data smoothing techniques. Finally, we discuss approaches to data cleaning as a process.

### **Missing values**

Imagine that you need to analyze the sales and customer data of a company. You note that many tuples have no recorded value for several attributes such as customer *income*. How can you go about filling in the missing values for this attribute? Let's look at the following methods.

- 1. Ignore the tuple:** This is usually done when the class label is missing (assuming the mining task involves classification). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably. By ignoring the tuple, we do not make use of the remaining attributes' values in the tuple. Such data could have been useful to the task at hand.
- 2. Fill in the missing value manually:** In general, this approach is time consuming and may not be feasible given a large data set with many missing values.
- 3. Use a global constant to fill in the missing value:** Replace all missing attribute values by the same constant such as a label like "Unknown" or  $-\infty$ . If missing values are replaced by, say, "Unknown," then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of "Unknown." Hence, although this method is simple, it is not foolproof.
- 4. Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value:** Section 2.2 discussed measures of central tendency, which indicate the "middle" value of a data distribution. For normal (symmetric) data distributions, the mean can be used, whereas skewed data distribution should employ the median (Section 2.2). For example, suppose that the data distribution regarding the income of the customers is symmetric and that the mean income is \$56,000. Use this value to replace the missing value for *income*.
- 5. Use the attribute mean or median for all samples belonging to the same class as the given tuple:** For example, if classifying customers according to *credit\_risk*, we may replace the missing value with the mean *income* value for customers in the same credit risk category as that of the given tuple. If the data distribution for a given class is skewed, the median value is a better choice.
- 6. Use the most probable value to fill in the missing value:** This may be determined with regression, inference-based tools using a Bayesian formalism or decision tree induction. For example, using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for *income*. Decision trees, regression, and Bayesian inference are described in detail in Chapters 6 and 7.

Methods 3 through 6 bias the data—the filled-in value may not be correct. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the values of other attributes in its estimation of the missing value for *income*, there is a greater chance that the relationships between *income* and the other attributes are preserved.

It is important to note that, in some cases, a missing value may not imply an error in the data! For example, when applying for a credit card, candidates may be asked to supply their driver's license number. Candidates who do not have a driver's license may naturally leave this field blank. Forms should allow respondents to specify values such as "not applicable." Software routines may also be



used to uncover other null values (e.g., “don’t know,” “?”, or “none”). Ideally, each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed and/or how such values should be handled or transformed. Fields may also be intentionally left blank if they are to be provided in a later step of the business process. Hence, although we can try our best to clean the data after it is seized, good database and data entry procedure design should help minimize the number of missing values or errors in the first place.

### Noisy data

“What is noise?” **Noise** is a random error or variance in a measured variable. Given a numeric attribute such as, say, *price*, how can we “smooth” out the data to remove the noise? Let’s look at the following data smoothing techniques.

**Binning:** Binning methods smooth a sorted data value by consulting its “neighborhood,” that is, the values around it. The sorted values are distributed into a number of “buckets,” or *bins*. Because binning methods consult the neighborhood of values, they perform *local* smoothing. Fig. 2.11 illustrates some binning techniques. In this example, the data for *price* are first sorted and then partitioned into *equal-frequency* bins of size 3 (i.e., each bin contains three values). In **smoothing by bin means**, each value in a bin is replaced by the mean value of the bin. For example, the mean of the values 4, 8, and 15 in Bin 1 is 9. Therefore each original value in this bin is replaced by the value 9.

Similarly, **smoothing by bin medians** can be employed, in which each bin value is replaced by the bin median. In **smoothing by bin boundaries**, the minimum and maximum values in a given bin are identified as the *bin boundaries*. Each bin value is then replaced by the closest boundary value. In general, the larger the width, the greater the effect of the smoothing. Alternatively, bins may be

Sorted data for *price* (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

<p><b>Partition into (equal-frequency) bins:</b></p> <p>Bin 1: 4, 8, 15</p> <p>Bin 2: 21, 21, 24</p> <p>Bin 3: 25, 28, 34</p> <p><b>Smoothing by bin means:</b></p> <p>Bin 1: 9, 9, 9</p> <p>Bin 2: 22, 22, 22</p> <p>Bin 3: 29, 29, 29</p> <p><b>Smoothing by bin boundaries:</b></p> <p>Bin 1: 4, 4, 15</p> <p>Bin 2: 21, 21, 24</p> <p>Bin 3: 25, 25, 34</p>
---

FIGURE 2.11

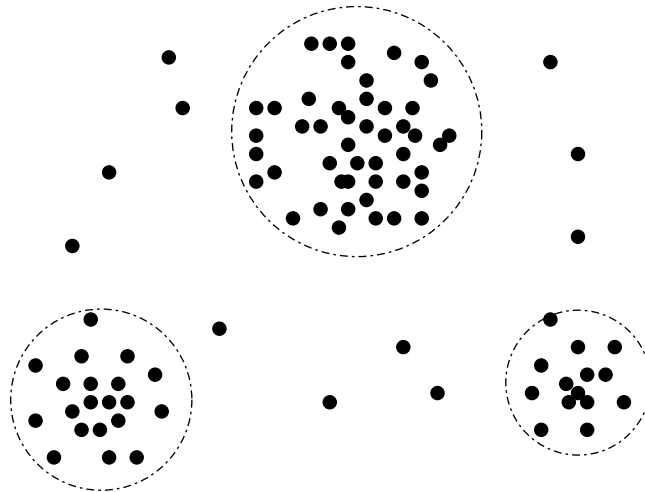
Data smoothing with different binning methods.

*equal width*, where the interval range of values in each bin is constant. Binning is also used as a discretization technique.

**Regression:** Data smoothing can also be done by regression, a technique that conforms data values to a function. *Linear regression* involves finding the “best” line to fit two attributes (or variables) so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved, and the data are fit to a multidimensional surface. Regression is further described in Chapter 6.

**Outlier analysis:** Outliers may be detected by clustering, for example, where similar values are organized into groups or “clusters.” Intuitively, values that fall outside of the set of clusters may be considered as outliers (Fig. 2.12). Chapter 11 is dedicated to the topic of outlier analysis.

Many data smoothing methods are also used for data discretization (a form of data transformation) and data reduction. For example, the binning techniques described before reduce the number of distinct values per attribute. This acts as a form of data reduction for logic-based data mining methods, such as decision tree induction, which repeatedly makes value comparisons on sorted data. Concept hierarchies are a form of data discretization that can also be used for data smoothing. A concept hierarchy for *price*, for example, may map real *price* values into *inexpensive*, *moderately\_priced*, and *expensive*, thereby reducing the number of data values to be handled by the mining process. Data discretization is discussed in Section 2.5.2. Some methods of classification have built-in data smoothing mechanisms. Classification is the topic of Chapters 6 and 7.



**FIGURE 2.12**

A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

### **Data cleaning as a process**

Missing values, noise, and inconsistencies contribute to inaccurate data. So far, we have looked at techniques for handling missing data and for smoothing data. “*But data cleaning is a big job. What about data cleaning as a process? How exactly does one proceed in tackling this task? Are there any tools out there to help?*”

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and inconsistent use of codes. Other sources of discrepancies include errors in instrumentation devices that record data and system errors. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).<sup>1</sup>

“*So, how can we proceed with discrepancy detection?*” As a starting point, use any knowledge you may already have regarding properties of the data. Such knowledge or “data about data” is referred to as **metadata**. This is where we can make use of the knowledge we gained about our data in the earlier sections. For example, what are the data type and domain of each attribute? What are the acceptable values for each attribute? The basic statistical data descriptions discussed in Section 2.2 are useful here to grasp data trends and identify anomalies. For example, find the mean, median, and mode values. Are the data symmetric or skewed? What is the range of values? Do all values fall within the expected range? What is the standard deviation of each attribute? For Gaussian-like distributions, values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. Are there any known dependencies between attributes? In this step, you may write your own scripts and/or use some of the tools that we discuss further later. From this, you may find noise, outliers, and unusual values that need investigation.

As a data analyst, you should be on the lookout for the inconsistent use of codes and any inconsistent data representations (e.g., “2010/12/25” and “25/12/2010” for *date*). **Field overloading** is another error source that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., an unused bit of an attribute that has a value range that uses only, say, 31 out of 32 bits).

The data should also be examined regarding uniqueness, consecutiveness, and null conditions. A **uniqueness rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutiveness rule** says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A **null condition rule** specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned earlier, reasons for missing values may include: (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license\_number* attribute left blank by nondrivers); (2) the data entry person does not know the correct value; or (3) the value is to be provided by a later step of the process. The null rule should specify how to record the null condition, for example, such as to store zero for numeric

<sup>1</sup> Data integration and the removal of redundant data that can result from such integration are further described in Section 2.4.3.

attributes, a blank for categorical attributes, or any other conventions that may be in use (e.g., entries like “don’t know” or “?” should be transformed to blank).

There are a number of different commercial tools that can aid in the discrepancy detection step. **Data scrubbing tools** use simple domain knowledge (e.g., knowledge of postal addresses and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. **Data auditing tools** find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. They are variants of data mining tools. For example, they may employ statistical analysis to find correlations, or clustering to identify outliers. They may also use the basic statistical data descriptions presented in Section 2.2.

Some data inconsistencies may be corrected manually using external references. For example, errors made at data entry may be corrected by performing a paper trace. Most errors, however, will require *data transformations*. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them.

Commercial tools can assist in the data transformation step. **Data migration tools** allow simple transformations to be specified such as to replace the string “gender” by “sex.” **ETL (extraction/transformation/loading) tools** allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transformations so that often we may also choose to write custom scripts for this step of the data cleaning process.

The two-step process of discrepancy detection and data transformation (to correct discrepancies) iterates. This process, however, is error-prone and time-consuming. Some transformations may introduce more discrepancies. Some *nested discrepancies* may only be detected after others have been fixed. For example, a typo such as “20010” in a year field may only surface once all date values have been converted to a uniform format. Transformations are often done as a batch process while the user waits without feedback. Only after the transformation is complete can the user go back and check that no new anomalies have been mistakenly created. Typically, numerous iterations are required before the user is satisfied. Any tuples that cannot be automatically handled by a given transformation are typically written to a file without any explanation regarding the reasoning behind their failure. As a result, the entire data cleaning process also suffers from a lack of interactivity.

New approaches to data cleaning emphasize increased interactivity. Potter’s Wheel, for example, is a publicly available data cleaning tool that integrates discrepancy detection and transformation. Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface. The transformations can be specified graphically or by providing examples. Results are shown immediately on the records that are visible on the screen. The user can choose to undo the transformations, so that transformations that have introduced additional errors can be “erased.” The tool automatically performs discrepancy checking in the background on the latest transformed view of the data. Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning. Section 2.5 will introduce some common data transformation techniques, including normalization, discretization, compression, and sampling.

Another approach to increasing interactivity in data cleaning is the development of declarative languages for the specification of data transformation operators. Such work focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently.

As we discover more about the data, it is important to keep updating the metadata to reflect this knowledge. This will help speed up data cleaning on future versions of the same data store.

### 2.4.3 Data integration

Data mining often requires data integration—the merging of data from multiple data stores. Careful integration can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent data mining process.

The semantic heterogeneity and structure of data pose great challenges in data integration. In this section, we first introduce the *entity identification problem*, which matches schema and objects from different sources. Then, we present *correlation tests* for spotting correlated numeric and nominal data. Finally, we introduce *tuple duplication* and the detection and resolution of *data value conflicts*.

#### **Entity identification problem**

It is likely that your data analysis task will involve *data integration*, which combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

There are a number of issues to consider during data integration. *Schema integration* and *object matching* can be tricky. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem**. For example, how can a data analyst or a computer be sure that *customer\_id* in one database and *cust\_number* in another refer to the same attribute? Moreover, metadata may be used to help entity identification (e.g., data codes for *pay\_type* in one database may be “H” and “S” but 1 and 2 in another). Examples of metadata for each attribute include the name, meaning, data type, range of values permitted for the attribute, and null rules for handling blank, zero, or null values (Section 2.4.2). Such metadata can be used to help avoid errors in schema integration. Hence, this step also relates to data cleaning, as described earlier.

When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. For example, in one system, a *discount* may be applied to the order, whereas in another system, it is applied to each individual line item within the order. If this is not caught before integration, items in the target system may be improperly discounted.

#### **Redundancy and correlation analysis**

*Redundancy* is another important issue in data integration. An attribute (such as *annual revenue*, for instance) may be redundant if it can be “derived” from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by **correlation analysis**. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For nominal data, we can use the  $\chi^2$  (*chi-square*) test. For numeric attributes, we can use the *correlation coefficient* and *covariance*, both of which assess how one attribute’s values vary from those of another.

### ***Tuple duplication***

In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level (e.g., where there are two or more identical tuples for a given unique data entry). The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy. Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all data occurrences. For example, if a purchase order database contains attributes for the purchaser's name and address instead of a key to this information in a purchaser database, discrepancies can occur, such as the same purchaser's name appearing with different addresses within the purchase order database.

### ***Data value conflict detection and resolution***

Data integration also involves the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. For instance, a *weight* attribute may be stored in metric units in one system and British imperial units in another. For a hotel chain, the *price* of rooms in different cities may involve not only different currencies but also different services (e.g., free breakfast) and taxes. When exchanging information between schools, for example, each school may have its own curriculum and grading scheme. One university may adopt a quarter system, offer three courses on database systems, and assign grades from A+ to F, whereas another may adopt a semester system, offer two courses on databases, and assign grades from 1 to 10. It is difficult to work out precise course-to-grade transformation rules between the two universities, making information exchange difficult.

Attributes may also differ on the abstraction level, where an attribute in one system is recorded at, say, a lower abstraction level than the “same” attribute in another. For example, the *total\_sales* in one database may refer to one branch of the company, whereas an attribute of the same name in another database may refer to the total sales for the stores in a given region.

The topic of discrepancy detection was described in Section 2.4.2 on data cleaning as a process.

---

## **2.5 Data transformation**

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Through appropriate data transformation, the resulting mining process may be more efficient, and the patterns found may be easier to understand. Various strategies for data transformation have been developed. In this section, we start with the introduction of data *normalization* (Section 2.5.1), where the attribute data are scaled so as to fall within a smaller range, such as  $-1.0$  to  $1.0$  or  $0.0$  to  $1.0$ . Then, we will learn data *discretization* (Section 2.5.2), which replaces the raw values of a numeric attribute (e.g., *age*) by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., *youth*, *adult*, *senior*). Data *compression* (Section 2.5.3) and *sampling* (Section 2.5.4) are two data reduction techniques that transform the input data to a reduced representation that is much smaller in volume, yet closely maintains the integrity of the original data.

### 2.5.1 Normalization

The measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for *height*, or from kilograms to pounds for *weight*, may lead to very different results. In general, expressing an attribute in smaller units will lead to a larger range for that attribute and thus tend to give such an attribute greater effect or “weight.” To help avoid dependence on the choice of measurement units, the data should be *normalized* or *standardized*. This involves transforming the data to fall within a smaller or common range such as  $[-1.0, 1.0]$  or  $[0.0, 1.0]$ . (The terms *standardize* and *normalize* are used interchangeably in data preprocessing, although in statistics, the latter term also has other connotations.)

Normalizing the data attempts to give all attributes an equal weight. Normalization is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbor classification and clustering. If using the neural network backpropagation algorithm for classification (Chapter 10), normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). It is also useful when given no prior knowledge of the data.

There are many methods for data normalization. We study *min-max normalization*, *z-score normalization*, and *normalization by decimal scaling*. For our discussion, let  $A$  be a numeric attribute with  $n$  observed values,  $v_1, v_2, \dots, v_n$ .

**Min-max normalization** performs a linear transformation on the original data. Suppose that  $\min_A$  and  $\max_A$  are the minimum and maximum values of an attribute,  $A$ . Min-max normalization maps a value,  $v_i$ , of  $A$  to  $v'_i$  in the range  $[\text{new\_min}_A, \text{new\_max}_A]$  by computing

$$v'_i = \frac{v_i - \min_A}{\max_A - \min_A}(\text{new\_max}_A - \text{new\_min}_A) + \text{new\_min}_A. \quad (2.32)$$

Min-max normalization preserves the relationships among the original data values. It will encounter an “out-of-bounds” error if a future input case for normalization falls outside of the original data range for  $A$ .

**Example 2.26. Min-max normalization.** Suppose that the minimum and maximum values for the attribute *income* are \$12,000 and \$98,000, respectively. We would like to map *income* to the range  $[0.0, 1.0]$ . By min-max normalization, a value of \$73,600 for *income* is transformed to  $\frac{73,600 - 12,000}{98,000 - 12,000}(1.0 - 0) + 0 = 0.716$ .  $\square$

In **z-score normalization** (or *zero-mean normalization*), the values for an attribute,  $A$ , are normalized based on the mean (i.e., average) and standard deviation of  $A$ . A value,  $v_i$ , of  $A$  is normalized to  $v'_i$  by computing

$$v'_i = \frac{v_i - \bar{A}}{\sigma_A}, \quad (2.33)$$

where  $\bar{A}$  and  $\sigma_A$  are the mean and standard deviation, respectively, of attribute  $A$ . The mean and standard deviation were discussed in Section 2.2, where  $\bar{A} = \frac{1}{n}(v_1 + v_2 + \dots + v_n)$ , and  $\sigma_A$  is computed as the square root of the variance of  $A$  (see Eq. (2.6)). This method of normalization is useful when the actual minimum and maximum of attribute  $A$  are unknown or when there are outliers that dominate the min-max normalization.

**Example 2.27. z-score normalization.** Suppose that the mean and standard deviation of the values for the attribute *income* are \$54,000 and \$16,000, respectively. With z-score normalization, a value of \$73,600 for *income* is transformed to  $\frac{73,600-54,000}{16,000} = 1.225$ .  $\square$

A variation of this z-score normalization replaces the standard deviation of Eq. (2.33) by the *mean absolute deviation* of  $A$ . The *mean absolute deviation* of  $A$ , denoted  $s_A$ , is

$$s_A = \frac{1}{n}(|v_1 - \bar{A}| + |v_2 - \bar{A}| + \cdots + |v_n - \bar{A}|). \quad (2.34)$$

Thus z-score normalization using the mean absolute deviation is

$$v'_i = \frac{v_i - \bar{A}}{s_A}. \quad (2.35)$$

The mean absolute deviation,  $s_A$ , is more robust to outliers than the standard deviation,  $\sigma_A$ . When computing the mean absolute deviation, the deviations from the mean (i.e.,  $|x_i - \bar{x}|$ ) are not squared; hence, the effect of outliers is somewhat reduced.

**Normalization by decimal scaling** normalizes by moving the decimal point of values of attribute  $A$ . The number of decimal points moved depends on the maximum absolute value of  $A$ . A value,  $v_i$ , of  $A$  is normalized to  $v'_i$  by computing

$$v'_i = \frac{v_i}{10^j}, \quad (2.36)$$

where  $j$  is the smallest integer such that  $\max(|v'_i|) < 1$ .

**Example 2.28. Decimal scaling.** Suppose that the recorded values of  $A$  range from  $-986$  to  $917$ . The maximum absolute value of  $A$  is  $986$ . To normalize by decimal scaling, we therefore divide each value by  $1000$  (i.e.,  $j = 3$ ) so that  $-986$  normalizes to  $-0.986$  and  $917$  normalizes to  $0.917$ .  $\square$

Note that normalization can change the original data quite a bit, especially when using z-score normalization or decimal scaling. It is also necessary to save the normalization parameters (e.g., the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

## 2.5.2 Discretization

Data discretization is a common data transformation technique, where the raw values of a numeric attribute (e.g., *age*) are replaced by interval labels (e.g.,  $0-10$ ,  $11-20$ , etc.) or conceptual labels (e.g., *youth*, *adult*, *senior*). The labels, in turn, can be recursively organized into higher-level concepts, resulting in a *concept hierarchy* for the numeric attribute. Fig. 2.13 shows a concept hierarchy for the attribute *price*. More than one concept hierarchy can be defined for the same attribute to accommodate the needs of various users.

Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut*



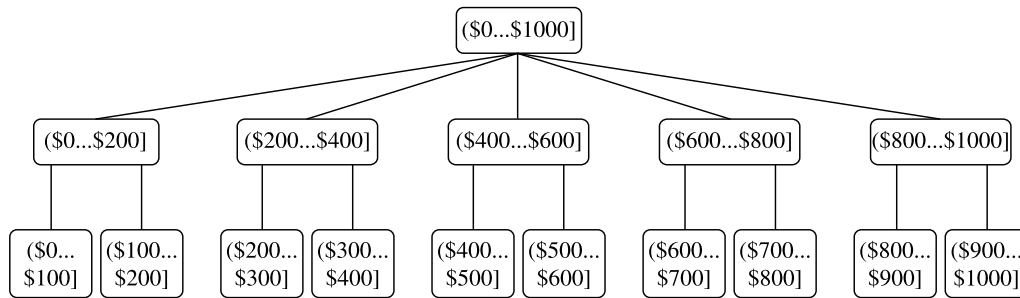


FIGURE 2.13

A concept hierarchy for the attribute *price*, where an interval  $(\$X \dots \$Y]$  denotes the range from  $\$X$  (exclusive) to  $\$Y$  (inclusive).

*points*) to split the entire attribute range and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals.

We introduce two basic discretization techniques, including binning and histogram analysis. Other methods for discretization include cluster analysis, decision tree analysis, and correlation analysis. Each of these techniques can be used to generate concept hierarchies for numeric attributes.

### ***Discretization by binning***

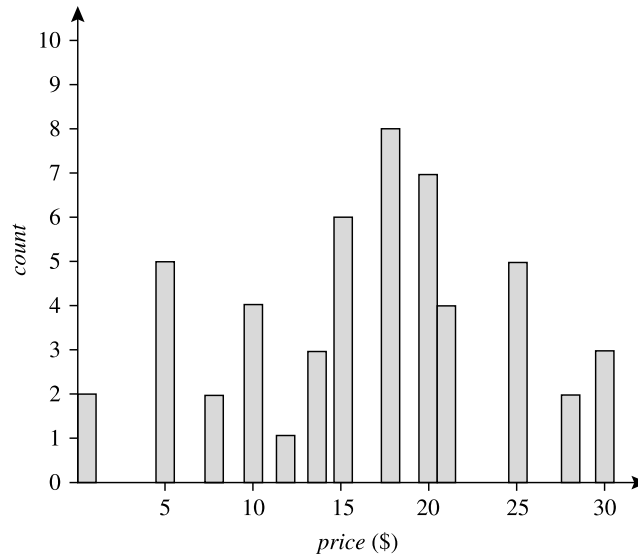
Binning is a top-down splitting technique based on a specified number of bins. Section 2.4.2 discussed binning methods for data smoothing. These methods are also used as discretization methods for data reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions to generate concept hierarchies.

Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

### ***Discretization by histogram analysis***

Histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms were introduced in Section 2.2.4. A histogram partitions the values of an attribute,  $A$ , into disjoint ranges called *buckets* or *bins*. If each bucket represents only a single attribute-value/frequency pair, the buckets are called *singleton buckets*. Singleton buckets are useful for storing high-frequency outliers. Often, buckets instead represent continuous ranges for the given attribute.

**Example 2.29.** The following data are a list of prices for commonly sold items in the company (rounded to the nearest dollar). The numbers have been sorted: 1, 1, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14,



**FIGURE 2.14**

A histogram for *price* using singleton buckets—each bucket represents one price–value/frequency pair.

14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 28, 28, 30, 30, 30.

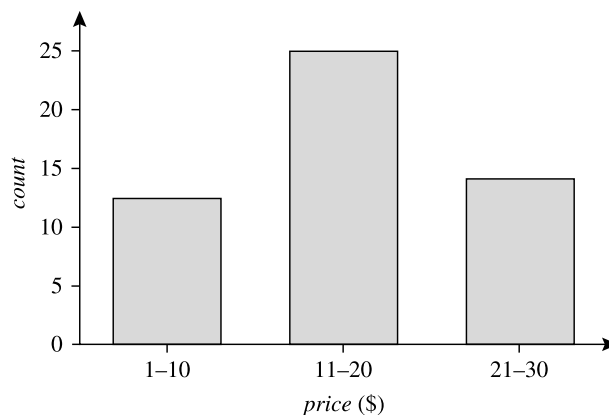
Fig. 2.14 shows a histogram for the data using singleton buckets. To further reduce the data, it is common to have each bucket denote a continuous value range for the given attribute. In Fig. 2.15, each bucket represents a different \$10 range for *price*. □

“How are the buckets determined and the attribute values partitioned?” There are several partitioning rules, including the following:

- **Equal-width:** In an equal-width histogram, the width of each bucket range is uniform (e.g., the width of \$10 for the buckets in Fig. 2.15).
- **Equal-frequency** (or equal-depth): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (i.e., each bucket contains roughly the same number of contiguous data samples).

Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described before for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. These histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for high dimensionalities.

The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to con-



**FIGURE 2.15**

An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of \$10.

control the recursive procedure. This specifies the minimum width of a partition or the minimum number of values for each partition at each level.

### 2.5.3 Data compression

In data compression, transformations are applied so as to obtain a reduced or “compressed” representation of the original data. If the original data can be *reconstructed* from the compressed data without any information loss, the data reduction is called **lossless**. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called **lossy**. There are several lossless algorithms for string compression; however, they typically allow only limited data manipulation. Dimensionality reduction techniques (Section 2.6) can also be considered as forms of data compression.

The **discrete wavelet transform (DWT)** is a linear signal processing technique that, when applied to a data vector  $\mathbf{x}$ , transforms it to a numerically different vector,  $\mathbf{x}'$ , of **wavelet coefficients**. The two vectors are of the same length. When applying this technique to data reduction, we consider each tuple as an  $n$ -dimensional data vector, that is,  $\mathbf{x} = (x_1, x_2, \dots, x_n)$ , depicting  $n$  measurements made on the tuple from  $n$  database attributes.

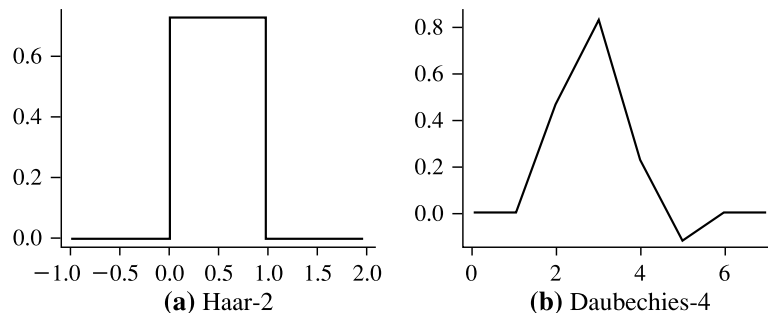
“How can this technique be useful for data reduction if the wavelet transformed data are of the same length as the original data?” The usefulness lies in the fact that the wavelet transformed data can be truncated. A compressed approximation of the data can be retained by storing only a small fraction of the strongest wavelet coefficients. For example, all wavelet coefficients larger than some user-specified threshold can be retained. All other coefficients are set to 0. The resulting data representation is therefore very sparse, so that operations that can take advantage of data sparsity are computationally very fast if performed in wavelet space. The technique also works to remove noise without smoothing out the main features of the data, making it effective for data cleaning as well. Given a set of coefficients, an approximation of the original data can be constructed by applying the *inverse* of the DWT used.

The DWT is closely related to the *discrete Fourier transform (DFT)*, a signal processing technique involving sines and cosines. In general, however, the DWT achieves better lossy compression. That is, if the same number of coefficients is retained for a DWT and a DFT of a given data vector, the DWT version will often provide a more accurate approximation of the original data. Hence, for an equivalent approximation, the DWT requires less space than the DFT. Unlike the DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several families of DWTs. Fig. 2.16 shows some wavelet families. Popular wavelet transforms include the Haar-2, Daubechies-4, and Daubechies-6. The general procedure for applying a discrete wavelet transform uses a hierarchical *pyramid algorithm* that halves the data at each iteration, resulting in fast computational speed. The method is as follows:

1. The length,  $L$ , of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary ( $L \geq n$ ).
2. Each transform involves applying two functions. The first applies some data smoothing, such as sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.
3. The two functions are applied to pairs of data points in  $X$ , that is, to all pairs of measurements  $(x_{2i}, x_{2i+1})$ . This results in two data sets of length  $L/2$ . In general, these represent a smoothed or low-frequency version of the input data and the high-frequency content of it, respectively.
4. The two functions are recursively applied to the data sets obtained in the previous iteration, until the resulting data sets obtained are of length 2.
5. Selected values from the data sets obtained in the previous iterations are designated as the wavelet coefficients of the transformed data.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients, where the matrix used depends on the given DWT. The matrix must be **orthonormal**, meaning that the columns are unit vectors and are mutually orthogonal, so that the matrix inverse is just its transpose. Although we do not have room to discuss it here, this property allows the reconstruction of the data from the smooth and smooth-difference data sets. Factoring the matrix used into a product



**FIGURE 2.16**

Examples of wavelet families. The number next to a wavelet name is the number of *vanishing moments* of the wavelet. This is a set of mathematical relationships that the coefficients must satisfy and is related to the number of coefficients.

of a few sparse matrices, the resulting “fast DWT” algorithm has a complexity of  $O(n)$  for an input vector of length  $n$ .

Wavelet transforms can be applied to multidimensional data such as a data cube. This is done by first applying the transform to the first dimension, then to the second, and so on. The computational complexity involved is linear with respect to the number of cells in the cube. Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes. Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard. Wavelet transforms have many real-world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

### 2.5.4 Sampling

Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random data sample (or subset). Suppose that a large data set,  $D$ , contains  $N$  tuples. Let’s look at the most common ways that we could sample  $D$  for data reduction.

- **Simple random sample without replacement (SRSWOR) of size  $s$ :** This is created by drawing  $s$  samples from  $D$ , and every time a sample is drawn, it is not to be placed back to the data set  $D$ .
- **Simple random sample with replacement (SRSWR) of size  $s$ :** This is similar to SRSWOR, except that each time a tuple is drawn from  $D$ , it is recorded and then *replaced*. That is, after a tuple is drawn, it is placed back in  $D$  so that it may be drawn again.
- **Cluster sample:** If the tuples in  $D$  are grouped into  $M$  mutually disjoint “clusters,” then a sample of  $s$  clusters can be obtained, where  $s < M$ . For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples. Other clustering criteria conveying rich semantics can also be explored. For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.
- **Stratified sample:** If  $D$  is divided into mutually disjoint parts called *strata*, a stratified sample of  $D$  is generated by obtaining a sample at each stratum. This helps ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where a stratum is created for each customer age group. In this way, the age group having the smallest number of customers will be sure to be represented.

An advantage of sampling for data reduction is that the cost of obtaining a sample is *proportional to the size of the sample,  $s$* , as opposed to  $N$ , the data set size. Hence, sampling complexity is potentially *sublinear* to the size of the data. Other data reduction techniques can require at least one complete pass through  $D$ . For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions,  $n$ , increases, whereas techniques using histograms, for example, could increase exponentially in  $n$ .

When applied to data reduction, sampling is most commonly used to estimate the answer to an aggregate query. It is possible (using the central limit theorem) to determine a sufficient sample size for estimating a given function within a specified degree of error. This sample size,  $s$ , may be extremely small in comparison to  $N$ . Sampling is a natural choice for the progressive refinement of a reduced data set. Such a set can be further refined by simply increasing the sample size.

---

## 2.6 Dimensionality reduction

Dimensionality reduction is the process of reducing the number of random variables or attributes or features under consideration. Dimensionality reduction methods include *principal components analysis* (PCA) (Section 2.6.1), which is a linear method that transforms or projects the original data onto a smaller space. *Attribute subset selection* is a method of dimensionality reduction in which irrelevant, weakly relevant, or redundant attributes or dimensions are detected and removed (Section 2.6.2). There are many nonlinear methods for dimensionality reduction (Section 2.6.3) such as kernel PCA and stochastic neighbor embedding.

### 2.6.1 Principal components analysis

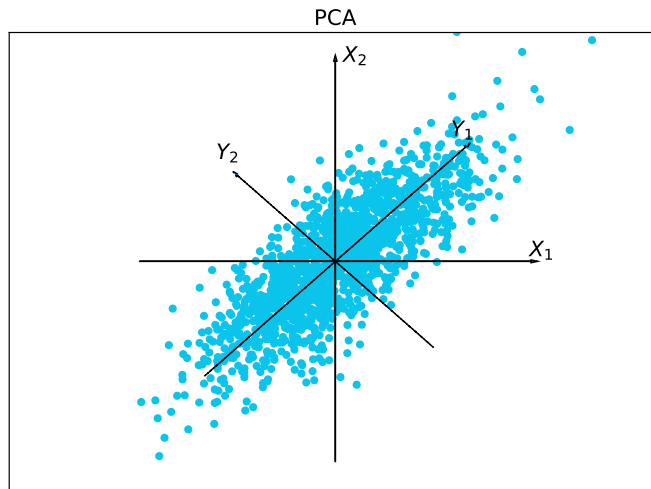
In this subsection, we provide an intuitive introduction to principal components analysis as a method of dimensionality reduction. A detailed theoretical explanation is beyond the scope of this book. For additional references, please see the bibliographic notes at the end of this chapter.

Suppose that the data to be reduced consist of tuples or data vectors described by  $d$  attributes or dimensions. **Principal components analysis (PCA)**; also called the Karhunen-Loeve, or K-L, method) searches for  $k$   $d$ -dimensional orthonormal vectors that can best be used to represent the data, where  $k \leq d$ . The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection (Section 2.6.2), which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA “combines” the essence of attributes by creating an alternative, smaller set of variables. The initial data can then be projected onto this smaller set. PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.

The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
2. PCA computes  $k$  orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that are perpendicular with each other. These vectors are referred to as the *principal components*. The input data are a linear combination of the principal components.
3. The principal components are sorted in order of decreasing “significance” or strength. The principal components essentially serve as a new set of axes for the data, providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on. For example, Fig. 2.17 shows the first two principal components,  $Y_1$  and  $Y_2$ , for the given set of data originally mapped to the axes  $X_1$  and  $X_2$ . This information helps identify groups or patterns within the data.
4. Because the components are sorted in descending order of “significance,” the data size can be reduced by eliminating the weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

PCA can be applied to ordered and unordered attributes and can handle sparse data and skewed data. Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions. Principal components may be used as inputs to multiple regression and cluster analysis.




---

**FIGURE 2.17**

Principal components analysis.  $Y_1$  and  $Y_2$  are the first two principal components for the given data.

### 2.6.2 Attribute subset selection

Data sets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant. For example, if the task is to classify customers based on whether or not they are likely to purchase a popular new music album when notified of a sale, attributes such as the customer's phone number are likely to be irrelevant, unlike attributes such as *age* or *music\_taste*. Although it may be possible for a domain expert to pick out some of the useful attributes, this can be a difficult and time-consuming task, especially when the data's behavior is not well known. (Hence, a reason behind its analysis!) Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, causing confusion for the mining algorithm employed. This can result in discovered patterns of poor quality. In addition, the added volume of irrelevant or redundant attributes can slow down the mining process.

**Attribute subset selection**<sup>2</sup> reduces the data set size by removing irrelevant or redundant attributes (or dimensions). This makes mining focused on the relevant dimensions. Mining on a reduced set of attributes has an additional benefit: It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

"How can we find a 'good' subset of the original attributes?" For  $d$  attributes, there are  $2^d$  possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as  $d$  and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically **greedy** in that, while searching through attribute space, they always make what looks to be the best choice at the time. Their strategy is to make a locally optimal choice in the hope that this will lead to a globally good

---

<sup>2</sup> In machine learning, attribute subset selection is known as *feature subset selection*.

Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  Initial reduced set: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  <pre> graph TD     A4["A4?"] -- Y --&gt; A1["A1?"]     A4 -- N --&gt; A6["A6?"]     A1 -- Y --&gt; C1_1((Class 1))     A1 -- N --&gt; C2_1((Class 2))     A6 -- Y --&gt; C1_2((Class 1))     A6 -- N --&gt; C2_2((Class 2))           </pre> $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$

FIGURE 2.18

Greedy (heuristic) methods for attribute subset selection.

solution. Such greedy methods are effective in practice and may come close to estimating an optimal solution.

The “best” (and “worst”) attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used such as the *information gain* measure used in building decision trees for classification.<sup>3</sup>

Basic heuristic methods of attribute subset selection include the following techniques, some of which are illustrated in Fig. 2.18.

- 1. Stepwise forward selection:** The procedure starts with an empty set of attributes as the reduced set. The best of the original attributes is determined and added to the reduced set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.
- 2. Stepwise backward elimination:** The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.
- 3. Combination of forward selection and backward elimination:** The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.
- 4. Decision tree induction:** Decision tree algorithms (e.g., ID3, C4.5, and CART) were originally intended for classification. Decision tree induction constructs a flowchart-like structure where each

<sup>3</sup> The information gain measure is described in detail in Chapter 6.



internal (nonleaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the “best” attribute to partition the data into individual classes.

When decision tree induction is used for attribute subset selection, a tree is constructed from the given data. All attributes that do not appear in the tree are assumed to be irrelevant. The set of attributes appearing in the tree form the reduced subset of attributes.

The stopping criteria for the methods may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

In some cases, we may want to create new attributes based on others. Such **attribute construction**<sup>4</sup> can help improve accuracy and understanding of structure in high-dimensional data. For example, we may wish to add the attribute *area* based on the attributes *height* and *width*. By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

### 2.6.3 Nonlinear dimensionality reduction methods

PCA is a linear method for dimensionality reduction in that each principal component is a linear combination of the original input attributes. This works well if the input data approximately follows a Gaussian distribution or forms a few linearly separable clusters. When the input data are linearly inseparable, however, PCA becomes ineffective. Luckily, there are many nonlinear methods we can resort to in this case.

#### *General procedure*

Suppose there are  $n$  data tuples  $\mathbf{x}_i$ , ( $i = 1, \dots, n$ ), each of which is represented by a  $d$ -dimensional attribute vector. How can we reduce the dimensionality to  $k$  where  $k \ll d$ ? In other words, we want to represent each of input data tuples by a  $k$ -dimensional attribute vector  $\hat{\mathbf{x}}_i$ , ( $i = 1, \dots, n$ ). Since  $k \ll d$ , we call the  $k$ -dimensional attribute vector  $\hat{\mathbf{x}}_i$ , ( $i = 1, \dots, n$ ) as low-dimensional representations of the original data tuples  $\mathbf{x}_i$ , ( $i = 1, \dots, n$ ).

For many nonlinear dimensionality reduction methods, they often follow the following two steps (see Fig. 2.19 for an illustration). In the first step (*constructing proximity matrix*), we construct an  $n \times n$  proximity matrix  $P$  whose entry  $P(i, j)$  ( $i, j = 1, \dots, n$ ) indicates the affinity or relevance between the two corresponding data tuples  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . In the second step (*preserving proximity*), we learn the new, low-dimensional representations of the input data tuples in the  $k$ -dimensional space  $\hat{\mathbf{x}}_i$  ( $i = 1, \dots, n$ ) so that the proximity matrix  $P$  constructed in the first step is somewhat preserved.

Depending on how the proximity matrix is constructed (Step 1) and how to preserve the constructed proximity matrix (Step 2), a variety of nonlinear dimensionality reduction techniques have been developed. Let’s look at two representative techniques below, including kernel PCA (KPCA) and stochastic hood embedding (SNE). A comparison of these two methods is summarized in Table 2.8.

<sup>4</sup> In the machine learning literature, attribute construction is known as *feature construction*.

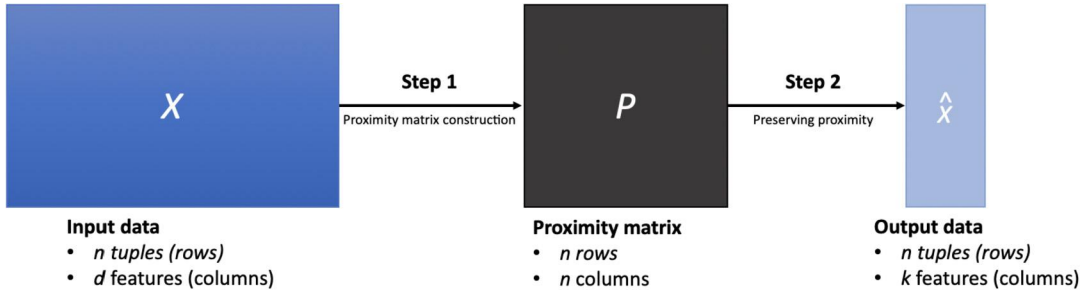


FIGURE 2.19

An illustration of nonlinear dimensionality reduction.

Table 2.8 Comparison of KPCA and SNE.		
	Step 1: Proximity Construction	Step 2: Preserving Proximity
KPCA	$P(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$	$\min \sum_{i,j=1}^n (P(i, j) - \hat{P}(i, j))^2 = \ P - \hat{P}\ _{fro}^2$
SNE	$P(i, j) = \frac{e^{-d_{ij}^2}}{\sum_{l=1, l \neq i}^n e^{-d_{il}^2}}$	$\min \sum_{i=1}^n \text{KL}(P_i    \hat{P}_i)$

### Kernel PCA

In kernel PCA (KPCA), we use a *kernel function*  $\kappa(\cdot)$  to construct the proximity matrix called kernel matrix (Step 1):  $P(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ ,  $(i, j = 1, \dots, n)$ . We will save the full details of kernel function  $\kappa(\cdot)$  to the later chapters (e.g., Chapter 7). In the simplest term, a kernel function computes the similarity of a pair of input data tuples in some high-dimensional, often nonlinear, space.

Meanwhile, we can also estimate such proximity (i.e., similarity) based on the learned low-dimensional representations:  $\hat{P}(i, j) = \hat{\mathbf{x}}_i \cdot \hat{\mathbf{x}}_j$ ,  $(i, j = 1, \dots, n)$  where  $\cdot$  is the vector inner product. What would be the best (i.e., optimal) low-dimensional representations  $\hat{\mathbf{x}}_i$ ,  $(i = 1, \dots, n)$ ? Intuitively we hope that the estimated proximity matrix  $\hat{P}$  is as close as possible to the kernel matrix  $P$ . This leads to the following optimization problem (Step 2), which says that the best low-dimensional representations should be those that minimize  $\sum_{i,j=1}^n (P(i, j) - \hat{P}(i, j))^2 = \|P - \hat{P}\|_{fro}^2$ , where  $\|\cdot\|_{fro}$  is the matrix Frobenius norm. We will not go into the mathematical details of how to solve this optimization problem. To make the long story short, it turns out the optimal low-dimensional representations  $\hat{\mathbf{x}}_i$ ,  $(i = 1, \dots, n)$  can be obtained by the top- $k$  eigenvectors and eigenvalues of the kernel matrix  $P$ . For a review of eigenvectors and eigenvalues, see Appendix A.

Typical choices for the kernel functions include (1) polynomial kernel:  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^p$  where  $p$  is the parameter, and (2) radial basis function (RBF)  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$ , where  $\sigma$  is the parameter. If we choose a linear kernel:  $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ , KPCA degenerates to the standard PCA.

### Stochastic neighbor embedding

In stochastic neighbor embedding (SNE), we first construct the proximity matrix  $P$  as follows:

$P(i, j) = \frac{e^{-d_{ij}^2}}{\sum_{l=1, l \neq i}^n e^{-d_{il}^2}}$ , where  $d_{ij}^2 = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}$  and  $\sigma$  is the parameter. We can view  $P(i, j)$  as the prob-

ability that data tuple  $x_j$  is the neighbor of data tuple  $x_i$ : the closer the two data tuples are (i.e., smaller  $d_{ij}$ ), the more likely  $x_j$  is the neighbor of  $x_i$ .<sup>5</sup>

Suppose we have learned the low-dimensional representations  $\hat{x}_i$ , ( $i = 1, \dots, n$ ). We can obtain another estimated proximity matrix in the similar way:  $\hat{P}(i, j) = \frac{e^{-\|\hat{x}_i - \hat{x}_j\|^2}}{\sum_{l=1, l \neq i}^n e^{-\|\hat{x}_i - \hat{x}_l\|^2}}$ . Again, the intuition is that if two data tuples share the similar low-dimensional representations (i.e., a small  $\|\hat{x}_i - \hat{x}_j\|$ ), the estimated proximity between them is large (i.e., a high  $\hat{P}(i, j)$ ). Now, in order to figure out the best low-dimensional representations  $\hat{x}_i$ , ( $i = 1, \dots, n$ ), we again seek those that make the estimated proximity  $\hat{P}$  be as close as possible to the proximity matrix  $P$ :  $P \approx \hat{P}$ .

Different from KPCA, in this case, each row of both matrices  $P$  and  $\hat{P}$  sums up to 1 and all the entries are nonnegative. In other words, each row of matrices  $P$  and  $\hat{P}$  is a probability distribution that tells the probability that each data tuple is the neighbor of a give data tuple. Naturally we can use KL divergence (see Section 2.3.8) to measure the difference between them, and the optimal low-dimensional representations  $\hat{x}_i$ , ( $i = 1, \dots, n$ ) are those that minimize the overall KL divergences between all rows of  $P$  and that of  $\hat{P}$ :  $\hat{x}_i = \arg \min_{\hat{x}_i, (i=1, \dots, n)} \sum_{i=1}^n D_{KL}(P_i || \hat{P}_i)$ , where  $P_i$  and  $\hat{P}_i$  are the  $i$ th rows of  $P$  and  $\hat{P}$ , respectively. Again, we will not go into the teeny weeny mathematical details of how to solve this optimization problem. Many off-the-shelf optimization packages can be used, such as gradient descent method.

A variant of SNE named t-SNE (t-distributed stochastic neighbor embedding) has been widely used to project the multi-dimensional representation produced by various deep learning models (Chapter 10) to a two- or three-dimensional space for the purpose of visualization.

Note that in the above introduction, we have omitted some implementation details of KPCA and SNE. For example, we need to ensure the data tuples are *centered* in KPCA; we often set  $P(i, i) = 0$  in SNE; and a variant of SNE constructs a symmetric proximity matrix  $P$ . Interested readers can refer to the related papers in the bibliographic notes.

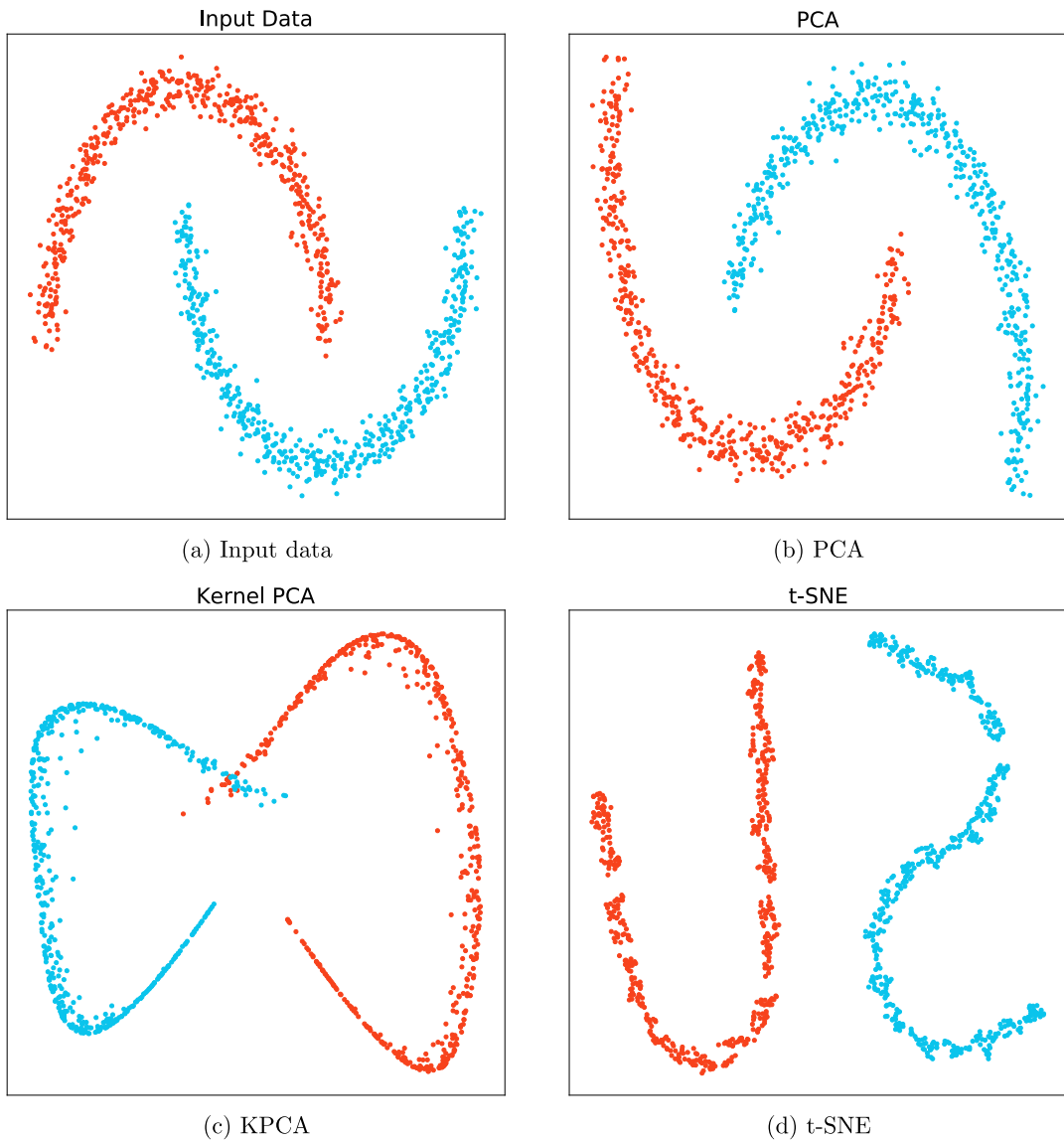
Let us look at an example.

**Example 2.30.** Given a collection of data tuples in 2-D space (Fig. 2.20(a)). The input data naturally form two clusters: one crescent shape facing up and one facing down. These two clusters are entangled with each other, and there is no way we can find a linear subspace (a linear line in this case) to separate them from each other. This means that no matter what kind of line we choose from the input space, if we project the original data tuples onto this line, the projected portions (i.e., the low-dimensional representation) will always be mixed with each other. This is what happens with PCA in Fig. 2.20(b), where we plot the projection of the input data onto the space spanned by two principal components. We can see that the two clusters are still mixed with each other, and the new representations by the principal components are essentially a linear rotation of the input data.

In contrast, using a nonlinear dimensionality reduction technique KPCA (Fig. 2.20(c)) or t-SNE (Fig. 2.20(d)), the two clusters are now better separated from each other in this new space.

Fig. 2.21 further shows the heatmaps of the similarity or proximity matrices in PCA (a), KPCA (b), and t-SNE (c), respectively. The two diagonal blocks indicate the proximity within the two clusters, respectively, and the two off-diagonal blocks indicate the proximity between the data from the two

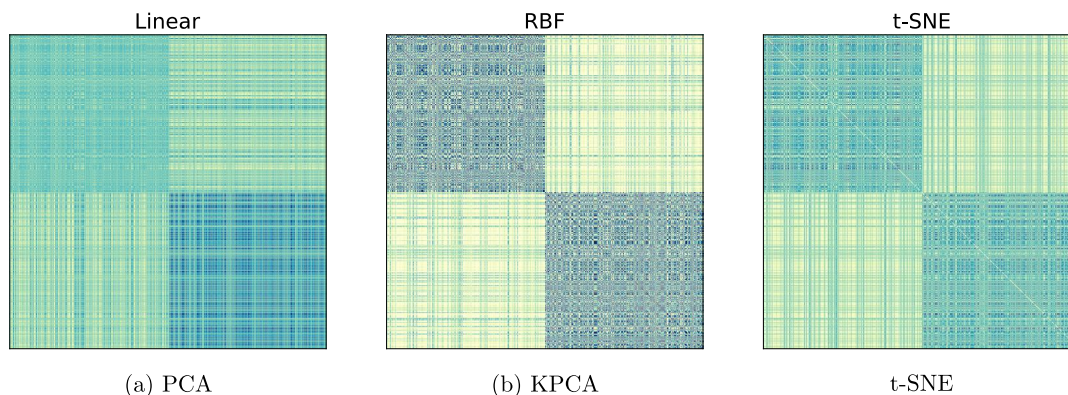
<sup>5</sup> An interesting observation is that the proximity matrix  $P$  here is the row-normalized kernel matrix in KPCA with the RBF kernel.

**FIGURE 2.20**

An example of linear vs. nonlinear dimensionality reduction methods.

clusters. We can see that, in general, by nonlinear methods (KPCA and t-SNE), the proximity between data tuples from the same cluster is much higher than the proximity between data tuples from different clusters. This in turn leads to better dimensionality reduction results than linear methods (e.g., PCA).

□

**FIGURE 2.21**

The heatmaps of the similarity or proximity matrices in PCA (a), KPCA (b), and t-SNE (c), respectively. The two diagonal blocks correspond to the two clusters in Fig. 2.20.

We can view PCA as the following process. First, we find principal components and *project* the original data tuples into the subspace spanned by the principal components. Then, we use the projected data tuples together with the principal components to *reconstruct* the original data tuples. This is a linear process in that both the projection step and the reconstruction step are linear operations. Using a specific deep learning technique called *autoencoder*, which will be introduced in Chapter 10, we can make both projection and reconstruction steps to be nonlinear. The output from such a nonlinear projection step thus forms the low-dimensional representations of the input data tuples.

PCA, attribute subset selection, KPCA, and SNE can be used as a data preprocessing step. That is, we first apply one of these techniques on the input data tuples to produce their low-dimensional representations *before* seeing the specific data mining task (e.g., classification, clustering, and outlier detection). We can also perform dimensionality reduction *together with* a specific data mining task. The rationality is that the dimensionality reduction and the corresponding data mining task are likely to mutually complement with each other. For example, when combining attribute subset selection with the classification task (called embedded feature selection), the classification model will guide the attribute selection process, and the selected features will in turn help build a better classification model; when combining dimensionality reduction with the clustering task, the clustering structure is likely to be more evident in the new, low-dimensional space, and meanwhile such a clustering structure will help find better low-dimensional representations. We will introduce such dimensionality reduction techniques in the chapter on classification.

Dimensionality reduction, we introduced in this section, and data compression and sampling methods introduced in the previous section are common data reduction techniques. Another type of data reduction technique is called **numerosity reduction**, which uses parametric or nonparametric models to obtain smaller representations of the original data. Parametric models store only the model parameters instead of the actual data. Examples include regression and log-linear models. Nonparametric methods include histograms, clustering, sampling, and data cube aggregation.

## 2.7 Summary

- Data sets are made up of data objects. A **data object** represents an entity. Data objects are described by attributes. Attributes can be nominal, binary, ordinal, or numeric.
- The values of a **nominal** (or **categorical**) **attribute** are symbols or names of things, where each value represents some kind of category, code, or state.
- **Binary attributes** are nominal attributes with only two possible states (such as 1 and 0 or true and false). If the two states are equally important, the attribute is *symmetric*; otherwise it is *asymmetric*.
- An **ordinal attribute** is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.
- A **numeric attribute** is *quantitative* (i.e., it is a measurable quantity) represented in integer or real values. Numeric attribute types can be *interval-scaled* or *ratio-scaled*. The values of an **interval-scaled attribute** are measured in fixed and equal units. **Ratio-scaled attributes** are numeric attributes with an inherent zero-point. Measurements are ratio-scaled in that we can speak of values as being an order of magnitude larger than the unit of measurement.
- **Basic statistical descriptions** provide the analytical foundation for data preprocessing. The basic statistical measures for data summarization include *mean*, *weighted mean*, *median*, and *mode* for measuring the central tendency of data; and *range*, *quantiles*, *quartiles*, *interquartile range*, *variance*, and *standard deviation* for measuring the dispersion of data. Graphical representations (e.g., *boxplots*, *quantile plots*, *quantile-quantile plots*, *histograms*, and *scatter plots*) facilitate visual inspection of the data and are thus useful for data preprocessing and mining.
- **Measures** of object **similarity** and **dissimilarity** are used in data mining applications such as clustering, outlier analysis, and nearest-neighbor classification. Such measures of *proximity* can be computed for each attribute type studied in this chapter, or for combinations of such attributes. Examples include the *Jaccard coefficient* for asymmetric binary attributes and *Euclidean*, *Manhattan*, *Minkowski*, and *supremum* distances for numeric attributes. For applications involving sparse numeric data vectors, such as term-frequency vectors, the *cosine measure* and the *Tanimoto coefficient* are often used in the assessment of similarity. To measure the difference between two probability distributions over the same variable  $x$ , *Kullback-Leibler divergence* (or the *KL divergence*) has been popularly used.  $D_{KL}(p(x)||q(x))$  measures the expected number of extra bits required to code samples from  $p(x)$  when using a code based on  $q(x)$  rather than using a code based on  $p(x)$ .
- **Data quality** is defined in terms of *accuracy*, *completeness*, *consistency*, *timeliness*, *believability*, and *interpretability*. These qualities are assessed based on the intended use of the data.
- **Data cleaning** routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually performed as an iterative two-step process consisting of discrepancy detection and data transformation.
- **Data integration** combines data from multiple sources to form a coherent data store. The resolution of semantic heterogeneity, metadata, correlation analysis, tuple duplication detection, and data conflict detection contribute to smooth data integration.
- **Data transformation** routines convert the data into appropriate forms for mining. For example, in **normalization**, attribute values are scaled; **data discretization** transforms numeric data by mapping values to interval or concept labels; and **data compression** and **data sampling**, as two typical data reduction techniques, transform the input data to a reduced representation.

- **Dimensionality reduction** reduces the number of random variables or attributes under consideration. Methods include *principal components analysis*, *attribute subset selection*, *kernel principal component analysis*, and *stochastic neighbor embedding*.

## 2.8 Exercises

- 2.1. Give three additional commonly used statistical measures that are not already illustrated in this chapter for the characterization of *data dispersion*, discuss how they can be computed efficiently in large databases.
- 2.2. Suppose that the data for analysis include the attribute *age*. The *age* values for the data tuples are (in ascending order) 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.
  - a. What is the *mean* of the data? What is the *median*?
  - b. What is the *mode* of the data? Comment on the data's modality (i.e., bimodal, trimodal, etc.).
  - c. What is the *midrange* of the data?
  - d. Can you find (roughly) the first quartile ( $Q_1$ ) and the third quartile ( $Q_3$ ) of the data?
  - e. Give the *five-number summary* of the data.
  - f. Show a *boxplot* of the data.
- 2.3. Suppose that the values for a given set of data are grouped into intervals. The intervals and corresponding frequencies are as follows:

Age	Frequency
1–5	200
6–15	450
16–20	300
21–50	1500
51–80	700
81–110	44

Compute an *approximate median* value for the data.

- 2.4. How is a *quantile-quantile plot* different from a *quantile plot*?
- 2.5. In our text, we state that the **variance** of  $N$  observations,  $x_1, x_2, \dots, x_N$  (when  $N$  is large), for a numeric attribute  $X$  is defined as

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \left( \frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2, \quad (2.37)$$

where  $\bar{x}$  is the mean value of the observations, as defined in Eq. (2.1). This is actually the formula for calculating the variance for the whole population using all the data (hence called the *population variance*). If we are calculation the variance using only a sample of data (hence called *sample variance*), we will need to use the following formula:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 = \frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - n\bar{x}^2 \right), \quad (2.38)$$

where  $n$  is size of the sample. With the sample size  $n$ , *sample standard deviation* can be defined similarly. Explain why there is such a minor difference at defining sample variance and population variance.

- 2.6. Reason why variance and standard deviation can be computed efficiently in very large data sets.
- 2.7. Suppose that a hospital tested the age and body fat data for 18 randomly selected adults with the following results:

<i>age</i>	23	23	27	27	39	41	47	49	50
<i>%fat</i>	9.5	26.5	7.8	17.8	31.4	25.9	27.4	27.2	31.2
<i>age</i>	52	54	54	56	57	58	58	60	61
<i>%fat</i>	34.6	42.5	28.8	33.4	30.2	34.1	32.9	41.2	35.7

- a. Calculate the mean, median, and standard deviation of *age* and *%fat*
- b. Draw the boxplots for *age* and *%fat*
- c. Draw a *scatter plot* and a *q-q plot* based on these two variables
- 2.8. Briefly outline how to compute the dissimilarity between objects described by the following:
- a. Nominal attributes
- b. Asymmetric binary attributes
- c. Numeric attributes
- d. Term-frequency vectors
- 2.9. Given two objects represented by the tuples (22, 1, 42, 10) and (20, 0, 36, 8):
- a. Compute the *Euclidean distance* between the two objects
- b. Compute the *Manhattan distance* between the two objects
- c. Compute the *Minkowski distance* between the two objects, using  $h = 3$
- d. Compute the *supremum distance* between the two objects
- 2.10. The *median* is one of the most important measures in data analysis. Propose several methods for median approximation. Analyze their respective complexity under different parameter settings and decide to what extent the real value can be approximated. Moreover, suggest a heuristic strategy to balance between accuracy and complexity, and then apply it to all methods you have given.
- 2.11. It is important to define or select similarity measures in data analysis. However, there is no commonly accepted subjective similarity measure. Results can vary depending on the similarity measures used. Nonetheless, seemingly different similarity measures may be equivalent after some transformation.
- Suppose we have the following 2-D data set:

	$A_1$	$A_2$
$x_1$	1.5	1.7
$x_2$	2	1.9
$x_3$	1.6	1.8
$x_4$	1.2	1.5
$x_5$	1.5	1.0

- a. Consider the data as 2-D data points. Given a new data point,  $x = (1.4, 1.6)$  as a query, rank the database points based on similarity with the query using Euclidean distance, Manhattan distance, supremum distance, and cosine similarity.



- b. Normalize the data set to make the norm of each data point equal to 1. Use Euclidean distance on the transformed data to rank the data points.
- 2.12. *Data quality* can be assessed in terms of several issues, including accuracy, completeness, and consistency. For each of the above three issues, discuss how data quality assessment can depend on the *intended use* of the data, giving examples. Propose two other dimensions of data quality.
- 2.13. In real-world data, tuples with *missing values* for some attributes are a common occurrence. Describe various methods for handling this problem.
- 2.14. Given the following data (in the ascending order) for the attribute *age*: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.
- Use *smoothing by bin means* to smooth these data, using equal-frequency bins of size 3. Illustrate your steps. Comment on the effect of this technique for the given data.
  - How might you determine *outliers* in the data?
  - What other methods are there for *data smoothing*?
- 2.15. Discuss issues to consider during *data integration*.
- 2.16. What are the value ranges of the following *normalization methods*?
- min-max normalization
  - z-score normalization
  - z-score normalization using the mean absolute deviation instead of standard deviation
  - normalization by decimal scaling
- 2.17. Use these methods to *normalize* the following group of data:

200, 300, 400, 600, 1000

- min-max normalization by setting  $new\_min = 0$  and  $new\_max = 1$
  - z-score normalization
  - z-score normalization using the mean absolute deviation instead of standard deviation
  - normalization by decimal scaling
- 2.18. Using the data for *age* given in Exercise 2.14, answer the following:
- Use min-max normalization to transform the value 35 for *age* onto the range [0.0, 1.0]
  - Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is 12.70 years
  - Use normalization by decimal scaling to transform the value 35 for *age*
  - Comment on which method you would prefer to use for the given data, giving reasons as to why
- 2.19. Using the data for *age* and *body fat* given in Exercise 2.7, answer the following:
- Normalize the two attributes based on *z-score normalization*
  - Calculate the *correlation coefficient* (Pearson's product moment coefficient). Are these two attributes positively or negatively correlated? Compute their covariance.
- 2.20. Suppose a group of 12 *sales price* records has been sorted as follows:

5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215.

Partition them into three bins by each of the following methods:

- Equal-frequency (equal-depth) partitioning
- Equal-width partitioning
- Clustering

- 2.21.** Use a flowchart to summarize the following procedures for *attribute subset selection*:
- Stepwise forward selection
  - Stepwise backward elimination
  - A combination of forward selection and backward elimination
- 2.22.** Using the data for *age* given in Exercise 2.14,
- Plot an equal-width histogram of width 10
  - Sketch examples of each of the following sampling techniques: SRSWOR, SRSWR, cluster sampling, and stratified sampling, using samples of size 5 and the strata “youth,” “middle-aged,” and “senior”
- 2.23.** Robust data loading poses a challenge in database systems because the input data are often dirty. In many cases, an input record may miss multiple values; some records could be *contaminated*, with some data values out of range or of a different data type than expected. Work out an automated *data cleaning and loading* algorithm so that the erroneous data will be marked and contaminated data will not be mistakenly inserted into the database during data loading.

---

## 2.9 Bibliographic notes

Data description, statistical data measurements, and descriptive data characterization have been introduced in most statistics introductory textbooks. For statistics-based visualization of data using boxplots, quantile plots, quantile-quantile plots, scatter plots, and loess curves, see Cleveland [Cle93].

Similarity and distance measures among various variables have been introduced in many textbooks that study cluster analysis, including Hartigan [Har75]; Jain and Dubes [JD88]; Kaufman and Rousseeuw [KR90]; Arabie, Hubert, and de Soete [AHS96]. Methods for combining attributes of different types into a single dissimilarity matrix were introduced by Kaufman and Rousseeuw [KR90].

Data preprocessing is discussed in a number of textbooks, including Pyle [Py199], Loshin [Los01], Redman [Red01], and Dasu and Johnson [DJ03], and García, Luengo, and Herrera [GLH15], and Luengo et al. [LGGRG+20].

For discussion regarding data quality, see Redman [Red01]; Wang, Storey, and Firth [WSF95]; Wand and Wang [WW96]; Ballou and Tayi [BT99]; and Olson [Ols03]. Potter’s Wheel, an interactive data cleaning tool described in Section 2.4.2, is presented by Raman and Hellerstein [RH01]. An example of the development of declarative languages for the specification of data transformation operators is given by Galhardas et al. [GFS+01]. The handling of missing attribute values is discussed by Friedman [Fri77]; Breiman, Friedman, Olshen, and Stone [BFOS84]; and Quinlan [Qui89]. Hua and Pei [HP07] present a heuristic approach to cleaning *disguised missing data*, where such data are captured when users falsely select default values on forms (e.g., “January 1” for *birthdate*) when they do not want to disclose personal information.

A method for the detection of outlier or “garbage” patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [GMV96]. Binning and data normalization are treated in many texts, including Kennedy et al. [KLV+98], Weiss and Indurkha [WI98], and Pyle [Py199]. Systems that include attribute (or feature) construction include BACON by Langley, Simon, Bradshaw, and Zytkow [LSBZ87]; Stagger by Schlimmer [Sch86]; FRINGE by Pagallo [Pag89]; and AQ17-DCI by Bloedorn and Michalski [BM98a]. Attribute construction is also described in Liu and Motoda [LM98]. Dasu et al.

build a BELLMAN system and propose a set of interesting methods for building a data quality browser by mining database structures [DJMS02].

A survey of data reduction techniques can be found in Barbará et al. [BDF<sup>+</sup>97]. For algorithms on data cubes and their precomputation, see Sarawagi and Stonebraker [SS94]; Agarwal et al. [AAD<sup>+</sup>96]; Harinarayan, Rajaraman, and Ullman [HRU96]; Ross and Srivastava [RS97]; and Zhao, Deshpande, and Naughton [ZDN97]. Attribute subset selection (or *feature subset selection*) is described in many texts such as Neter, Kutner, Nachtsheim, and Wasserman [NKNW96]; Dash and Liu [DL97]; and Liu and Motoda [LM98]. A combination of forward selection and backward elimination method is proposed by Siedlecki and Sklansky [SS88]. A wrapper approach to attribute selection is described by Kohavi and John [KJ97]. Unsupervised attribute subset selection is described by Dash, Liu, and Yao [DLY97].

For a general introduction to histograms, see Barbará et al. [BDF<sup>+</sup>97] and Devore and Peck [DP97]. For extensions of single-attribute histograms to multiple attributes, see Muralikrishna and DeWitt [MD88], and Poosala and Ioannidis [PI97].

There are many methods for assessing attribute relevance. Each has its own bias. The information gain measure is biased toward attributes with many values. Many alternatives have been proposed, such as gain ratio (Quinlan [Qui93]), which considers the probability of each attribute value. Other relevance measures include the Gini index (Breiman, Friedman, Olshen, and Stone [BFOS84]), the  $\chi^2$  contingency table statistic, and the uncertainty coefficient (Johnson and Wichern [JW92]). For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [BN92]. For additional methods, see Liu and Motoda [LM98], Dash and Liu [DL97], and Almuallim and Dietterich [AD91].

Liu et al. [LHTD02] perform a comprehensive survey of data discretization methods. Entropy-based discretization with the C4.5 algorithm is described by Quinlan [Qui93]. In Catlett [Cat91], the D-2 system binarizes a numeric feature recursively. ChiMerge by Kerber [Ker92] and Chi2 by Liu and Setiono [LS95] are methods for the automatic discretization of numeric attributes that both employ the  $\chi^2$  statistic.

For a description of wavelets for dimensionality reduction, see Press, Teukolosky, Vetterling, and Flannery [PTVF07]. A general account of wavelets can be found in Hubbard [Hub96]. For a list of wavelet software packages, see Bruce, Donoho, and Gao [BDG96]. Daubechies transforms are described by Daubechies [Dau92]. Routines for PCA are included in most statistical software packages such as SAS (<http://www.sas.com>). An introduction of KPCA can be found in [MSS<sup>+</sup>98] by Mika, Schölkopf, and Smola. Stochastic neighbor embedding is proposed by Hinton and Roweis [HR02]. A comparative review on dimensionality reduction is by van der Maaten et al. [vdMPvdH09].

# Data warehousing and online analytical processing

**Data analytics**, also often known as *business intelligence*, is the strategies and technology that enable enterprises to gain deep and actionable insights into business data. Data mining plays the core role in data analytics and business intelligence. Fundamentally, *data warehouses* generalize and consolidate data in multidimensional space. The construction of data warehouses involves data cleaning, data integration, and data transformation, and can be viewed as an important preparation step for data mining. Moreover, data warehouses provide *online analytical processing (OLAP)* tools for interactive analysis of multidimensional data of varied granularities, which facilitates effective data generalization and data mining. Many other data mining functions, such as association, classification, prediction, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction. OLAP tools typically use *data cube*, a multidimensional data model, to provide flexible access to summarized data. Data lakes as enterprise information infrastructure collect extensive data in enterprises and integrate metadata so that data exploration can be conducted effectively. Hence, data warehouses, OLAP, data cubes, and data lakes have become essential data and information backbone for enterprises. This chapter presents an in-depth and comprehensive introduction to data warehouse, OLAP, data cube, and data lake technology. This overview is essential for understanding the overall data mining and knowledge discovery process and practical applications. In addition, it can serve as a well-informed introduction to data analytics and business intelligence.

In this chapter, we first study a well-accepted definition of the data warehouse, introduce the architecture, and discuss the concept of data lake (Section 3.1). We then study the logical design of a data warehouse as a multidimensional data model (Section 3.2). Next, we look at OLAP operations and how to index OLAP data for efficient analytics (Section 3.3). Last, we introduce the techniques of building data cube as a way of implementing a data warehouse (Section 3.4).

---

## 3.1 Data warehouse

This section introduces data warehouses. We begin with a definition of data warehouses and explain how data warehouses can serve as the foundation of business intelligence (Section 3.1.1). Next, we discuss data warehouse architecture (Section 3.1.2). Last, we discuss data lakes (Section 3.1.3).

### 3.1.1 Data warehouse: what and why?

More often than not, data in organizations are recorded at the operational level. For example, for the sake of business efficiency, an e-commerce company often records the details of customer transactions in a table, the information about customers in another table, and the particulars about product suppliers

in a third table. Operational data are mainly concerned about individual business functionings, such as a purchase transaction, the registration of a new customer, and the shipment of a batch of products to a store. The major advantage is that business operations, such as a customer purchasing a product, can be conducted efficiently by inserting, deleting, or modifying only one or several records in one or a small number of tables, and thus many business operations can be conducted concurrently.

At the same time, business analysts and executives often focus on historical, current, and predictive views of business operations instead of individual transaction details. For example, a business analyst in an e-commerce company may want to investigate the categories of customers, such as their demographical groups, who spend the most last month, and the major categories of products they purchase. Computing answers to such analytic questions is often time and resource consuming, since it has to join multiple data tables and conduct a large number of group-by aggregation operations, and thus needs exclusive access to the data. Many analysis tasks may be periodic and some may be ad hoc and thus may severely affect business operations, which are expected to be online, frequent, and concurrent.

To address the gap between business operations and analysis, data warehousing provides architectures and tools for business analysts and executives to systematically organize, understand, and use their data to make strategic decisions. Data warehouse systems are valuable tools in today's competitive, fast-evolving world. In the last two decades, many firms have spent billions of dollars in building enterprise-wide data warehouses. It is well recognized that, with competition mounting in every industry, data warehousing is the must-have business infrastructure—a way to retain customers by learning more about their demands and behavior.

“Then, what exactly is a data warehouse?” In general, a data warehouse refers to a data repository that is specific for analysis and is maintained separately from an organization's operational databases. Data warehouse systems support information processing by providing a solid platform of consolidated historic data for analysis.

According to William H. Inmon, a leading architect in construction of data warehouse systems, “A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision making process” [Inm96]. This short but comprehensive definition presents the major features of a data warehouse. The four keywords—*subject-oriented*, *integrated*, *time-variant*, and *nonvolatile*—distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems, and file systems.

- **Subject-oriented:** A data warehouse is organized around major subjects that are often identified enterprise or department wise, such as customer, supplier, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on modeling and analyzing data for decision makers. Hence, data warehouses typically provide a simple and concise view of particular subject issues by excluding data that are not useful in the decision support process.
- **Integrated:** A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and online transaction records. Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.
- **Time-variant:** Data are stored to provide information from a historic perspective (e.g., the past 5–10 years). Every key structure in a data warehouse contains, either implicitly or explicitly, a time element. In other words, a data warehouse typically records data crossing a substantial history of time.

- **Nonvolatile:** A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require strong transaction processing, recovery, and concurrency control mechanisms and thus has no interference with the operational systems. It usually requires only two operations in data accessing: *initial loading of data* and *access of data*. In other words, the data stored in a data warehouse are typically not deleted.

In sum, a data warehouse is a semantically consistent and persistent data store that serves as a physical implementation of a decision support data model. It stores the information that an enterprise needs to make strategic decisions. A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/or ad hoc queries, analytical reporting, and decision making. Correspondingly, **data warehousing** is the process of constructing and using data warehouses. The construction of a data warehouse requires data cleaning, data integration, and data consolidation.

“*How do organizations use information from data warehouses?*” Many organizations use this information to support business decision-making activities. For example, by identifying the groups of most active customers an e-commerce company can design promotion campaigns to retain those customers firmly. By analyzing the sales patterns of products in different seasons, a company may design supply chain strategies to reduce the stocking cost of seasonal products. Analytic results from data warehouses are often presented to analysts and decision makers through periodic or ad hoc reports, such as daily, weekly, and monthly sales analysis reports analyzing sales patterns on customer groups, regions, products, and promotions.

“*What are the major differences between operational database systems and data warehouses?*” The major task of traditional operational database systems is to perform **online transaction processing (OLTP)**. These OLTP systems cover most of the day-to-day operations of an organization, such as purchasing, inventory, manufacturing, banking, payroll, registration, and accounting. Data warehouse systems serve business analysts and executives (in general, also known as *knowledge workers*) in the role of obtaining business insights and making decisions by organizing and presenting data in various perspectives in order to accommodate the diverse needs from different users. These systems are known as **online analytical processing (OLAP)** systems.

The major distinguishing features of OLTP and OLAP are as follows:

- **Users and system orientation:** An OLTP system is *transaction-oriented* and is used for operation execution by clerks and clients. An OLAP system is *business insight-oriented* and is used for data summarization and analysis by knowledge workers, including managers, executives, and analysts.
- **Data contents:** An OLTP system manages current data that are typically too detailed to be easily used for business decision making. An OLAP system manages large amounts of historic data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity, such as weekly-monthly-annually. These features make data easier to be used for informed decision making.
- **Database design:** An OLTP system usually adopts an entity-relationship (ER) data model and an application-oriented database design. An OLAP system typically adopts either a *star* model or a *snowflake* model (see Section 3.2.2) and a subject-oriented database design.
- **View:** An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historic data or data in different organizations. In contrast, an OLAP system often

spans multiple versions of a database schema, due to the evolutionary process of an organization. OLAP systems also deal with information that originates from different organizations, integrating information from many data stores.

- **Access patterns:** The access patterns of an OLTP system consist mainly of short, atomic transactions, such as transferring an amount from one account to another. Such a system requires concurrency control and recovery mechanisms. However, accesses to OLAP systems are mostly read-only operations (because most data warehouses store historic rather than up-to-date information). Many accesses may be complex queries.

*“Why not perform OLAP directly on operational databases instead of constructing a separate data warehouse?”* A major reason for a separation is to ensure the *high performance of both systems*. An operational database is designed and tuned from known tasks and workloads like indexing and hashing using primary keys, searching for particular records, and optimizing “canned” queries, which are pre-programmed and frequently used queries in business. OLAP queries, however, are often complex. They involve the computation of large data groups at summarized levels and may require the use of special data organization, access, and implementation methods based on multidimensional views. Processing OLAP queries directly in operational databases may substantially jeopardize the performance of operational tasks. An operational database supports the concurrent processing of multiple transactions. Concurrency control and recovery mechanisms (e.g., locking and logging) are required to ensure the consistency and robustness of transactions. An OLAP query often needs read-only access of massive data records for summarization and aggregation. Concurrency control and recovery mechanisms, if applied for such OLAP operations, may seriously delay the execution of concurrent transactions and thus substantially reduce the throughput of an OLTP system.

Finally, the separation of operational databases from data warehouses is based on the different structures, contents, and uses of the data in these two kinds of systems. Decision support requires historic data, whereas operational databases do not typically maintain historic data. In this context, the data in operational databases are usually far from complete for decision making. Decision support requires consolidation (e.g., aggregation and summarization) of data from heterogeneous sources, resulting in high-quality, clean, and integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis. Because the two systems provide quite different functionalities and require different kinds of data, it is presently necessary to maintain separate databases.

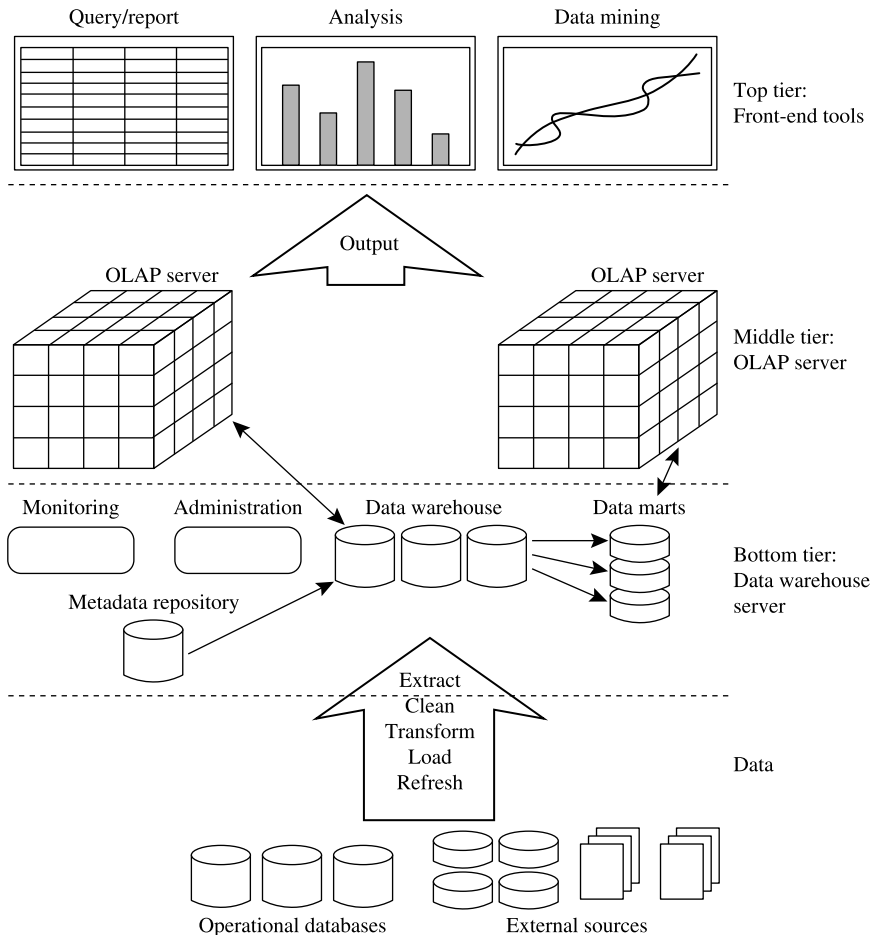
### 3.1.2 Architecture of data warehouses: enterprise data warehouses and data marts

*“What does the architecture of a data warehouse look like?”* To answer this question, we first introduce the general three-tier architecture of data warehouses and then discuss two major data warehouse models: the enterprise warehouse and the data mart.

#### ***The three-tier architecture***

Data warehouses often adopt a three-tier architecture, as shown in Fig. 3.1.

The bottom level is a **warehouse database server** that is typically a main-stream database system, such as a relational database or a key-value store. Back-end tools and data extraction/transformation/loading (ETL) utilities are used to feed data into the bottom tier from operational databases or

**FIGURE 3.1**

A three-tier data warehousing architecture.

other external sources (e.g., customer profile information provided by external partners). These tools and utilities perform data extraction, cleaning, and transformation, as well as load and refresh functions to update the data warehouse. This tier also contains a metadata repository, which stores information about the data warehouse and its contents.

The middle tier is an **OLAP server** that is typically implemented using either a **relational OLAP (ROLAP)** model (i.e., an extended relational DBMS that maps operations on multidimensional data to standard relational operations) or a **multidimensional OLAP (MOLAP)** model (i.e., a special-purpose server that directly implements multidimensional data and operations). We will discuss OLAP servers in detail soon.



The top tier is a **front-end client layer**, which contains tools for querying, reporting, visualization, analysis, and/or data mining, such as trend analysis and prediction.

“*What are metadata in a warehouse database server?*” **Metadata** are data about data. When used in a data warehouse, metadata are the data that defines warehouse objects. Metadata are created for the data names and definitions of the given warehouse. Additional metadata may be created and captured for timestamping any extracted data, the source of the extracted data, and missing fields that have been added by data cleaning or integration processes.

In addition, a metadata repository may contain a description of the *data warehouse structure* (e.g., the schema, view, dimensions, derived data definition, etc.), *operational metadata* (e.g., data transformation lineage, freshness of data), definitions of data summarization, mapping from operational data to the data warehouse, system information, and related business information.

Metadata play a very different role than other data warehouse data and are important for many reasons. For example, metadata are used as a directory to help analysts locate the contents of a data warehouse and as a guide to data mapping when data are transformed from the operational environment to the data warehouse environment. Metadata also serve as a guide to the algorithms used for summarization between the current detailed data and the lightly summarized data and between the lightly summarized data and the highly summarized data. Metadata should be stored and managed persistently (i.e., on disk).

Data warehouse systems use back-end tools and utilities to populate and refresh their data (Fig. 3.1). These tools and utilities include the functions of **data extraction** (gathering data from multiple, heterogeneous, and external sources), **data cleaning** (detecting errors in data and rectifying them when possible), **data transformation** (converting data from legacy or host format to warehouse format), **loading** (sorting, summarizing, consolidating, computing views, checking integrity, and building indices and partitions), and **refreshing** (propagating the updates from the data sources to the warehouse). In addition, data warehouse systems usually provide a good set of data warehouse management tools.

### ***ETL for data warehouses***

In order to load and periodically refresh contents in data warehouses, typically data warehousing systems implement some ETL modules. We discuss the essential techniques and methods for data extraction, transformation, and loading in Chapter 2, which serve data warehousing, too. Here, we briefly introduce some major tasks of ETL for data warehouses.

#### Data extraction

A data extraction process extracts data from external sources and is often the most important aspect of ETL. For example, a data warehouse may need to extract transaction data from an OLTP database and also user review data from social media repository. To encapsulate the details of various data sources, *wrappers* are often developed and deployed, which interact with data sources and supply extracted data to the ETL module. Due to the diversity and dynamics of data sources, manually developing wrappers is often inefficient and ineffective in quality. Recently, more and more wrappers are data driven and can automatically adapt to changes of data sources, such as changes in schema, update frequency, layout, and encoding. For example, a wrapper for an OLTP database can monitor and adapt to schema updates. A wrapper for a social media can crawl data from the social media and extract key fields from text, such as product name and sentiment of user review. Moreover, the wrapper may also be able to adapt to changes in social media layouts and remain robust against spamming.

### Data transformation

More often than not, data extracted from sources may not meet the requirements of a data warehouse immediately. There may be some gaps, such as mismatching in data format, enforcement of business integrity constraints, and requirements on data quality. Data transformation applies rules and functions to transform the extracted data, enforce business logic and improve quality, so that the transformed data is ready to be loaded into the data warehouse. For example, in the transformation step, data about addresses can be cleansed so that standard representation of addresses is used, and the correct information about country, state, city, and postcode is identified and encoded. Moreover, through transformation, we can enforce business logic, such as requiring that every transaction of amount over 1 million dollars has to be associated with a customer representative. Data cleansing and quality improvement are also important tasks in the data transformation stage.

Data transformation is a dynamic process. Data mining techniques are frequently used in data transformation. For example, data mining techniques can be used to detect data quality issues and improve data cleansing. Moreover, as business advances, business logic also evolves correspondingly. The data transformation process has to be updated accordingly.

### Data loading

After data extracted from sources and transformed, the loading phase loads data into data warehouses. Loading may take many different ways. For example, a relative small data warehouse may load data in a centralized and periodic way (e.g., loading on a daily, weekly, or monthly basis). A large data warehouse crossing many distributed servers may have to load data in a distributed manner. If a data warehouse supports a highly time-sensitive business, the data warehouse may have to load data in a more frequent or even real-time manner. Loading data is often time consuming and the slowest part of an ETL process. Loading may also affect the availability, usability, and bandwidth of a data warehouse. Various techniques are developed to achieve high performance in loading data into data warehouses and to minimize interferes to regular services provided by data warehouses.

### ***Enterprise data warehouse and data mart***

From the architecture point of view, there are two major data warehouse models, namely the *enterprise warehouse* and the *data mart*.

**Enterprise warehouse:** An enterprise warehouse collects all information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data and summarized data and can range in size from hundreds of gigabytes to terabytes or beyond. It requires extensive business modeling at the enterprise level and may take years to design and build.

**Data mart:** A data mart contains a subset of corporate-wide data that is of value to a specific group of users, such as those within a business department. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, marketing channel and sales. A risk control data mart may focus on customer credit, risk, and different types of frauds. The data contained in data marts tend to be summarized. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning are not enterprise-wide.

Depending on the source of data, data marts can be categorized as independent or dependent. *Independent* data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area. *Dependent* data marts are sourced directly from enterprise data warehouses. In practice, many data marks load data from both enterprise data warehouses and external or specific internal data sources.

Some circumstances also employ a *virtual warehouse*, which is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but put excess overhead on operational database servers.

*“It is often said that enterprises use artificial intelligence (AI for short) more and more in business and data is a foundation of AI. What is the relationship between data warehouses and AI?”* In general, data warehouses can support deployment of AI and machine learning functionalities. At the same time, artificial intelligence and machine learning tools can be used on top of data warehouses to take the best advantage of data warehouses.

Artificial intelligence refers to various computer systems that can conduct tasks that normally need human intelligence, such as playing board games, automatic driving, and dialog with humans. Machine learning, one of the core technologies in AI, is to build computer systems that can learn without being explicitly programmed the specific instructions. Many machine learning techniques are used by data mining, such as classification and clustering, which will be discussed in detail later in this book.

Artificial intelligence and machine learning tools need to consume considerable amounts of data to build various models for sophisticated tasks. Data warehouses organize and summarize data at proper levels and thus can support deployment of AI and machine learning functionalities. For example, an e-commerce company may want to build an AI model to categorize customers into different groups for better customer-relation management. This sophisticated task can be substantially benefited from a data mart of customer information, which can provide cleaned, integrated, and summarized data about customers.

At the same time, AI and machine learning techniques are widely used in various steps of data warehousing. For example, machine learning techniques can be used in constructing data warehouses, such as filling in missing values and identifying entities in data cleaning (see Chapter 2). Moreover, the output from AI models may be included in a data warehouse. For example, a data mart of customer information may likely include customer profiles, where customers and customer groups are often labeled based on their behavior, such as age groups, income levels, and consumption preferences. Those labels are often predicted by machine learning models trained from customer data. Third, AI and machine learning techniques can be used to optimize data warehouse performance. For example, machine learning techniques can be used to tune up the performance of data indexing and task execution in data warehouses distributed in large data centers and also can help to lower down power consumption substantially. Last but not least, AI and machine learning techniques are essential for knowledge workers to explore and understand data in data warehouses and make well-informed decisions. For example, an analyst can build machine learning models to explore the relationship between business growth rates in different regions and the marketing cost. More examples will be given in the later part of this book.

### 3.1.3 Data lakes

*“In some organizations, people mention ‘data lakes’. What are data lakes and what are the relations and differences between data lakes and data warehouses?”* In a big organization, there are often a massive number of complicated data sources with a wild variety in data types, formats, and quality, such as business data in relational databases, communication records between customers and the organization, regulations, market analysis, and external market information. Many data exploration analyses are one-time and may have to use data from different corners. It may take a long time to design and develop a data warehouse, where data is integrated, transformed, structured, and loaded according to defined usages. Moreover, many data-driven explorations have to be self-service business intelligence so that data scientists can analyze and explore data by themselves. To address the vast data usage demands in the organization, as an alternative, a data lake may be built.

Conceptually, a *data lake* is a single repository of all enterprise data in the natural format, such as relational data, semistructured data (e.g., XML, CSV, JSON), unstructured data (e.g., emails, PDF files), and even binary data (e.g., images, audio, video). More often than not, a data lake takes the form of object blocks or files and is hosted using a cloud-based or distributed data repository. A data lake often stores both raw data copies and transformed data. Many analytical tasks, such as reporting, visualization, analytics, and data mining, can be conducted on data lakes.

*“What are the essential differences between data warehouses and data lakes?”* First, to build a data warehouse, one has to analyze the data sources, understand the business processes, and develop the corresponding data models. The subjects in data warehouses reflect the factors in the corresponding business analysis and decision-making processes. In contrast, a data lake retains all data in an organization, including the current and historical data, as well as data being used now and that not used at this time. The rationale is that the data lake as the complete repository can be used as the base of all data-related tasks now and in the future.

Second, a data warehouse typically stores data extracted from transactional data, including quantitative metrics and attribute values and does not cover much nonrelational data, such as text, images, and video. Data are loaded to data warehouses according to predefined schemas. In contrast, a data lake natively embraces all data types. Data are transformed when it is used.

Third, a data warehouse is designed for data analysts and executives. The queries on a data warehouse are typically supporting decision making. In contrast, since a data lake includes all data in the natural form, it can support all users in an organization, including operational users, analysts, and executives.

Fourth, the well-designed structures in a data warehouse provide high-quality support to target analytical tasks. However, for new queries or business changes that are not covered by the data warehouse design, it takes time to upgrade the data warehouse to address the new demands, which is the major pain-point in data warehousing. In contrast, a data lake stores all data in the raw form and thus is always available for exploring any novel usages. Data scientists can directly work on data lakes to conduct data analysis. The analysis results may also become a part of the data lake.

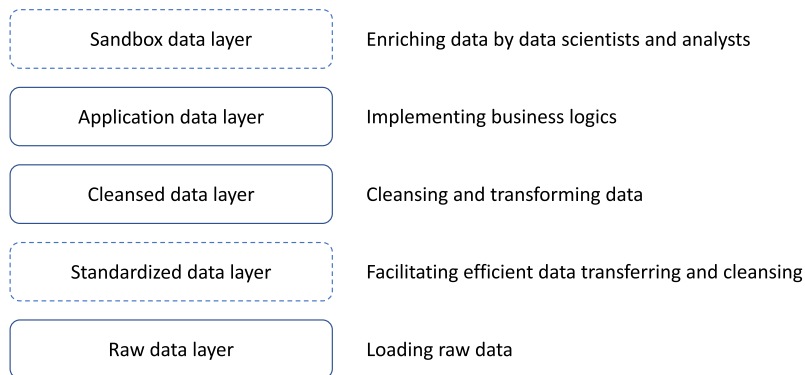
Last, since building a data warehouse takes time and resource, a data warehouse typically cannot cover all business and analytic users in an organization. For those business and users not supported by a data warehouse, they can still use a data lake to obtain faster insights.

Data warehouses and data lakes represent two views on data analytics. Data warehouses are more top-down, structured, and centralized. In contrast, data lakes are more bottom-up, quick prototyping, and democratic. In enterprise practice, a combination is often exercised to harvest the best gain.

“As a data lake has to store all enterprise data, which is often huge in size and diverse in type and format, how are data in a data lake stored and organized?” Typically data lakes have a core storage layer, which stores raw and/or lightly processed data. There are several important considerations in designing and implementing the data lake storage. First, since data lakes are served as the centralized data repository for an entire enterprise, the data storage has to be exceptionally scalable. Second, as data lakes have to respond to a wide variety of queries and analytic tasks, data robustness is critical. Consequently, the data storage layer has to have high durability. In other words, the data stored in a data lake should be intact and pristine all the time. Third, to address the diversity of data in enterprises, data lake storage has to support different types of data in various format, including structured data, semistructured data, and unstructured data. All such data has to be stored and managed consistently and harmonically in the same repository. Fourth, as data lakes are used to support different kinds of queries, analyses, and applications, the data storage should be able to support various data schemas, many of which may not be known or available when data lakes are designed. In other words, the data lake storage must be independent from any fixed schema. Last, in contrast to many applications where data and computation are corporate, the storage layer of data lakes should be decoupled with computation resources, so that various computation resources, ranging from legacy mainframe servers to clouds, can access data in data lakes. This separation can allow the maximum scalability in both data lakes and applications supported by data lakes.

Conceptually, a data lake has a storage layer as a single repository. In implementation, the data repository is still divided into multiple layers. Typically, the repository has three mandatory layers: raw data, cleansed data, and application data. Optionally, a standardized data layer and a sandbox layer may be added. Let us explain the layers bottom up. Fig. 3.2 summarizes the layers.

The raw data layer is the lowest layer and is also known as the ingestion layer or the landing area. At this layer, raw data is loaded in the native format. No data processing is conducted, such as cleansing, duplicate removing, or data transforming. Data are typically organized into folders by areas, data sources, objects, and time of ingestion. The data at this level are not ready for use yet, and thus end users of data lakes should not be allowed to access to the raw data layer.



**FIGURE 3.2**

The layers of data storage in data lakes.

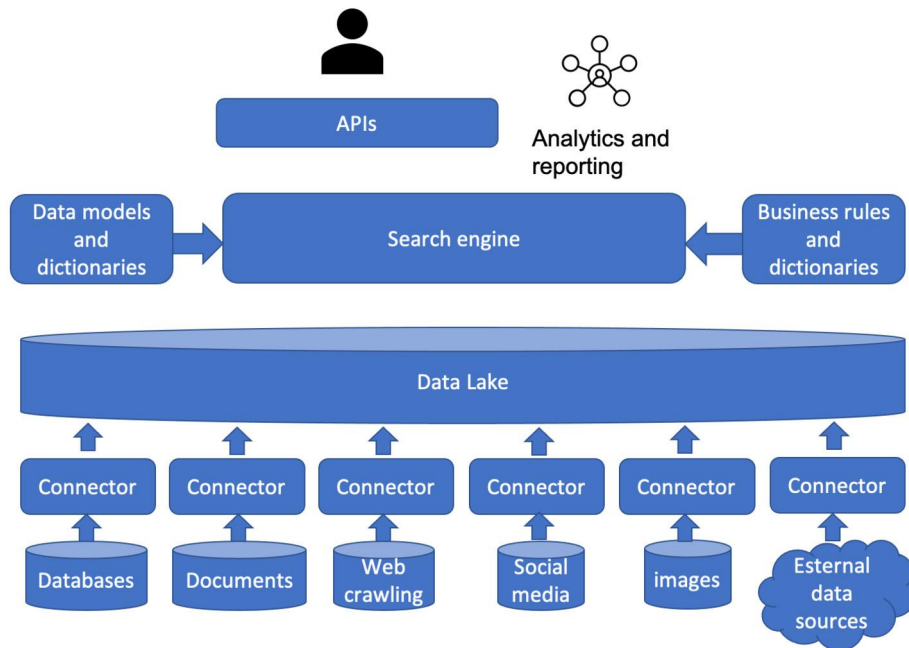
Optionally, a data lake may have a standardized data layer on top of the raw data layer. The main objective of the standardized data layer is to facilitate high performance in data transferring and cleansing. For example, in the raw data layer, data are stored in its native format. In the standardized data layer, data may be transformed into some formats that are best for cleansing. Moreover, data may be divided into structures of finer grain for more efficient access and processing.

The next layer is the cleansed data layer, also known as the curated layer or the conformed layer. At this layer, data are cleansed and transformed, such as being denormalized or consolidated. Moreover, data are organized into data sets and stored into tables or files. End users of data lakes are allowed to access data at this layer.

On top of the cleansed data layer is the application data layer, also known as the trusted layer, the secure layer, or the production layer. Business logics are implemented at this layer. Therefore, many applications, including those data mining and machine learning applications, can be built based on this layer.

In some organizations, data scientists and analysts may conduct experiments and find patterns and correlations. Their projects may enrich the data substantially and thus create new data. Such data may be stored at the optional sandbox data layer.

“Centered by the data lake storage, what is the architecture of data lakes? What are the other important components in data lakes in addition to data storage?” Fig. 3.3 shows the conceptual architecture of data lakes. A data lake takes data from a wide spectrum of data repositories in an enterprise or an



**FIGURE 3.3**

The conceptual architecture of data lakes.

organization, such as the databases, the store of documents, data crawled from the web, social media, images (e.g., products), and possibly external data sources. Data from those data sources are loaded into the data lake through connectors in a continuous manner. Once data are ingested into a data lake, the data goes through the layers that we just discussed.

Data lakes serve as the centralized data repositories of enterprises and organizations. End users, such as analysts and data scientists, can access data sets in data lakes, at the cleansed data layer and the upper layers. A major type of access is to discover the data sets that can be used to fulfill analytic tasks. These “data discovery” tasks are conducted through an enterprise search engine. For example, a data scientist designing a marketing campaign may want to find all data related to customers in industry section “electronics manufacturing.” Through the search engine, the data scientist may find data sets like purchase transactions from the operational databases, communication documents with those related customers, product categories of those customers crawled from the web, product reviews from social media, and product images and product availability data provided by those customers as external data. Clearly, without a data lake as a centralized data repository, the data scientist may have to spend a lot of time to find such data scattered in different departments of the enterprise and obtain access to those data sets. In order to facilitate better utility of data in data lakes, data models and dictionaries and business rules and dictionaries are employed as domain business knowledge bases for the enterprise search engine so that the search of data sets is business oriented instead of technical oriented. Last, many applications can be built on top of the data services provided by data lakes through the corresponding application programming interfaces (APIs). Regular analytics and reporting services can also be developed and maintained accordingly.

Data lakes as centralized data repositories in enterprises bring in huge efficiency and advantage in data-driven business operation and decision making. At the same time, data lakes also post grand challenges in management and administration. In addition to the data storage layer, data lakes also need to address a series of important aspects. Among others, security is a central piece. Access to data lakes should be properly defined and assigned to the right people for the right periods. Data stored in data lakes should be protected properly. Authentication, accountability, authorization, and data protection should be held consistently and comprehensively. In order to ensure security and tune for high performance, data lakes should be under systematic governance. For example, monitoring, logging, and lineage should be conducted regularly. Availability, usability, security, and integrity of data lakes should be monitored and managed all the time. In addition, data quality, data auditing, archives, and stewardship are some other important aspects in data lakes.

---

## 3.2 Data warehouse modeling: schema and measures

As discussed in the last section, a data warehouse integrates historical and current data in a subject-oriented and nonvolatile manner. The data models used in data warehouses organize data according to subjects. Here, a subject, such as customers, is captured by dimensions, such as gender, age group, and occupation, and measures, such as total purchase and average transaction amount. Naturally, data warehouses and OLAP tools are based on **multidimensional data models**, which view data in the form a *data cube*. In this section, you will learn how data cubes model *n*-dimensional data (Section 3.2.1). In Section 3.2.2, various multidimensional models are explained: star schema, snowflake schema, and fact constellation. Data in a data warehouse may be analyzed in different granularities, defined by concept

hierarchies. You will learn concept hierarchies in Section 3.2.3. You will also learn about different categories of measures and how they can be computed efficiently (Section 3.2.4).

### 3.2.1 Data cube: a multidimensional data model

“*What is a data cube?*” At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. A **data cube** allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

A multidimensional data model is typically organized around a central theme, also known as a subject, such as sales. The information about a subject can be divided into two parts in analysis. The first part is the perspectives that the subject is to be analyzed. For example, for subject sales in a company, the possible perspectives may include time, item, branch, and location. Those perspectives are modeled as **dimensions**. In the simplest multidimensional data model, a dimension table can be built for each dimension. For example, a dimension table for *item* may contain the attributes *item\_name*, *brand* and *type*.

The second part is the measurements on a subject. Those measurements are called **facts**. For example, for subject sales in a company, the facts may be *dollars\_sold* (sales amount in dollars), *units\_sold* (number of units sold), and *amount\_budgeted*. Facts are typically numerical, but still may take some other data types, such as categorical data or text.

In a data warehouse, a **fact table** stores the names of the *facts*, or measures, as well as (foreign) keys referencing to each of the related dimension tables.

In general, a data cube can have as many dimensions as the business needs and thus is *n*-dimensional. To elaborate data cubes and the multidimensional data model, let us start by looking at a simple 2-D data cube that is, in fact, a table or spreadsheet for sales data for a company. In particular, we will look at the sales data for items sold per quarter in a city, say Vancouver. The data are shown in Table 3.1. In this 2-D representation, the sales for Vancouver are shown with respect to the *time* dimension (organized in quarters) and the *item* dimension (organized according to the types of items sold). The fact or measure displayed is *dollars\_sold* (in thousands).

Now, suppose that we would like to view the sales data with a third dimension. For instance, suppose we would like to view the data according to *time* and *item*, as well as *location*, for the cities Chicago, New York, Toronto, and Vancouver. These 3-D data are shown in Table 3.2. The 3-D data in the table

**Table 3.1 2-D view of sales data according to *time* and *item*.**

location = “Vancouver”				
time (quarter)	item (type)			
	home entertainment	computer	phone	security
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

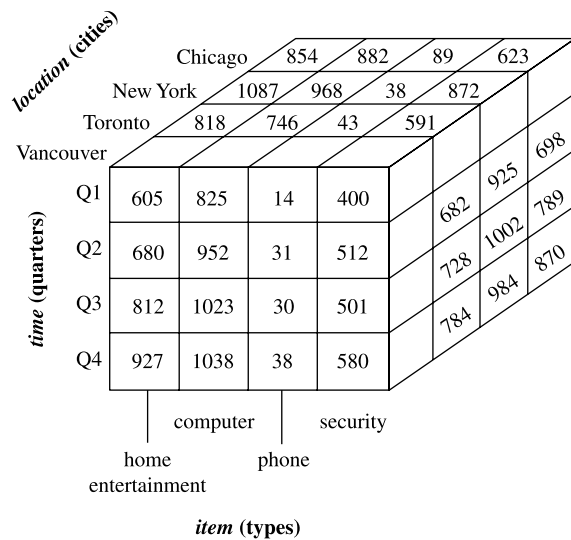
*Note: The sales are from branches located in the city of Vancouver. The measure displayed is dollars\_sold (in thousands).*



**Table 3.2 3-D view of sales data according to *time*, *item*, and *location*.**

time	location = "Chicago"				location = "New York"				location = "Toronto"				location = "Vancouver"			
	item				item				item				item			
	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

*Note: The measure displayed is dollars\_sold (in thousands).*



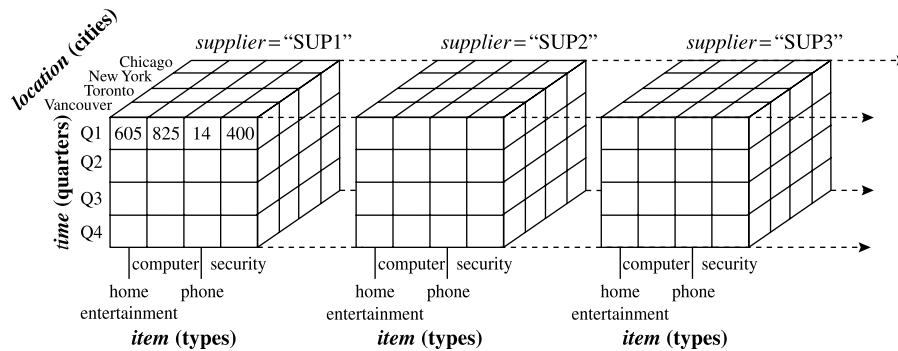
**FIGURE 3.4**

A 3-D data cube representation of the data in Table 3.2, according to *time*, *item*, and *location*. The measure displayed is *dollars\_sold* (in thousands).

are represented as a series of 2-D tables. Conceptually, we may also represent the same data in the form of a 3-D data cube, as in Fig. 3.4.

Suppose that we would now like to view our sales data with an additional fourth dimension, say *supplier*. Visualizing things in 4-D becomes tricky. However, we can think of a 4-D cube as a series of 3-D cubes, as shown in Fig. 3.5. If we continue in this way, we may display any *n*-dimensional data as a series of (*n* - 1)-dimensional “cubes.” The data cube is a metaphor for multidimensional data storage. The actual physical storage of such data may differ from its logical representation. The important thing to remember is that data cubes are *n*-dimensional and do not confine data to 3-D.

Tables 3.1 and 3.2 show the data at different degrees of summarization. In the data warehousing research literature, a data cube like those shown in Figs. 3.4 and 3.5 is often referred to as a **cuboid**.



**FIGURE 3.5**

A 4-D data cube representation of sales data, according to *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars\_sold* (in thousands). For improved readability, only some of the cube values are shown.

In SQL terms, these aggregations are referred to as *group-by*'s. Each *group-by* can be represented by a cuboid.

Given a set of dimensions, we can generate a cuboid for each of the possible subsets of the given dimensions, including the empty set. The result would form a *lattice* of cuboids, each showing the data at a different level of summarization, or *group-by*. The lattice of cuboids is then referred to as a data cube. Fig. 3.6 shows a lattice of cuboids forming a data cube for dimensions *time*, *item*, *location*, and *supplier*.

The cuboid that holds the lowest level of summarization is called the **base cuboid**. For example, the 4-D cuboid in Fig. 3.5 is the base cuboid for the given *time*, *item*, *location*, and *supplier* dimensions. Fig. 3.4 is a 3-D (nonbase) cuboid for *time*, *item*, and *location*, summarized for all suppliers. The 0-D cuboid, which holds the highest level of summarization, is called the **apex cuboid**. In our example, this is the total sales, or *dollars\_sold*, summarized over all four dimensions. The apex cuboid is typically denoted by `all`.

### 3.2.2 Schemas for multidimensional data models: stars, snowflakes, and fact constellations

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships among them. Normalization is conducted to break a wide table into narrower tables so that many transactional operations only have to access very few records in one or a small number of tables, and thus concurrency of transactional operations can be maximized. Such a data model is appropriate for online transaction processing. An online data analysis often has to scan a lot of data. To support online data analysis, a data warehouse requires a concise, subject-oriented schema that facilitates scanning a large amount of data efficiently.

The most popular data model for a data warehouse is a **multidimensional model**. The most common paradigm of multidimensional model is **star schema**, in which a data warehouse contains (1) a large central table (**fact table**) containing the bulk of the data, with no redundancy, and (2) a set of smaller

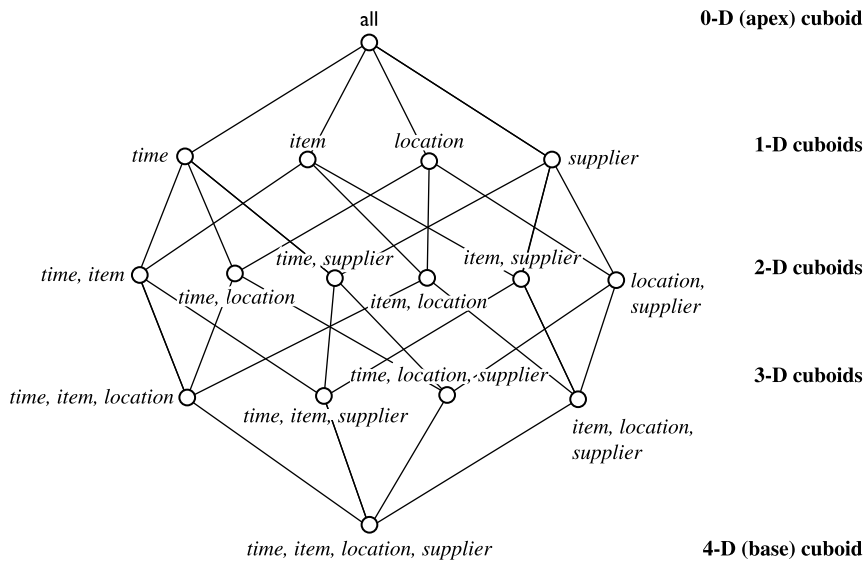


FIGURE 3.6

Lattice of cuboids, making up a 4-D data cube for *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

attendant tables (**dimension tables**), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

**Example 3.1. Star schema.** A star schema for sales is shown in Fig. 3.7. Sales are considered along four dimensions: *time*, *item*, *branch*, and *location*. The schema contains a central fact table for *sales* that contains the keys to each of the four dimensions, along with two measures: *dollars\_sold* and *units\_sold*. To minimize the size of the fact table, dimension identifiers (e.g., *time\_key* and *item\_key*) are system-generated identifiers. □

Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the *location* dimension table contains the attribute set  $\{location\_key, street, city, province\_or\_state, country\}$ . This constraint may introduce some redundancy. For example, “Urbana” and “Chicago” are both cities in the state of Illinois, USA. Entries for such cities in the *location* dimension table create redundancy among the attributes *province\_or\_state* and *country*, that is, (... , Urbana, IL, USA) and (... , Chicago, IL, USA).

**Snowflake schema** is a variant of star schema, where some dimension tables are *normalized*, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake.

The major difference between the snowflake schema and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies. Such a table is easy to maintain and saves storage space. However, this space savings is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure may reduce the effectiveness

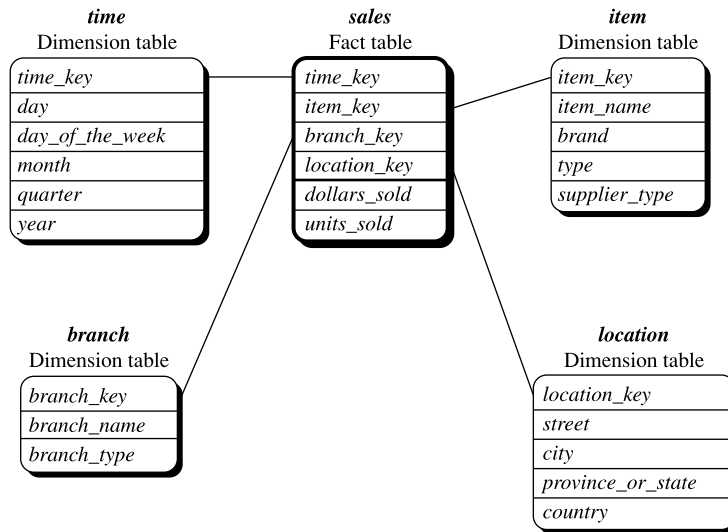


FIGURE 3.7

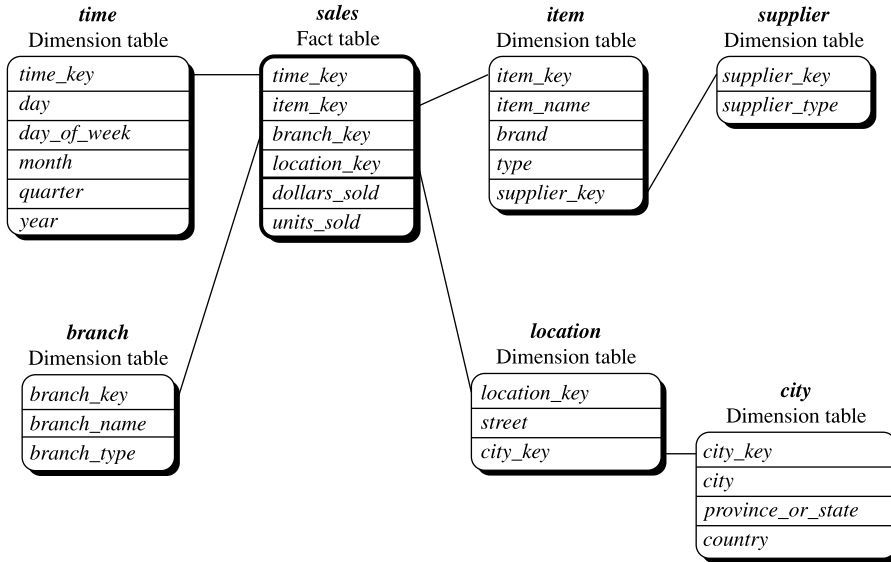
Star schema of *sales* data warehouse.

of browsing, since more joins are needed to execute a query. Consequently, the system performance may be adversely impacted. Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.

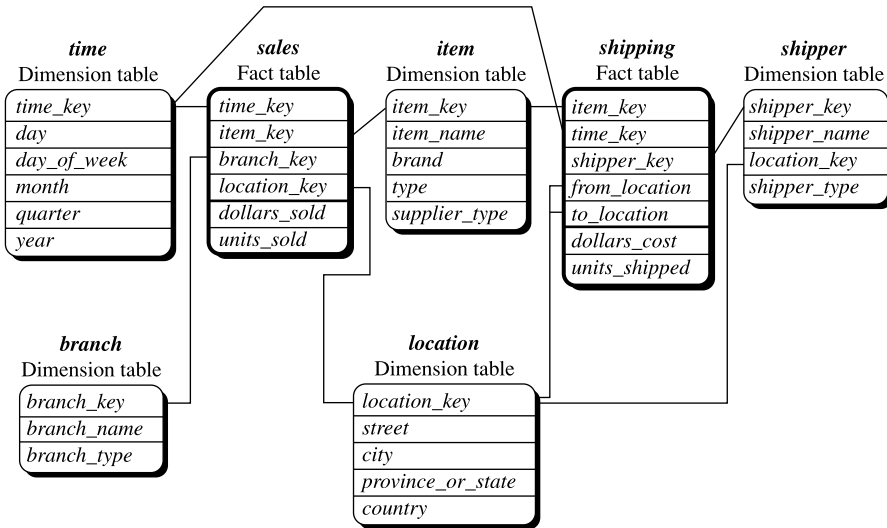
**Example 3.2. Snowflake schema.** A snowflake schema for sales is given in Fig. 3.8. Here, the *sales* fact table is identical to that of the star schema in Fig. 3.7. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for *item* in the star schema is normalized in the snowflake schema, resulting in the new *item* and *supplier* tables. For example, the *item* dimension table now contains attributes *item\_key*, *item\_name*, *brand*, *type*, and *supplier\_key*, where *supplier\_key* is linked to the *supplier* dimension table, containing *supplier\_key* and *supplier\_type* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two new tables: *location* and *city*. The *city\_key* in the new *location* table links to the *city* dimension. Notice that, when desirable, further normalization can be performed on *province\_or\_state* and *country* in the snowflake schema shown in Fig. 3.8. □

Sophisticated applications may require multiple fact tables to share dimension tables. This kind of schema can be viewed as a collection of stars and hence is called a **galaxy schema** or a **fact constellation**.

**Example 3.3. Fact constellation.** A fact constellation schema is shown in Fig. 3.9. This schema specifies two fact tables, *sales* and *shipping*. The *sales* table definition is identical to that of the star schema (Fig. 3.7). The *shipping* table has five dimensions, or keys—*item\_key*, *time\_key*, *shipper\_key*, *from\_location*, and *to\_location*—and two measures—*dollars\_cost* and *units\_shipped*. A fact constel-



**FIGURE 3.8**  
Snowflake schema of a sales data warehouse.



**FIGURE 3.9**  
Fact constellation schema of a sales and shipping data warehouse.

lation schema allows dimension tables to be shared between fact tables. For example, the dimension tables for *time*, *item*, and *location* are shared between the *sales* and *shipping* fact tables. □

### 3.2.3 Concept hierarchies

Dimensions define concept hierarchies. A **concept hierarchy** defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Consider a concept hierarchy for the dimension *location*. City values for *location* include Vancouver, Toronto, New York, and Chicago. Each city, however, can be mapped to the province or state to which it belongs. For example, Vancouver can be mapped to British Columbia and Chicago to Illinois. The provinces and states can in turn be mapped to the country (e.g., Canada or the United States) to which they belong. These mappings form a concept hierarchy for the dimension *location*, mapping a set of low-level concepts (i.e., cities) to higher-level, more general concepts (i.e., countries). This concept hierarchy is illustrated in Fig. 3.10.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension *location* is described by the attributes *number*, *street*, *city*, *province\_or\_state*, *zip\_code*, and *country*. These attributes are related by a total order, forming a concept hierarchy such as “*street* < *city* < *province\_or\_state* < *country*.” This hierarchy is shown in Fig. 3.11(a). Alternatively, the attributes of a dimension may be organized in a partial order, forming an acyclic directed graph. An example of a partial order for the *time* dimension based on the attributes *day*, *week*, *month*, *quarter*, and *year* is “*day* < {*month* < *quarter*; *week*} < *year*.”<sup>1</sup> This partial order structure is shown in Fig. 3.11(b).

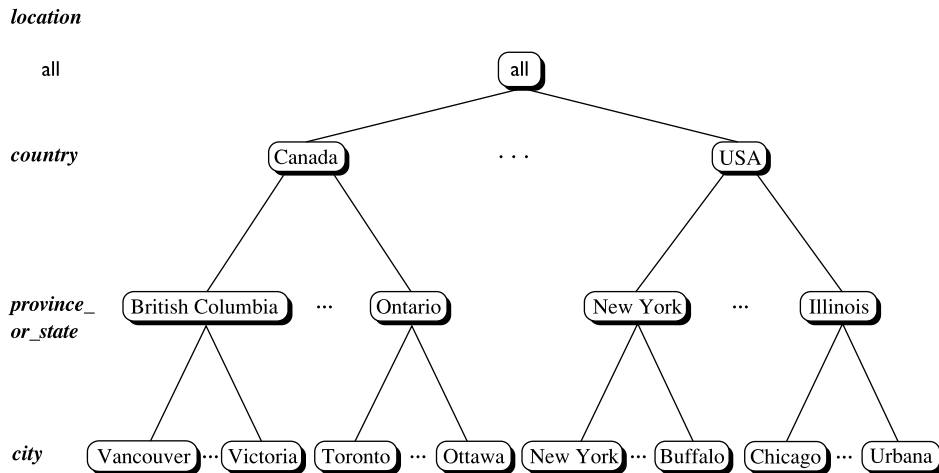


FIGURE 3.10

A concept hierarchy for *location*. Due to space limitations, not all of the hierarchy nodes are shown, indicated by ellipses between nodes.

<sup>1</sup> Since a *week* often crosses the boundary of two consecutive months, it is usually not treated as a lower abstraction of *month*. Instead, it is often treated as a lower abstraction of *year*, since a year contains approximately 52 weeks.

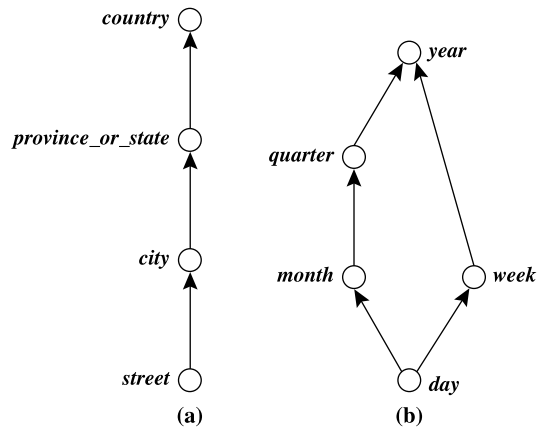


FIGURE 3.11

Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location* and (b) a lattice for *time*.

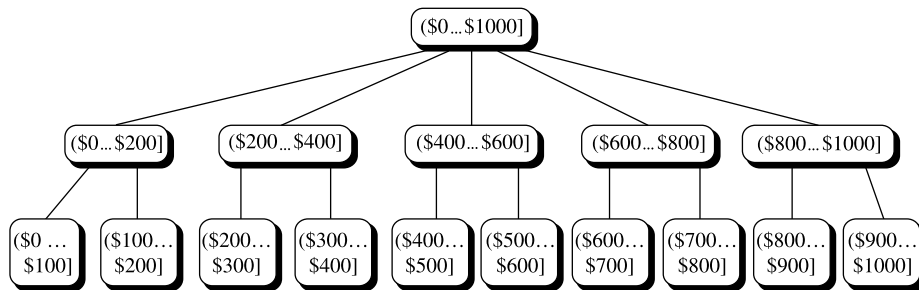


FIGURE 3.12

A concept hierarchy for *price*.

A concept hierarchy that is a total or partial order among attributes in a database schema is called a **schema hierarchy**. Concept hierarchies that are common to many applications (e.g., *for time*) may be predefined in the data mining system. Data mining systems should provide users with the flexibility to tailor predefined hierarchies according to their particular needs. For example, users may want to define a fiscal year starting on April 1 or an academic year starting on September 1.

Concept hierarchies may also be defined by discretizing or grouping values for a given dimension or attribute, resulting in a **set-grouping hierarchy**. A total or partial order can be defined among groups of values. An example of a set-grouping hierarchy is shown in Fig. 3.12 for the dimension *price*, where an interval  $(\$X \dots \$Y]$  denotes the range from  $\$X$  (exclusive) to  $\$Y$  (inclusive).

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. For instance, a user may prefer to organize *price* by defining ranges for *inexpensive*, *moderately\_priced*, and *expensive*.

Concept hierarchies may be provided manually by system users, domain experts or knowledge engineers, or may be automatically generated based on statistical analysis of the data distribution. Concept hierarchies allow data to be handled at varying levels of abstraction, as we will see in Section 3.2.4.

### 3.2.4 Measures: categorization and computation

“*How are measures computed?*” To answer this question, we first study how measures can be categorized. Note that a *multidimensional point* in the data cube space, also known as a *cell* in the data cube, can be defined by a set of dimension–value pairs; for example,  $\langle \text{time} = \text{“Q1,” location} = \text{“Vancouver,” item} = \text{“computer”} \rangle$ . A **measure** in a data cube is a numeric function that can be evaluated at each point in the data cube space. A measure value is computed for a given point by aggregating the data corresponding to the respective dimension–value pairs defining the given point. For example, the measure *total-sales* for the cell  $\langle \text{time} = \text{“Q1,” location} = \text{“Vancouver,” item} = \text{“computer”} \rangle$  is computed by summing up all the amounts happened in Q1, at the branch of Vancouver, and about computers from the fact table.

Measures can be organized into three categories—distributive, algebraic, and holistic—based on the kind of aggregate functions used.

**Distributive:** An aggregate function is *distributive* if it can be computed in a distributed manner as follows. Suppose the data is partitioned into  $n$  sets arbitrarily. We apply the aggregate function to each partition, resulting in  $n$  aggregate values. If the result derived by applying the function to the  $n$  aggregate values is the same as that derived by applying the function to the entire data set (i.e., without partitioning), the function is said to be computed in a distributed manner.

For example, `sum()` can be computed for a data cube by first partitioning the cube into a set of subcubes, computing `sum()` for each subcube, and then summing up the counts obtained for each subcube. Hence `sum()` is a distributive aggregate function. For the same reason, `count()`, `min()`, and `max()` are distributive aggregate functions. By treating the count value of each nonempty base cell as 1 by default, `count()` of any cell in a cube can be viewed as the sum of the count values of all of its corresponding child cells in its subcube. Thus `count()` is distributive. A measure is *distributive* if it is obtained by applying a distributive aggregate function. Distributive measures can be computed efficiently because of the way the computation can be partitioned.

**Algebraic:** An aggregate function is *algebraic* if it can be computed by an algebraic function with  $M$  arguments (where  $M$  is a fixed positive integer), each of which is obtained by applying a distributive aggregate function. For example, `avg()` (average) can be computed by `sum()/count()` with two arguments, where both `sum()` and `count()` are distributive aggregate functions. Similarly, it can be shown that `min_N()` and `max_N()` (which find the  $N$  minimum and  $N$  maximum values, respectively, in a given set) and `standard_deviation()` are algebraic aggregate functions. A measure is *algebraic* if it is obtained by applying an algebraic aggregate function.

**Holistic:** An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function with  $M$  arguments (where  $M$  is a constant) that characterizes the computation. Some examples of holistic functions



include `median()`, `mode()`, and `rank()`. A measure is *holistic* if it is obtained by applying a holistic aggregate function.

Most large data cube applications require efficient and scalable computation, and thus distributive and algebraic measures are often used. Many efficient techniques for computing data cubes using distributive and algebraic measures exist. We will introduce some principled methods later in this chapter. In contrast, it is difficult to compute holistic measures efficiently. Efficient techniques to *approximate* the computation of some holistic measures, however, do exist. In many cases, such techniques are sufficient to overcome the difficulties of efficient computation of holistic measures.

---

### 3.3 OLAP operations

A data warehouse needs to support online multidimensional analytic queries. In this section, you will learn a series of typical OLAP operations on data warehouses (Section 3.3.1) and how to index data to support some OLAP queries (Section 3.3.2). An important problem is how data can be stored properly to support OLAP operations, which will be explained in Section 3.3.3.

#### 3.3.1 Typical OLAP operations

“How can multidimensional OLAP operations be used in data analysis?” In a multidimensional model, data are organized into multiple dimensions, and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides users with the flexibility to view data from different perspectives. A number of OLAP data cube operations empower interactive querying and analysis of the data at hand. Hence, OLAP provides a user-friendly environment for interactive data analysis.

**Example 3.4. OLAP operations.** Let us look at some typical OLAP operations for multidimensional data. Each of the following operations is illustrated in Fig. 3.13. At the center of the figure is a data cube for sales in a company. The cube contains three dimensions, *location*, *time*, and *item*, where *location* is aggregated with respect to city values, *time* is aggregated with respect to quarters, and *item* is aggregated with respect to item types. To aid in our explanation, we refer to this cube as the central cube. The measure displayed is *dollars\_sold* (in thousands). (For the sake of readability, only some cell values in the cubes are shown.) The data examined are for the cities Chicago, New York, Toronto, and Vancouver.

**Roll-up:** The roll-up operation (also called the *drill-up* operation by some vendors) performs aggregation on a data cube, either by *climbing up a concept hierarchy* for a dimension or by *dimension reduction*. Fig. 3.13 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for *location* given in Fig. 3.10. This hierarchy was defined as the total order “*street* < *city* < *province\_or\_state* < *country*.” The roll-up operation shown aggregates the data by ascending the *location* hierarchy from the level of *city* to the level of *country*. In other words, rather than grouping the data by city, the resulting cube groups the data by country.

When roll-up is performed by dimension reduction, one or more dimensions are removed from the given cube. For example, consider a sales data cube containing only the *location* and *time*

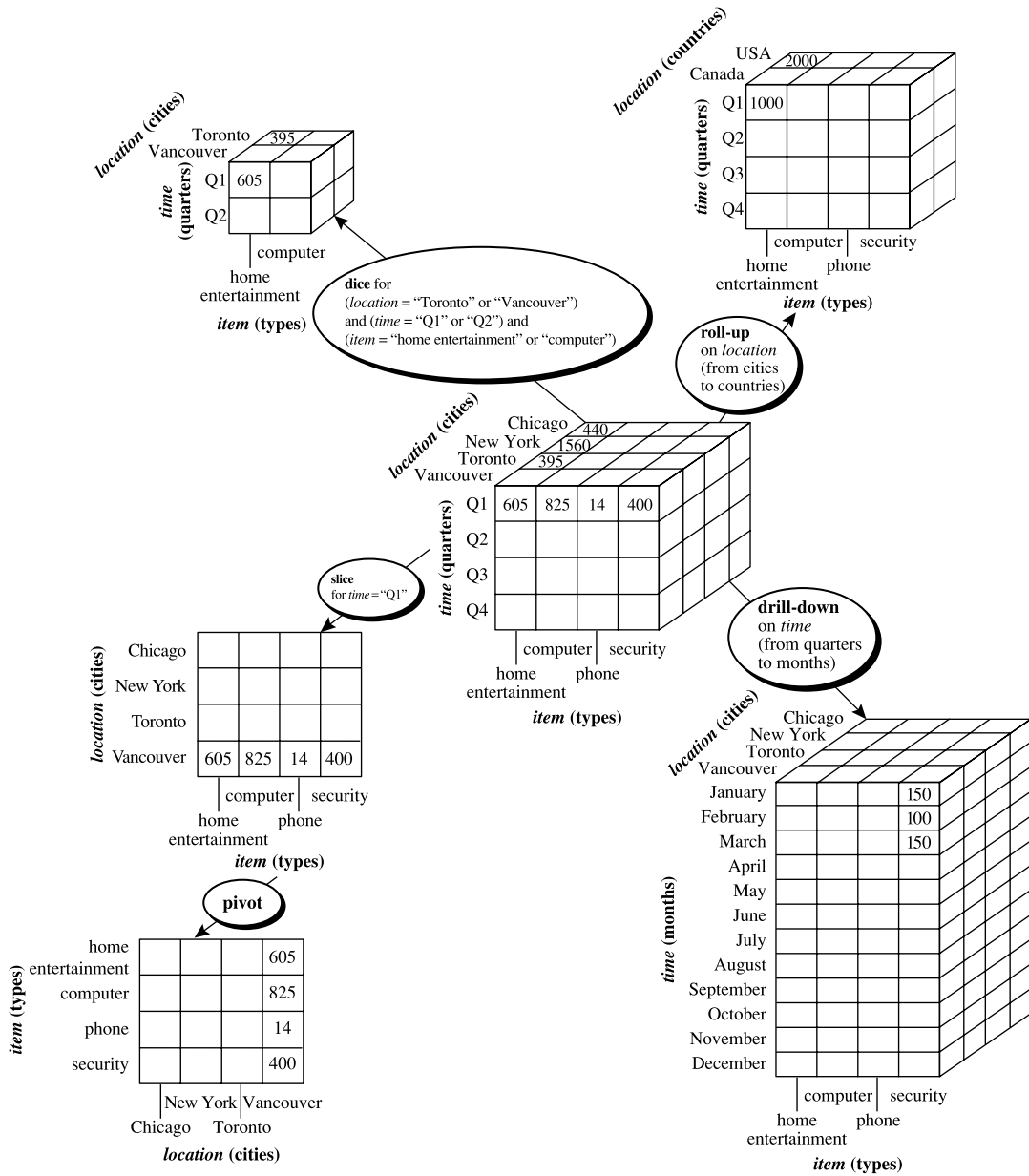


FIGURE 3.13

Examples of typical OLAP operations on multidimensional data.

dimensions. Roll-up may be performed by removing, say, the *location* dimension, resulting in an aggregation of the total sales by time of the whole company, rather than by location and by time.

**Drill-down:** Drill-down is the reverse of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either *stepping down a concept hierarchy* for a dimension or *introducing additional dimensions*. Fig. 3.13 shows the result of a drill-down operation performed on the central cube by stepping down a concept hierarchy for *time* defined as “*day < month < quarter < year*.” Drill-down occurs by descending the *time* hierarchy from the level of *quarter* to the more detailed level of *month*. The resulting data cube details the total sales per month rather than summarizing them by quarter.

Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube. For example, a drill-down on the central cube in Fig. 3.13 can be conducted by introducing an additional dimension, such as *customer\_group*.

**Slice and dice:** The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Fig. 3.13 shows a slice operation where the sales data is selected from the central cube for the dimension *time* using the criterion *time* = “Q1.” The *dice* operation defines a subcube by performing a selection on two or more dimensions. Fig. 3.13 shows a dice operation on the central cube based on the following selection criterion that involves three dimensions: (*location* = “Toronto” or “Vancouver”) and (*time* = “Q1” or “Q2”) and (*item* = “home entertainment” or “computer”).

**Pivot (rotate):** *Pivot* (also called *rotate*) is a visualization operation that rotates the data axes in view to provide an alternative data presentation. Fig. 3.13 shows a pivot operation where the *item* and *location* axes in a 2-D slice are rotated. Other examples include rotating the axes in a 3-D cube or transforming a 3-D cube into a series of 2-D planes.

**Other OLAP operations:** Some OLAP systems offer additional drilling operations. For example, **drill-across** executes queries involving (i.e., across) more than one fact table. The **drill-through** operation uses relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables.

Other OLAP operations may include ranking the top *N* or bottom *N* items in lists, as well as computing moving averages, growth rates, interests, internal return rates, depreciation, currency conversions, and statistical functions. □

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, and so on, and for computing measures across multiple dimensions. It can generate summarizations, aggregations, and hierarchies at each granularity level and at every dimension intersection. OLAP also supports functional models for forecasting, trend analysis, and statistical analysis. In this context, an OLAP engine is a powerful data analysis tool.

### 3.3.2 Indexing OLAP data: bitmap index and join index

To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views (using cuboids). We will discuss the general methods to select cuboids for materialization in Section 3.4. In this subsection, we examine how to index OLAP data by *bitmap indexing* and *join indexing*.

### Bitmap indexing

The **bitmap indexing** method is popular in OLAP products because it allows quick searching in data cubes. A bitmap index is an alternative representation of the *record\_ID (RID)* list. In the bitmap index for a given attribute, there is a distinct bit vector,  $B_v$ , for each value  $v$  in the attribute's domain. If the domain of a given attribute consists of  $n$  values, then  $n$  bits are needed for each entry in the bitmap index (i.e., there are  $n$  bit vectors). If the attribute has the value  $v$  for a given row in the data table, then the bit representing that value is set to 1 in the corresponding row of the bitmap index. All other bits for that row are set to 0.

**Example 3.5. Bitmap indexing.** Consider a customer information table shown in Fig. 3.14, where there is an attribute *gender*. To keep our discussion simple, assume there are two possible values on attribute *gender*. We may use one character, that is, 8 bits, for each record to represent the *gender* value, such as *F* for female and *M* for male. Bitmap index represents the *gender* value using one bit, such as 0 for female and 1 for male. This representation immediately brings in an eight-fold saving in storage.

More importantly, bitmap index can speed up many aggregate queries. For example, let us count the number of female customers in the customer information table. A straightforward method has to scan each record and count. For a table having 10,000 records and each record taking 100 bytes, the total I/O cost is  $10,000 \times 100 = 1,000,000$  bytes.

A bitmap index uses only 1 bit for each record. Those bits are packed into words in storage. For example, for the first 8 records in the table, the bitmap index values are packed into a byte 01010011. Scanning the whole bitmap index takes only 10,000 bits in I/O, that is 1250 bytes, 800 times less than scanning the whole table.

To calculate the number of 0s in a byte, we can simply use a precomputed hash table that uses the byte values as the index and stores the corresponding numbers of 0s. For example, the hash table stores value 4 in the 83rd entry, since 83 is the decimal value of binary 01010011 and the binary string has 4 0s. Using the byte 01010011, which is 53 in decimal, to search the hash table, we immediately know that there are 4 female customers in the first 8 records. We can compute the number of 0s in the whole

Customer information				Bitmap index	A table counting 0s in bytes	
<b>Name</b>	...	<b>Gender</b>	...	<b>Gender</b>	<b>Byte</b>	<b>Number of 0s</b>
Ada	...	Female	...	0	00000000	8
Bob	...	Male	...	1	00000001	7
Cathy	...	Female	...	0	00000010	7
Dan	...	Male	...	1	00000011	6
Elsa	...	Female	...	0	...	...
Flora	...	Female	...	0	01010011	4
George	...	Male	...	1	...	...
Hogan	...	Male	...	1		
...	...	...	...	...		

**FIGURE 3.14**

Indexing OLAP data using bitmap indices.

A fact table			The bit-sliced index											
...	...	<b>Amount</b>												
...	...	15.21												
...	...	27.06												
...	...	...												

Weights											
2 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
0	1	0	1	1	1	1	1	0	0	0	1
1	0	1	0	1	0	0	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...

**FIGURE 3.15**

Indexing OLAP data using bitmap indices.

gender attribute byte by byte using the bitmap index, and sum up the byte-wise counts to derive the total number of female customers. In practice, one can use machine words instead of bytes to further speed up the counting process.  $\square$

Bitmap indexing is advantageous compared to hash and tree indices in answering some types of OLAP queries. It is especially useful for low-cardinality domains because comparison, join, and aggregation operations are then reduced to bit arithmetic, which substantially reduces the processing time. Bitmap indexing leads to significant reductions in space and input/output (I/O) since a string of characters can be represented by a single bit.

Bitmap indexing can be extended to bit-sliced indexing for numeric data. Let us illustrate the ideas using an example.

**Example 3.6. Bit-sliced indexing.** Suppose we want to compute the sum of the amount attribute in the fact table in Fig. 3.15. We can write an amount into an integer number of pennies and then represent it as a binary number of  $n$  bits. If we represent an amount using 32 bits, that is, 4 bytes, it is good for amounts up to \$42,949,672.96 and sufficient for many application scenarios.

After we represent all amount numbers in binary, we can build a bitmap index for every bit. To compute the sum of all amounts, we count for each bit the number of 1s. Denote by  $x_i$  ( $i \geq 0$ ) the number of 1s in the  $i$ th bits of the amounts from right to left, the rightmost being bit 0. Since a 1 at the  $i$ th bit carries a weight of  $2^i$  pennies, the  $x_i$  1s in the  $i$ th bits of all amounts represent  $x_i \cdot 2^i$  pennies in the sum of the amounts. Therefore, the sum of amounts is  $\sum_{i \geq 0} x_i \cdot 2^i$  pennies or  $\frac{\sum_{i \geq 0} x_i \cdot 2^i}{100}$  dollars.  $\square$

### Join indexing

In a data warehousing schema such as the star-schema, we often need to join the fact table and the dimension tables. Joining tables again and again for various queries is definitely costly. Therefore, **join indexing** is used to precompute and store the identifier pairs of the join results so that the join results can be accessed efficiently.

**Example 3.7. Join indexing.** In Example 3.1, we defined a star schema of the form “*sales\_star* [*time*, *item*, *branch*, *location*]: *dollars\_sold* = *sum* (*sales\_in\_dollars*).” An example of a join index between the *sales* fact table and the *locations* and *items* dimension tables is shown in Fig. 3.16. Consider the OLAP query “the total sales of smartphone and desktop in BC.” If no index presents, then we have to

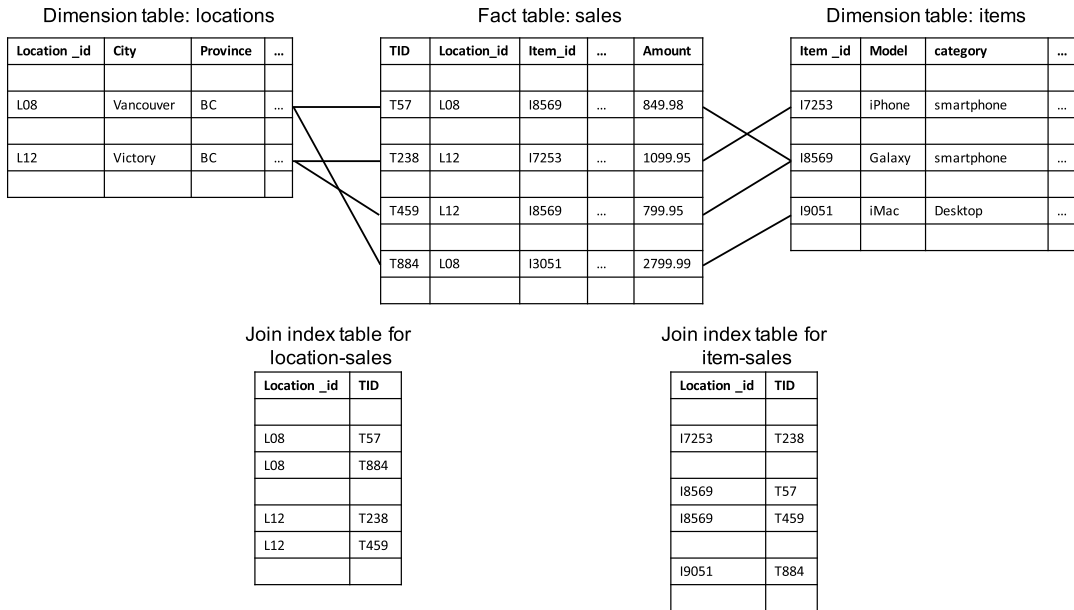


FIGURE 3.16

Join index.

join the fact table and the dimension tables *locations* and *items* and select only those join results about “smartphone” and “desktop.”

A join index table records the primary keys of the matching tuples in two tables. For example, in the join index table for location-sales, the pairs of *location\_id* and *TID* of the matching tuples in the dimension table *locations* and *sales* are recorded. From the join index table, we can quickly find out the TIDs of the tuples in the *sales* fact table belonging to “BC.” Similarly, using the join index for item-sales, we can identify the tuples in the sales table about “smartphone” and “desktop.” Using the identified TIDs as such, we can accurately access the tuples in the fact table that are needed to compute the OLAP aggregate and reduce the I/O cost. Typically, a data warehouse only contains a very small percentage of transactions about a selected area and product categories. For example, there may be only 0.1% of the transactions in the fact table that are smartphones and desktops sold in BC. Without using any index, we have to read the whole fact table into main memory in order to compute the aggregate. Using the join indexes, even each page contains 100 transaction records in the fact table, and all those transactions of smartphones and desktops sold in BC are evenly distributed, we only need to read 10% of the pages into main memory and thus save 90% of I/O. □

### 3.3.3 Storage implementation: column-based databases

“How to store data so that OLAP queries can be answered efficiently?” In many applications, a fact table may be wide and contain tens or even hundreds of attributes. More often than not, an OLAP

query may compute the aggregate of all records or a large portion of records on a small number of attributes. If the data is stored in a traditional relational table where records are stored row by row, then we have to scan all records in order to answer a query, but only a small segment in a record is used. This observation presents a significant opportunity to develop more efficient storage scheme for OLAP data.

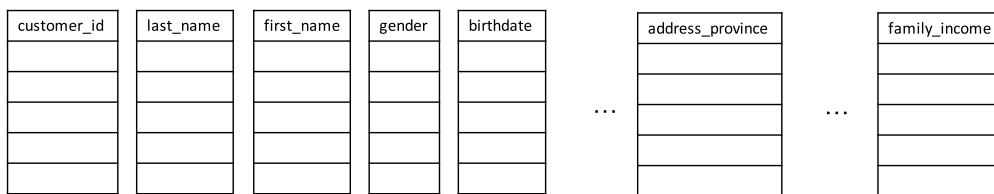
To make the storage more efficient for answering OLAP queries, a column-based database stores a wide table that is often used for aggregate queries in a column by column style. Specifically, a column-based database stores the values of all records on a column in consecutive storage blocks. All records are listed in the same order across all columns.

**Example 3.8. Column-based database.** Consider a fact table about customer information, which includes attributes and storage space in number of bytes `customer_id` (2), `last_name` (20), `first_name` (20), `gender` (1), `birthdate` (2), `address_street` (50), `address_city` (2), `address_province` (1), `address_country` (1), `email` (30), `registration_date` (2), and `family_income` (2). Each record occupies 133 bytes. If the fact table contains 10 million customer records, then the total space is over 1 GB.

If the data are stored row by row and we want to answer the OLAP query of the average family income of female customers by province, then we have to scan the whole table, reading all records. The I/O cost is 1 GB. At the same time, for each record, we only need to use 4 bytes among the 133 bytes, that is, attributes `gender`, `address_province`, and `family_income`. In other words, only  $\frac{4}{133} = 3\%$  of the data read are useful to answer the query.

A column-based database stores the data attribute by attribute in column, as shown in Fig. 3.17. To answer the above query, a column-based database only needs to read three columns, `gender`, `address_province`, and `family_income`. It checks the values on `gender` and increments the total and count for `address_province` accordingly. Overall, the total amount of I/O incurred to a column-based database in this case is  $4 \times 10$  million = 40 MB. A huge saving in I/O is achieved.

In implementation, preferably a column-based database processes a column at a time and uses bitmaps to keep the intermediate results so that they can be passed to the next column. In this example, we can first process the column `gender` and use a bitmap to keep the list of female customers. That is, each customer is associated with a bit, female being 0 and male being 1. Next, we can process the column `address_province`, and form a bitmap for each province. If a customer lives in BC, for example, then the associated bit in the bitmap of BC is set to 1, otherwise, it is set to 0. Last, to calculate the average family income of customers in BC, we only need to conduct the bitwise AND operation between the bitmap for `gender` and the bitmap for province BC. The resulting bitmap is used to select the entries in column `family_income` to calculate the average. □



**FIGURE 3.17**

Column-based storage.

Column-based databases have been extensively implemented in industry data warehousing and OLAP databases. Column-based databases have remarkable advantages for OLAP-like workloads, such as those aggregate queries searching a few columns of all records in a wide table. At the same time, column-based databases have to separate transactions into columns and compressed transactions as they are stored, which make column-based databases costly for OLTP workloads.

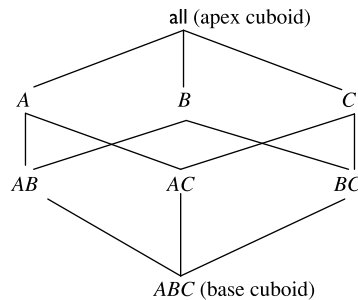
## 3.4 Data cube computation

Data warehouses contain huge volumes of data. OLAP servers demand that decision support queries be answered in the order of seconds. Data cubes are the core of data warehouses. Therefore, it is crucial for data warehouse systems to support highly efficient data cube computation, access, and query processing. In this section, we present an overview of the ideas behind data cube computation. Section 3.4.1 introduces the basic terminology. Section 3.4.2 discusses various ideas in fully or partially materializing a data cube. Section 3.4.3 explains how data cubes may be stored using various architectures. Section 3.4.4 overviews the general strategies frequently used in data cube computation. The detailed algorithms for data cube computation will be introduced in Section 3.5.

### 3.4.1 Terminology of data cube computation

One approach to cube computation is to compute aggregates over all subsets of the dimensions specified by a user. This can require excessive storage space, especially for large numbers of dimensions. To discuss the details about data cube computation and analysis, we need some terminology.

Fig. 3.18 shows a 3-D data cube for three dimensions,  $A$ ,  $B$ , and  $C$ , and an aggregate measure,  $M$ . Hereafter in this chapter, we always use the term *data cube* to refer to a lattice of cuboids rather than an individual cuboid. A tuple in a cuboid is also called a **cell**, which represents a point in the data cube space. A cell in the base cuboid is a **base cell**. A cell from a nonbase cuboid is an **aggregate cell**. An aggregate cell aggregates over one or more dimensions, where each aggregated dimension is indicated by a  $*$  in the cell notation. Suppose we have an  $n$ -dimensional data cube. Let  $a = (a_1, a_2, \dots, a_n, \text{measures})$



**FIGURE 3.18**

Lattice of cuboids making up a 3-D data cube with the dimensions  $A$ ,  $B$ , and  $C$  for some aggregate measure,  $M$ .



be a cell from one of the cuboids making up the data cube. We say that  $a$  is an  **$m$ -dimensional cell** (i.e., from an  $m$ -dimensional cuboid) if exactly  $m$  ( $m \leq n$ ) values among  $\{a_1, a_2, \dots, a_n\}$  are *not* \*. If  $m = n$ , then  $a$  is a base cell; otherwise, it is an aggregate cell (i.e., where  $m < n$ ).

**Example 3.9. Base and aggregate cells.** Consider a data cube with three dimensions, *month*, *city*, and *customer\_group*, and the measure *sales*. (*Jan*, \*, \*, 2800) and (\*, *Chicago*, \*, 1200) are 1-D cells; (*Jan*, \*, *Business*, 150) is a 2-D cell; and (*Jan*, *Chicago*, *Business*, 45) is a 3-D cell. Here, since the data cube has 3 dimensions, all base cells are 3-D, whereas 1-D and 2-D cells are aggregate cells.

(*month*, *city*, \*) is a 2-D cuboid, which contains all 2-D cells having non-\* values on attributes *month* and *city*. The base cuboid (*month*, *city*, *customer\_group*) contains all base cells. The apex cuboid ALL contains only one 0-D cell (\*, \*, \*). □

An ancestor–descendant relationship may exist between cells. In an  $n$ -dimensional data cube, an  $i$ -D cell  $a = (a_1, a_2, \dots, a_n, measures_a)$  is an **ancestor** of a  $j$ -D cell  $b = (b_1, b_2, \dots, b_n, measures_b)$ , and  $b$  is a **descendant** of  $a$ , if and only if (1)  $i < j$ , and (2) for  $1 \leq k \leq n$ ,  $a_k = b_k$  whenever  $a_k \neq *$ . In particular, cell  $a$  is called a **parent** of cell  $b$ , and  $b$  is a **child** of  $a$ , if and only if  $j = i + 1$ .

**Example 3.10. Ancestor and descendant cells.** Referring to Example 3.9, 1-D cell  $a = (\textit{Jan}, *, *, 2800)$  and 2-D cell  $b = (\textit{Jan}, *, \textit{Business}, 150)$  are *ancestors* of 3-D cell  $c = (\textit{Jan}, \textit{Chicago}, \textit{Business}, 45)$ ;  $c$  is a *descendant* of both  $a$  and  $b$ ;  $b$  is a *parent* of  $c$ ; and  $c$  is a *child* of  $b$ . □

“How many cuboids are there in an  $n$ -dimensional data cube?” If there is no hierarchy associated with any dimension, then the total number of cuboids for an  $n$ -dimensional data cube, as we have seen, is  $\binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{n} = 2^n$ . However, in practice, many dimensions do have hierarchies. For example, *time* is often explored at multiple conceptual levels such as in the hierarchy “*day < month < quarter < year*.” On a dimension that is associated with  $L$  levels, cuboid has  $L + 1$  possible choices, that is, one of the  $L$  levels or the virtual top level `all` meaning not including the dimension in the group-by. Thus, for an  $n$ -dimensional data cube, the total number of cuboids that can be generated (including the cuboids generated by climbing up the hierarchies along each dimension) is

$$\text{Total number of cuboids} = \prod_{i=1}^n (L_i + 1), \quad (3.1)$$

where  $L_i$  is the number of levels associated with dimension  $i$ .

For example, the time dimension as specified before has four conceptual levels, or five if we include the virtual level `all`. If the cube has 10 dimensions and each dimension has five levels (including `all`), the total number of cuboids that can be generated is  $5^{10} \approx 9.8 \times 10^6$ . The size of each cuboid, that is, the number of cells in a cuboid, also depends on the *cardinality* (i.e., number of distinct values) of each dimension. For example, if every item is sold in each city, there would be  $|city| \times |item|$  tuples in the (*city*, *item*) group-by alone. As the number of dimensions, number of conceptual hierarchies, or cardinality increases, the storage space required for many of the group-by’s will grossly exceed the (limited) size of the input relation. Indeed, given a base table and a set of dimensions, how to fast calculate or estimate the number of tuples in the resulting data cube remains an unsolved challenge.

### 3.4.2 Data cube materialization: ideas

By now, you probably realize that in large-scale applications, it may not be desirable or realistic to precompute and materialize all cuboids that can possibly be generated for a data cube (i.e., from a base cuboid). If there are many cuboids, and these cuboids are large in size, a more reasonable option is *partial materialization*; that is, to materialize only *some* of the possible cuboids that can be generated.

There are three possible choices for data cube materialization.

1. **No materialization:** Do not precompute any of the “nonbase” cuboids. This leads to computing expensive multidimensional aggregates on-the-fly, which can be extremely slow.
2. **Full materialization:** Precompute all of the cuboids. The resulting lattice of computed cuboids is referred to as the *full cube*. This choice typically requires huge amounts of memory space in order to store all of the precomputed cuboids.
3. **Partial materialization:** Selectively compute a proper subset of the whole set of possible cuboids, such as a subset of the cube that contains only those cells that satisfy some user-specified criterion (e.g., the aggregate count of each cell is above some threshold). We use the term *subcube* to refer to the latter case, where only some of the cells may be precomputed for various cuboids. Partial materialization of data cubes offers an interesting trade-off between storage space and response time for OLAP. Instead of computing the full cube, we can compute only a subset of the data cube’s cuboids, or subcubes consisting of subsets of cells from the various cuboids.

Nonetheless, full cube computation algorithms are important. We can use such algorithms to compute smaller cubes, consisting of a subset of the given set of dimensions, or a smaller range of possible values for some of the dimensions. In these cases, the smaller cube is a full cube for the given subset of dimensions and/or dimension values. A thorough understanding of full cube computation methods will help us develop efficient methods for computing partial cubes. Hence, it is important to explore scalable methods for computing all the cuboids making up a data cube, that is, for full materialization. These methods must take into consideration the limited amount of main memory available for cuboid computation, the total size of the computed data cube, as well as the time required for such computation.

Many cells in a cuboid may actually be of little or no interest to data analysts. Recall that each cell in a full cube records an aggregate value such as `count` or `sum`. For many cells in a cuboid, the measure value will be zero. For example, if item “snow-tire” is not sold in city “Pheonix” in June at all, the corresponding aggregate cell will have measure value of 0 for `count` or `sum`. In a cuboid, when most of the cells have measure 0, that is, the product of the cardinalities for the dimensions in the cuboid is much larger than the number of nonzero-valued tuples stored in the cuboid, then we say that the cuboid is **sparse**. If a cube contains many sparse cuboids, we say that the cube is **sparse**.

In many cases, a substantial amount of the cube’s space could be taken up by a large number of cells with very low measure values. This is because the cube cells are often quite sparsely distributed within a multidimensional space. For example, a customer may only buy a few items in a store at a time. Such an event will generate only a few nonempty cells, leaving most other cube cells empty. In such situations, it is useful to materialize only those cells in a cuboid (group-by) with a measure value above some minimum threshold. In a data cube about sales, say, we may wish to materialize only those cells for which `count`  $\geq 10$  (i.e., where at least 10 tuples exist for the cell’s given combination of dimensions) or only those cells representing `sales`  $\geq \$100$ . This not only saves processing time and disk space but also leads to a more focused analysis. The cells that cannot pass the threshold are likely to be too trivial to warrant further analysis.

Such partially materialized cubes are known as **iceberg cubes**. The minimum threshold is called the **minimum support threshold**, or *minimum support* (*min\_sup*), for short. By materializing only a fraction of the cells in a data cube, the result is seen as the “tip of the iceberg,” where the “iceberg” is the potential full cube including all cells. An iceberg cube can be specified using an SQL query, as shown in Example 3.11.

**Example 3.11. Iceberg cube.** Consider the following iceberg cube query.

```
compute cube sales_iceberg as
select month, city, customer_group, count(*)
from salesInfo
cube by month, city, customer_group
having count(*) >= min_sup
```

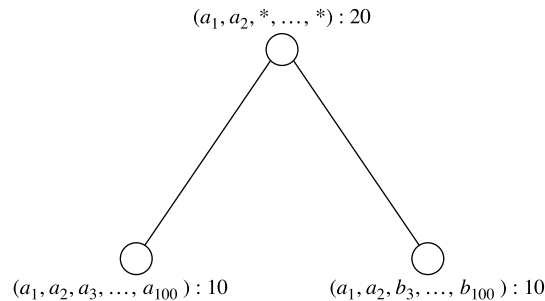
The compute cube statement specifies the precomputation of the iceberg cube, *sales\_iceberg*, with three dimensions, *month*, *city*, and *customer\_group*, and the aggregate measure *count()*. The input tuples are in the *salesInfo* relation. The cube by clause specifies that aggregates (group-by’s) are to be formed for each of the possible subsets of the given dimensions. If we were computing the full cube, each group-by would correspond to a cuboid in the data cube lattice. The constraint specified in the having clause is known as the **iceberg condition**. Here, the iceberg measure is *count()*. Note that the iceberg cube computed here can be used to answer group-by queries on any combination of the specified dimensions of the form having *count(\*)*  $\geq v$ , where  $v \geq \text{min\_sup}$ . Instead of *count()*, the iceberg condition may specify more complex measures, such as *average()*.

If we were to omit the having clause, we would end up with the full cube. Let us call this cube *sales\_cube*. The iceberg cube, *sales\_iceberg*, excludes all the cells of *sales\_cube* with a count that is less than *min\_sup*. Obviously, if we were to set the minimum support to 1 in *sales\_iceberg*, the resulting cube would be the full cube, *sales\_cube*.  $\square$

A naïve approach to computing an iceberg cube would be to first compute the full cube and then prune the cells that do not satisfy the iceberg condition. However, this is still prohibitively expensive. An efficient approach is to compute only the iceberg cube directly without computing the full cube. Section 3.5.2 discusses methods for efficient iceberg cube computation.

Introducing iceberg cubes lessens the burden of computing trivial aggregate cells in a data cube. However, we may still end up with a large number of uninteresting cells to compute. For example, suppose that there are 2 base cells for a database of 100 dimensions, denoted as  $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10\}$ , where each has a cell count of 10. If the minimum support is set to 10, there are still be an impermissible number of cells to compute and store, although most of them are not interesting. For example, there are  $2^{101} - 6$  distinct aggregate cells,<sup>2</sup> like  $\{(a_1, a_2, a_3, a_4, \dots, a_{99}, *) : 10, \dots, (a_1, a_2, *, a_4, \dots, a_{99}, a_{100}) : 10, \dots, (a_1, a_2, a_3, *, \dots, *, *) : 10\}$ , but most of them do not contain much new information. If we ignore all the aggregate cells that can be obtained by replacing some constants with \*’s while keeping the same measure value, there are only three distinct cells left:  $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10, (a_1, a_2, *, \dots, *) : 20\}$ . That is, out of  $2^{101} - 4$  distinct base and aggregate cells, only three really offer valuable information.

<sup>2</sup> The proof is left as an exercise for the reader.



**FIGURE 3.19**

Three closed cells forming the lattice of a closed cube.

To systematically compress a data cube, we need to introduce the concept of *closed coverage*. The **coverage** of a cell  $c$  is the set of base cells that are descendants of  $c$ . The measure of  $c$  is computed by the base cells that are descendants of  $c$ . In other words, the measure of  $c$  is determined by the coverage of  $c$ . Clearly, if two cells  $c_1$  and  $c_2$  have the same coverage, they have same measure no matter what aggregate functions are used. Based on this observation, A cell,  $c$ , is a *closed cell* if there exists no cell,  $d$ , such that  $d$  is a descendant of  $c$  (i.e.,  $d$  is obtained by replacing at least one  $*$  in  $c$  with a non- $*$  value), and  $d$  has the same coverage as  $c$ . A **quotient cube** is a data cube consisting of only closed cells. For example, the three cells derived in the preceding paragraph are the three closed cells of the data cube for the data set  $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10\}$ . They form the lattice of a closed cube as shown in Fig. 3.19. Other nonclosed cells can be derived from their corresponding closed cells in this lattice. For example, “ $(a_1, *, *, \dots, *) : 20$ ” can be derived from “ $(a_1, a_2, *, \dots, *) : 20$ ” because the former is a generalized nonclosed cell of the latter. Similarly, we have “ $(a_1, a_2, b_3, *, \dots, *) : 10$ .”

Another strategy for partial materialization is to precompute only the cuboids involving a small number of dimensions such as three to five. These cuboids form a **cube shell** for the corresponding data cube. Queries on additional combinations of the dimensions will have to be computed on-the-fly. For example, we could compute all cuboids with three dimensions or less in an  $n$ -dimensional data cube, resulting in a cube shell of size 3. This, however, can still result in a large number of cuboids to compute, particularly when  $n$  is large. Alternatively, we can choose to precompute only portions or *fragments* of the cube shell based on cuboids of interest. Section 3.5.3 discusses a method for computing **shell fragments** and explores how they can be used for efficient OLAP query processing.

### 3.4.3 OLAP server architectures: ROLAP vs. MOLAP vs. HOLAP

There are many methods for efficient data cube computation, based on the various kinds of cubes described earlier in this section. In general, there are two basic data structures used for storing cuboids. The implementation of relational OLAP (ROLAP) uses relational tables, whereas multidimensional arrays are used in multidimensional OLAP (MOLAP). In some situations, we may also combine ROLAP and MOLAP to obtain the hybrid OLAP (HOLAP) approach. Let us look at the details here.

Logically, OLAP servers present business users with multidimensional data from data warehouses or data marts, without concerns regarding how or where the data are stored. However, the physical

architecture and implementation of OLAP servers must consider data storage issues. Implementations of a data warehouse server for OLAP processing may have the following options.

**Relational OLAP (ROLAP) servers:** These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use a *relational* or *extended-relational DBMS* to store and manage warehouse data, and OLAP middleware to support missing pieces. ROLAP servers include optimization for each DBMS back end, implementation of aggregation navigation logic, and additional tools and services. ROLAP technology tends to have greater scalability than MOLAP technology.

**Multidimensional OLAP (MOLAP) servers:** These servers support multidimensional data views through *array-based multidimensional storage engines*. They map multidimensional views directly to data cube array structures. The advantage of using a data cube is that it allows fast indexing to precomputed summarized data. Notice that with multidimensional data stores, the storage utilization may be low if the data set is sparse. In such cases, sparse matrix compression techniques should be explored.

Many MOLAP servers adopt a two-level storage representation to handle dense and sparse data sets: Denser subcubes are identified and stored as array structures, whereas sparse subcubes employ compression technology for efficient storage utilization.

**Hybrid OLAP (HOLAP) servers:** The hybrid OLAP approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detailed data to be stored in a relational database, whereas aggregations are kept in a separate MOLAP store.

**Specialized SQL servers:** To meet the growing demand of OLAP processing in relational databases, some database system vendors implement specialized SQL servers that provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

“How are data actually stored in ROLAP and MOLAP architectures?” Let’s first look at ROLAP. As its name implies, ROLAP uses relational tables to store data for online analytical processing. Recall that the fact table associated with a base cuboid is referred to as a *base fact table*. The base fact table stores data at the abstraction level indicated by the join keys in the schema for the given data cube. Aggregated data can also be stored in fact tables, referred to as **summary fact tables**. Some summary fact tables store both base fact table data and aggregated data. Alternatively, separate summary fact tables can be used for each abstraction level to store only aggregated data.

**Example 3.12. A ROLAP data store.** Table 3.3 shows a summary fact table that contains both base fact data and aggregated data. The schema is “*(record\_identifier (RID), item, . . . , day, month, quarter, year, dollars\_sold)*,” where *day*, *month*, *quarter*, and *year* define the sales date, and *dollars\_sold* is the sales amount. Consider the tuples with an *RID* of 1001 and 1002, respectively. The data of these tuples are at the base fact level, where the sales dates are October 15, 2010, and October 23, 2010, respectively. Consider the tuple with an *RID* of 5001. This tuple is at a more general level of abstraction than the tuples 1001 and 1002. The *day* value has been generalized to all, so that the corresponding *time* value is October 2010. That is, the *dollars\_sold* amount shown is an aggregation representing the entire month of October 2010, rather than just October 15 or 23, 2010. The special value all is used to represent subtotals in summarized data. □

RID	item	...	day	month	quarter	year	dollars_sold
1001	TV	...	15	10	Q4	2010	250.60
1002	TV	...	23	10	Q4	2010	175.00
...	...	...	...	...	...	...	...
5001	TV	...	all	10	Q4	2010	45,786.08
...	...	...	...	...	...	...	...

MOLAP uses multidimensional array structures to store data for online analytical processing.

Most data warehouse systems adopt a client-server architecture. A relational data store always resides at the data warehouse/data mart server site. A multidimensional data store can reside at either the database server site or the client site.

### 3.4.4 General strategies for data cube computation

Although ROLAP and MOLAP may each explore different cube computation techniques, some optimization techniques are popularly used.

#### Optimization Technique 1: sorting, hashing, and grouping

Sorting, hashing, and grouping operations should be applied to the dimension attributes to reorder and cluster related tuples.

In cube computation, aggregation is performed on the tuples (or cells) that share the same set of dimension values. Thus it is important to explore sorting, hashing, and grouping operations to access and group such data together to facilitate computation of such aggregates.

To compute total sales by *branch*, *day*, and *item*, for example, it can be more efficient to sort tuples or cells first by *branch*, then by *day*, and last by *item*. Using the sorted data, it is easy to group them according to the *item* name. Efficient implementations of such operations in large data sets have been extensively studied in the algorithm and database research communities, such as counting sort. Such implementations can be extended to data cube computation.

This technique can also be further extended to perform **shared-sorts** (i.e., sharing sorting costs across multiple cuboids when sort-based methods are used), or to perform **shared-partitions** (i.e., sharing the partitioning cost across multiple cuboids when hash-based algorithms are used). For example, using the data sorted first by *branch*, then by *day* and last by *item*, we can compute not only the cuboid (*branch*, *day*, *item*) but also the cuboids (*branch*, *day*, \*), (*branch*, \*, \*) and ().

#### Optimization Technique 2: simultaneous aggregation and caching of intermediate results

In cube computation, it is efficient to compute higher-level aggregates from previously computed lower-level aggregates, rather than from the base fact table, since the number of tuples of higher-level aggregates is far less than the number of tuples at the base fact table. For example, to compute the total sales amount of a year, it is more efficient to aggregate from the subtotals of different items of the year. Moreover, simultaneous aggregation from cached intermediate computation results may lead to a reduction of expensive disk input/output (I/O) operations. This technique can be further extended to

perform **amortized scans** (i.e., computing as many cuboids as possible at the same time to amortize disk reads).

Optimization Technique 3: aggregation from the smallest child when there exist multiple child cuboids

When there exist multiple child cuboids, it is usually more efficient to compute the desired parent (i.e., more generalized) cuboid from the smallest in size, previously computed child cuboid. For example, to compute a sales cuboid,  $C_{branch}$ , when there exist two previously computed cuboids,  $C_{\{branch,year\}}$  and  $C_{\{branch,item\}}$ , it is obviously more efficient to compute  $C_{branch}$  from the former than from the latter if there are many more distinct items than distinct years.

Optimization Technique 4: the downward antimonotonicity can be used to prune search space in iceberg cube computation

For many aggregate measures, the downward antimonotonicity may hold. *If a given cell does not satisfy the iceberg condition, then no descendant of the cell (i.e., more specialized cell) can satisfy the iceberg condition.*

For example, consider the iceberg condition “`count(*) >= 1000`.” If a cell  $(*, \text{Bellingham}, *)$ : 800 fails the iceberg condition, then any descendant of the cell, such as  $(\text{March}, \text{Bellingham}, *)$  and  $(*, \text{Bellingham}, \text{small-business})$ , must also fail the condition, and thus cannot be entitled to be included in the iceberg cube.

The antimonotonicity property can be used to substantially reduce the computation of iceberg cubes. A common iceberg condition is that the cells must satisfy a *minimum support* threshold such as a minimum count or sum. In this situation, the antimonotonicity property can be used to prune away the exploration of the cell’s descendants.

In the next section, we introduce several popular methods for efficient cube computation that explore these optimization strategies.

---

## 3.5 Data cube computation methods

Data cube computation is an essential task in data warehouse implementation. The precomputation of all or part of a data cube can greatly reduce the response time and enhance the performance of online analytical processing. However, such computation is challenging because it may require substantial computational time and storage space. This section explores efficient methods for data cube computation. Section 3.5.1 describes the *multiway array aggregation* (MultiWay) method for computing full cubes. Section 3.5.2 describes the BUC method, which computes iceberg cubes from the apex cuboid downward. Section 3.5.3 describes a shell-fragment cubing approach that computes shell fragments for efficient high-dimensional OLAP. Last, Section 3.5.4 demonstrates how to answer OLAP queries using cuboids in data cubes.

To simplify our discussion, we exclude the cuboids that would be generated by climbing up any existing hierarchies for the dimensions. Those cube types can be computed by straightforward extensions of the discussed methods. Methods for the efficient computation of closed cubes are left as an exercise for interested readers.

### 3.5.1 Multiway array aggregation for full cube computation

The **multiway array aggregation** (or simply **MultiWay**) method computes a full data cube using a multidimensional array as its basic data structure. It is a typical MOLAP approach that uses direct array addressing, where dimension values are accessed via the position or index of their corresponding array locations. MultiWay constructs an array-based cube as follows.

1. Partition the array into chunks. A **chunk** is a subcube that is small enough to fit into the memory available for cube computation. **Chunking** is a method for dividing an  $n$ -dimensional array into small  $n$ -dimensional chunks, where each chunk is stored as an object on disk. The chunks are compressed to remove wasted space resulting from *empty array cells*. A cell is *empty* if it does not contain any valid data (i.e., its cell count is 0). For instance, “*chunkID + offset*” can be used as a cell-addressing mechanism to **compress a sparse array structure** and when searching for cells within a chunk. Such a compression technique is powerful at handling sparse cubes, both on disk and in memory.
2. Compute aggregates by visiting (i.e., accessing the values at) cube cells. The order in which cells are visited can be optimized to *minimize the number of times that each cell must be revisited*, thereby reducing memory access and storage costs. The idea is to exploit this ordering so that portions of the aggregate cells in multiple cuboids can be computed simultaneously, and any unnecessary revisiting of cells is avoided.

This chunking technique involves “overlapping” some of the aggregation computations; therefore it is referred to as multiway array aggregation. It performs **simultaneous aggregation**, that is, it computes aggregations simultaneously on multiple dimensions.

We explain this approach to array-based cube construction by looking at a concrete example.

**Example 3.13. Multiway array cube computation.** Consider a 3-D data array containing three dimensions  $A$ ,  $B$ , and  $C$ . The 3-D array is partitioned into small, memory-based chunks. In this example, the array is partitioned into 64 chunks as shown in Fig. 3.20. Dimension  $A$  is organized into four equal-sized partitions:  $a_0, a_1, a_2$ , and  $a_3$ . Dimensions  $B$  and  $C$  are similarly organized into four partitions each. Chunks 1, 2, ..., 64 correspond to the subcubes  $a_0b_0c_0, a_1b_0c_0, \dots, a_3b_3c_3$ , respectively. Suppose that the cardinality of the dimensions  $A$ ,  $B$ , and  $C$  is 40, 400, and 4000, respectively. Thus the size of the array for each dimension,  $A$ ,  $B$ , and  $C$ , is also 40, 400, and 4000, respectively. Since the number of partitions of each dimension is 4, the size of each partition in  $A$ ,  $B$ , and  $C$  is therefore 10, 100, and 1000, respectively. Full materialization of the corresponding data cube involves the computation of all the cuboids defining this cube. The resulting full cube consists of the following cuboids:

- The base cuboid, denoted by  $ABC$  (from which all the other cuboids are directly or indirectly computed). This cube is already computed and corresponds to the given 3-D array.
- The 2-D cuboids,  $AB$ ,  $AC$ , and  $BC$ , which respectively correspond to the group-by’s  $AB$ ,  $AC$ , and  $BC$ . These cuboids need to be computed.
- The 1-D cuboids,  $A$ ,  $B$ , and  $C$ , which respectively correspond to the group-by’s  $A$ ,  $B$ , and  $C$ . These cuboids need to be computed.
- The 0-D (apex) cuboid, denoted by  $all$ , which corresponds to the group-by ( $()$ ); that is, there is no group-by here. These cuboids need to be computed. It consists of only one value. If, say, the data cube measure is `count`, then the value to be computed is simply the total count of all the tuples in  $ABC$ .



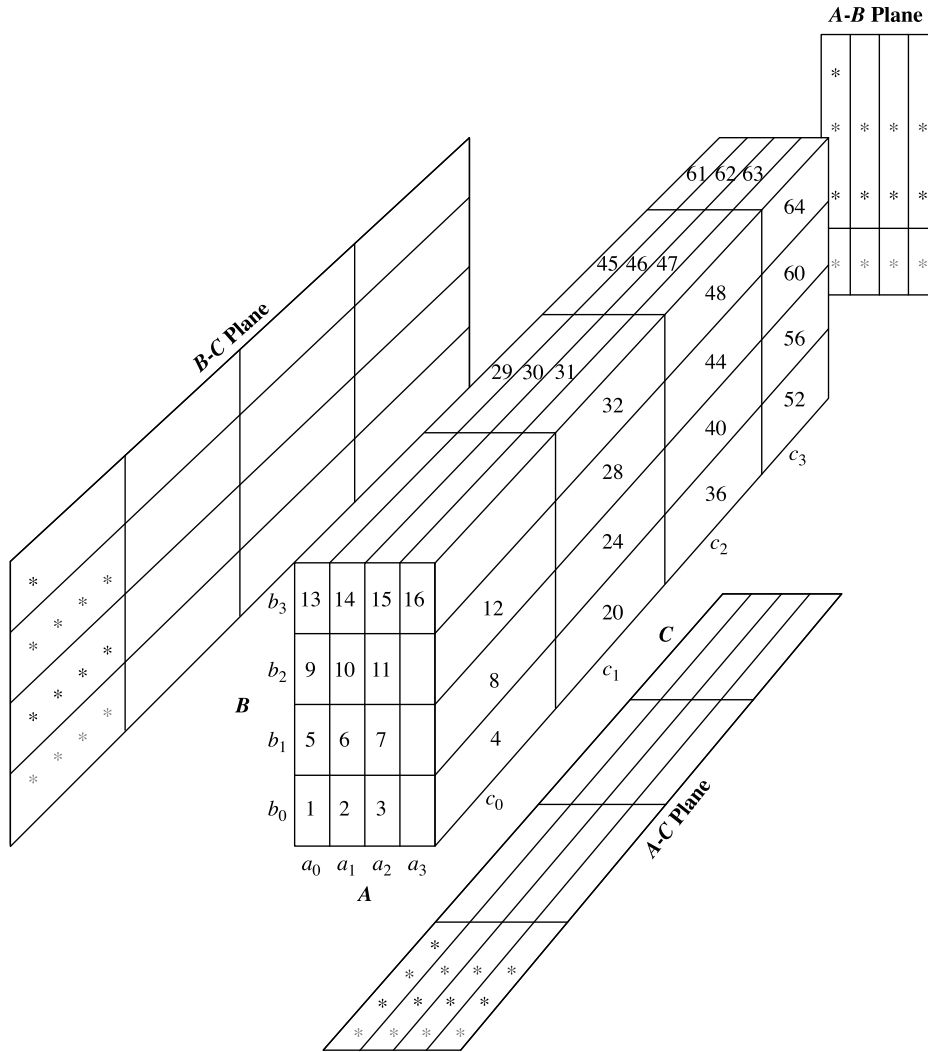


FIGURE 3.20

A 3-D array for the dimensions A, B, and C, organized into 64 *chunks*. Each chunk is small enough to fit into the memory available for cube computation. The \*'s indicate the chunks from 1 to 13 that have been aggregated so far in the process.

Let's look at how the multiway array aggregation technique is used in this computation. There are many possible orderings with which chunks can be read into memory for use in cube computation. Consider the ordering labeled from 1 to 64, shown in Fig. 3.20. Suppose we want to compute the  $b_0c_0$  chunk of the *BC* cuboid. We allocate space for this chunk in *chunk memory*. By scanning *ABC* chunks

1 through 4, the  $b_0c_0$  chunk is computed. That is, the cells for  $b_0c_0$  are aggregated over  $a_0$  to  $a_3$ . The chunk memory can then be assigned to the next chunk,  $b_1c_0$ , which completes its aggregation after the scanning of the next four  $ABC$  chunks: 5 through 8. Continuing in this way, the entire  $BC$  cuboid can be computed. Therefore only *one*  $BC$  chunk needs to be in memory at a time for the computation of all the  $BC$  chunks.

In computing the  $BC$  cuboid, we will have scanned each of the 64 chunks. “*Is there a way to avoid having to rescan all of these chunks for the computation of other cuboids, such as  $AC$  and  $AB$ ?*” The answer is, most definitely, *yes*. This is where the “multiway computation” or “simultaneous aggregation” idea comes in. For example, when chunk 1 (i.e.,  $a_0b_0c_0$ ) is being scanned (say, for the computation of the 2-D chunk  $b_0c_0$  of  $BC$ , as described previously), all of the other 2-D chunks related to  $a_0b_0c_0$  can be simultaneously computed. That is, when  $a_0b_0c_0$  is being scanned, each of the three chunks ( $b_0c_0$ ,  $a_0c_0$ , and  $a_0b_0$ ) on the three 2-D aggregation planes ( $BC$ ,  $AC$ , and  $AB$ ) should be computed then as well. In other words, multiway computation simultaneously aggregates to each of the 2-D planes while a 3-D chunk is in memory.

Now let’s look at how different orderings of chunk scanning and of cuboid computation can affect the overall data cube computation efficiency. Recall that the size of the dimensions  $A$ ,  $B$ , and  $C$  is 40, 400, and 4000, respectively. Therefore the largest 2-D plane is  $BC$  (of size  $400 \times 4000 = 1,600,000$ ). The second largest 2-D plane is  $AC$  (of size  $40 \times 4000 = 160,000$ ).  $AB$  is the smallest 2-D plane (of size  $40 \times 400 = 16,000$ ).

Suppose that the chunks are scanned in the order shown, from chunks 1 to 64. As previously mentioned,  $b_0c_0$  is fully aggregated after scanning the row containing chunks 1 through 4;  $b_1c_0$  is fully aggregated after scanning chunks 5 through 8, and so on. Thus we need to scan four chunks of the 3-D array to *fully* compute one chunk of the  $BC$  cuboid (where  $BC$  is the largest of the 2-D planes). In other words, by scanning in this order, one  $BC$  chunk is fully computed for each row scanned. In comparison, the complete computation of one chunk of the second largest 2-D plane,  $AC$ , requires scanning 13 chunks, given the ordering from 1 to 64. That is,  $a_0c_0$  is fully aggregated only after the scanning of chunks 1, 5, 9, and 13.

Finally, the complete computation of one chunk of the smallest 2-D plane,  $AB$ , requires scanning 49 chunks. For example,  $a_0b_0$  is fully aggregated after scanning chunks 1, 17, 33, and 49. Hence,  $AB$  requires the longest scan of chunks to complete its computation. To avoid bringing a 3-D chunk into memory more than once, the minimum memory requirement for holding all relevant 2-D planes in chunk memory, according to the chunk ordering of 1 to 64, is as follows:  $40 \times 400$  (for the whole  $AB$  plane) +  $40 \times 1000$  (for one column of the  $AC$  plane) +  $100 \times 1000$  (for one  $BC$  plane chunk) =  $16,000 + 40,000 + 100,000 = 156,000$  memory units.

Suppose, instead, that the chunks are scanned in the order 1, 17, 33, 49, 5, 21, 37, 53, and so on. That is, suppose the scan is in the order of first aggregating toward the  $AB$  plane and then toward the  $AC$  plane, and lastly toward the  $BC$  plane. The minimum memory requirement for holding 2-D planes in chunk memory would be as follows:  $400 \times 4000$  (for the whole  $BC$  plane) +  $10 \times 4000$  (for one  $AC$  plane row) +  $10 \times 100$  (for one  $AB$  plane chunk) =  $1,600,000 + 40,000 + 1000 = 1,641,000$  memory units. Notice that this is *more than 10 times* the memory requirement of the scan ordering of 1 to 64.

Similarly, we can work out the minimum memory requirements for the multiway computation of the 1-D and 0-D cuboids. Fig. 3.21 shows the most efficient way to compute 1-D cuboids. Chunks for 1-D cuboids  $A$  and  $B$  are computed during the computation of the smallest 2-D cuboid,  $AB$ . The smallest 1-D cuboid,  $A$ , will have all of its chunks allocated in memory, whereas the larger 1-D cuboid,

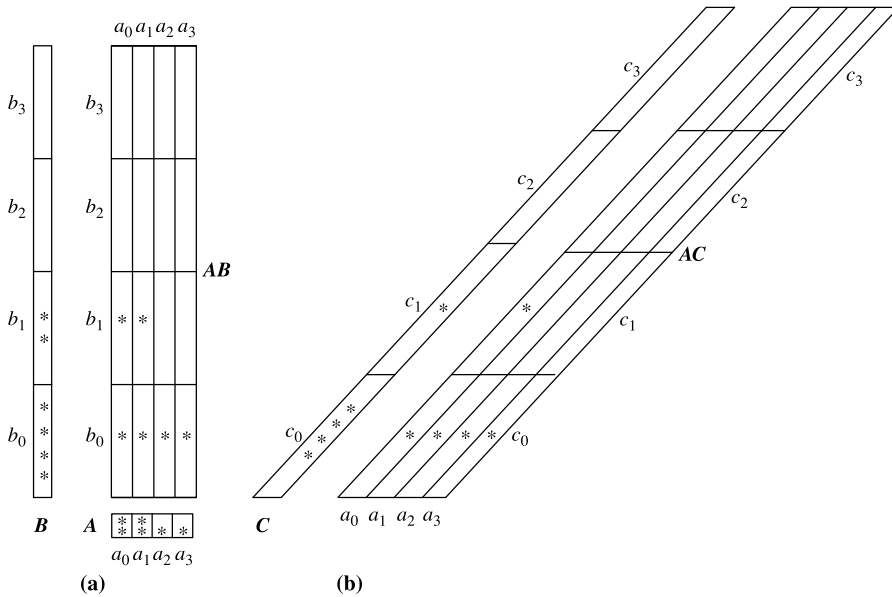


FIGURE 3.21

Memory allocation and computation order for computing Example 5.4’s 1-D cuboids. (a) The 1-D cuboids, A and B, are aggregated during the computation of the smallest 2-D cuboid, AB. (b) The 1-D cuboid, C, is aggregated during the computation of the second smallest 2-D cuboid, AC. The \*’s represent chunks that so far have been aggregated to.

B, will have only one chunk allocated in memory at a time. Similarly, chunk C is computed during the computation of the second smallest 2-D cuboid, AC, requiring only one chunk in memory at a time. Based on this analysis, we see that the most efficient ordering in this array cube computation is the chunk ordering of 1 to 64, with the stated memory allocation strategy. □

Example 3.13 assumes that there is enough memory space for *one-pass* cube computation (i.e., to compute all of the cuboids from one scan of all the chunks). If there is insufficient memory space, the computation will require more than one pass through the 3-D array. In such cases, however, the basic principle of ordered chunk computation remains the same. MultiWay is most effective when the product of the cardinalities of dimensions is moderate and the data are not too sparse. When the dimensionality is high or the data are very sparse, the in-memory arrays become too large to fit in memory, and this method becomes impractical.

With the use of appropriate sparse array compression techniques and careful ordering of the computation of cuboids, it has been shown by experiments that MultiWay array cube computation is significantly faster than traditional ROLAP (relational record-based) computation. Unlike ROLAP, the array structure of MultiWay does not require saving space to store search keys. Furthermore, MultiWay uses direct array addressing, which is faster than ROLAP’s key-based addressing search strategy. For ROLAP cube computation, instead of cubing a table directly, it can be faster to convert the table to an

array, cube the array, and then convert the result back to a table. However, this observation works only for cubes with a relatively small number of dimensions, because the number of cuboids to be computed is exponential to the number of dimensions.

“*What would happen if we tried to use MultiWay to compute iceberg cubes?*” Remember that the downward antimonotonicity property states that if a given cell does not satisfy the iceberg property, then neither will any of its descendants. Unfortunately, MultiWay’s computation starts from the base cuboid and progresses upward toward more generalized, ancestor cuboids. It cannot take advantage of possible pruning using the antimonotonicity, which requires a parent node to be computed before its child (i.e., more specific) nodes. For example, if the count of a cell  $c$  in, say,  $AB$ , does not satisfy the minimum support specified in the iceberg condition, we cannot prune away cell  $c$ , because the count of  $c$ ’s ancestors in the  $A$  or  $B$  cuboids may be greater than the minimum support, and their computation will need aggregation involving the count of  $c$ .

### 3.5.2 BUC: computing iceberg cubes from the apex cuboid downward

**BUC** is an algorithm for the computation of sparse and iceberg cubes. Unlike MultiWay, BUC constructs the cube from the apex cuboid toward the base cuboid. This allows BUC to share data partitioning costs. This processing order also allows BUC to prune during construction, using the downward antimonotonicity property.

Fig. 3.22 shows a lattice of cuboids, making up a 3-D data cube with the dimensions  $A$ ,  $B$ , and  $C$ . The apex (0-D) cuboid, representing the concept all (i.e.,  $(*, *, *)$ ), is at the top of the lattice. This is the most aggregated or generalized level. The 3-D base cuboid,  $ABC$ , is at the bottom of the lattice. It is the least aggregated (most detailed or specialized) level. This representation of a lattice of cuboids, with the apex at the top and the base at the bottom, is commonly accepted in data warehousing. It consolidates the notions of *drill-down* (where we can move from a highly aggregated cell to lower, more detailed cells) and *roll-up* (where we can move from detailed, low-level cells to higher-level, more aggregated cells).

BUC stands for “Bottom-Up Construction.” However, according to the lattice convention described before and used throughout this book, the BUC processing order is actually top-down! The BUC authors view a lattice of cuboids in the reverse order, with the apex cuboid at the bottom and the base cuboid at the top. In that view, BUC does bottom-up construction. However, because we adopt the application worldview where *drill-down* refers to drilling from the apex cuboid down toward the base cuboid, the exploration process of BUC is regarded as top-down. BUC’s exploration for the computation of a 3-D data cube is shown in Fig. 3.22.

The BUC algorithm is shown in Fig. 3.23. We first give an explanation of the algorithm and then follow up with an example. Initially, the algorithm is called with the input relation (set of tuples). BUC aggregates the entire input (line 1) and writes the resulting total (line 3). (Line 2 is an optimization feature that is discussed later in our example.) For each dimension  $d$  (line 4), the input is partitioned on  $d$  (line 6). On return from `Partition()`, `dataCount` contains the total number of tuples for each distinct value of dimension  $d$ . Each distinct value of  $d$  forms its own partition. Line 8 iterates through each partition. Line 10 tests the partition for minimum support. That is, if the number of tuples in the partition satisfies (i.e., is  $\geq$ ) the minimum support, then the partition becomes the input relation for a recursive call made to BUC, which computes the iceberg cube on the partitions for dimensions  $d + 1$  to `numDims` (line 12).

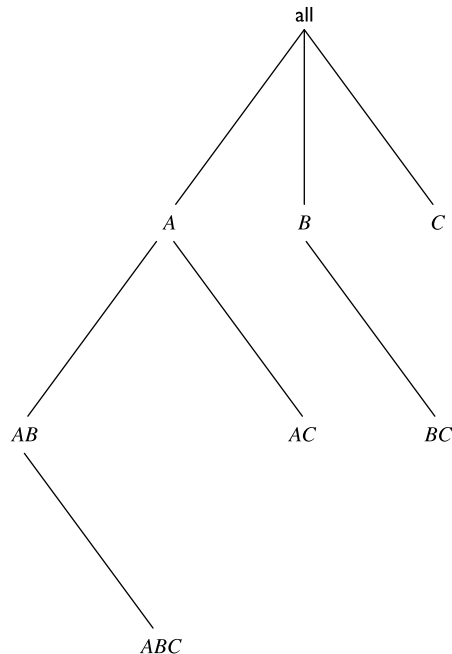


FIGURE 3.22

BUC's exploration for a 3-D data cube computation. Note that the computation starts from the apex cuboid.

Note that for a full cube (i.e., where minimum support in the having clause is 1), the minimum support condition is always satisfied. Thus the recursive call descends one level deeper into the lattice. On return from the recursive call, we continue with the next partition for  $d$ . After all the partitions have been processed, the entire process is repeated for each of the remaining dimensions.

**Example 3.14. BUC construction of an iceberg cube.** Consider the iceberg cube expressed in SQL as follows:

```

compute cube iceberg_cube as
select A, B, C, D, count(*)
from R
cube by A, B, C, D
having count(*) >= 3
  
```

Let's see how BUC constructs the iceberg cube for the dimensions  $A$ ,  $B$ ,  $C$ , and  $D$ , where 3 is the minimum support count. Suppose that dimension  $A$  has four distinct values,  $a_1, a_2, a_3, a_4$ ;  $B$  has four distinct values,  $b_1, b_2, b_3, b_4$ ;  $C$  has two distinct values,  $c_1, c_2$ ; and  $D$  has two distinct values,  $d_1, d_2$ . If we consider each group-by to be a *partition*, then we must compute every combination of the grouping attributes that satisfy the minimum support (i.e., that have three tuples).

Fig. 3.24 illustrates how the input is partitioned first according to the different attribute values of dimension  $A$  and then  $B$ ,  $C$ , and  $D$ . To do so, BUC scans the input, aggregating the tuples to obtain

**Algorithm: BUC.** Algorithm for the computation of sparse and iceberg cubes.

**Input:**

- *input*: the relation to aggregate;
- *dim*: the starting dimension for this iteration.

**Globals:**

- constant *numDims*: the total number of dimensions;
- constant *cardinality[numDims]*: the cardinality of each dimension;
- constant *min\_sup*: the minimum number of tuples in a partition for it to be output;
- *outputRec*: the current output record;
- *dataCount[numDims]*: stores the size of each partition. *dataCount[i]* is a list of integers of size *cardinality[i]*.

**Output:** Recursively output the iceberg cube cells satisfying the minimum support.

**Method:**

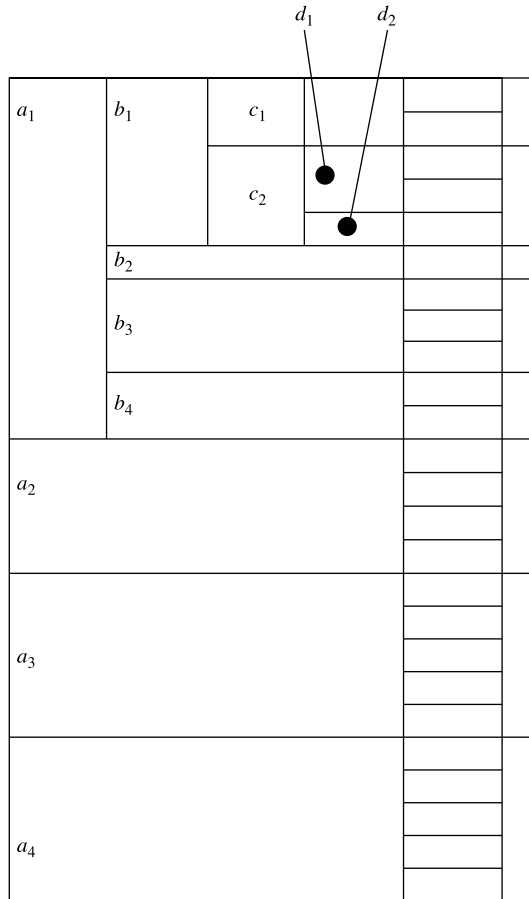
- (1) Aggregate(input); // Scan *input* to compute measure, e.g., count. Place result in *outputRec*.
- (2) **if** input.count() == 1 **then** // Optimization  
    WriteDescendants(input[0], dim); **return**;  
    **endif**
- (3) write outputRec;
- (4) **for** (*d = dim*; *d < numDims*; *d ++*) **do** //Partition each dimension
- (5)     *C* = cardinality[d];
- (6)     Partition(input, d, *C*, dataCount[d]); //create *C* partitions of data for dimension *d*
- (7)     *k* = 0;
- (8)     **for** (*i = 0*; *i < C*; *i ++*) **do** // for each partition (each value of dimension *d*)
- (9)         *c* = dataCount[d][*i*];
- (10)         **if** *c* >= *min\_sup* **then** // test the iceberg condition
- (11)             outputRec.dim[d] = input[k].dim[d];
- (12)             BUC(input[k..*k + c - 1*], *d + 1*); // aggregate on next dimension
- (13)         **endif**
- (14)         *k* += *c*;
- (15)     **endfor**
- (16)     outputRec.dim[d] = all;
- (17) **endfor**

**FIGURE 3.23**

BUC algorithm for sparse or iceberg cube computation. *Source:* Beyer and Ramakrishnan [BR99].

a count for all, corresponding to the cell  $(*, *, *, *)$ . Dimension *A* is used to split the input into four partitions, one for each distinct value of *A*. The number of tuples (counts) for each distinct value of *A* is recorded in *dataCount*.

BUC uses the downward antimonotonicity property to save time while searching for tuples that satisfy the iceberg condition. Starting with *A* dimension value,  $a_1$ , the  $a_1$  partition is aggregated, creating one tuple for the *A* group-by, corresponding to the cell  $(a_1, *, *, *)$ . Suppose  $(a_1, *, *, *)$  satisfies the minimum support, in which case a recursive call is made on the partition for  $a_1$ . BUC partitions  $a_1$  on the dimension *B*. It checks the count of  $(a_1, b_1, *, *)$  to see if it satisfies the minimum support. If it does, it outputs the aggregated tuple to the *AB* group-by and recurses on  $(a_1, b_1, *, *)$  to partition on *C*, starting with  $c_1$ . Suppose the cell count for  $(a_1, b_1, c_1, *)$  is 2, which does not satisfy the minimum support. According to the downward antimonotonicity property, if a cell does not satisfy the minimum support, then neither can any of its descendants. Therefore, BUC prunes any further exploration of



**FIGURE 3.24**

BUC partitioning snapshot given an example 4-D data set.

$(a_1, b_1, c_1, *)$ . That is, it avoids partitioning this cell on dimension  $D$ . It backtracks to the  $a_1, b_1$  partition and recurses on  $(a_1, b_1, c_2, *)$ , and so on. By checking the iceberg condition each time before performing a recursive call, BUC saves a great deal of processing time whenever a cell's count does not satisfy the minimum support.

The partition process is facilitated by a linear sorting method, CountingSort. CountingSort is fast because it does not perform any key comparisons to find partition boundaries. For example, to sort 10,000 tuples according to an attribute  $A$  whose value is an integer in the range between 1 and 100, we can set up 100 counters and scan the data once to count the number of 1's, 2's, ..., 100's on attribute  $A$ . Suppose there are  $i_1$  tuples having 1 on  $A$ ,  $i_2$  tuples having 2 on  $A$ , and so on. Then, in the next scan, we can move all the tuples having value 1 on attribute  $A$  to the first  $i_1$  slots, the tuples having value 2 on attribute  $A$  to the slots  $i_1 + 1, \dots, i_1 + i_2$ , and so on. After those two scans, the tuples are

sorted according to  $A$ . In addition, the counts computed during the sort can be reused to compute the group-by's in BUC.

Line 2 is an optimization for partitions having a count of 1 such as  $(a_1, b_2, *, *)$  in our example. To save on partitioning costs, the count is written to each of the tuple's descendant group-by's. This is particularly useful since, in practice, many partitions may have a single tuple.  $\square$

The BUC performance is sensitive to the order of the dimensions and to skew in the data. Ideally, the most discriminating dimensions should be processed first. Dimensions should be processed in the order of decreasing cardinality. The higher the cardinality, the smaller the partitions, and thus the more partitions there will be, thereby providing BUC with a greater opportunity for pruning. Similarly, the more uniform a dimension (i.e., having less skew), the better it is for pruning.

BUC's major contribution is the idea of sharing partitioning costs. However, unlike MultiWay, it does not share the computation of aggregates between parent and child group-by's. For example, the computation of cuboid  $AB$  does not help that of  $ABC$ . The latter needs to be computed essentially from scratch.

### 3.5.3 Precomputing shell fragments for fast high-dimensional OLAP

Materializing data cubes facilitates flexible and fast OLAP operations. However, computing full data cube of high dimensionality needs massive storage space and unrealistic computation time. Although there are proposals of computing iceberg cubes and closed cubes, they are still confined to low-dimensional data (e.g., less than 12 dimensions) and cannot handle the challenges of high dimensionality. One possible alternative is to compute a thin **cube shell**, such as computing all cuboids with three dimensions or less in a 60-dimensional data cube, resulting in a cube shell of size 3. However, such a cube shell cannot support OLAP or query involving four or more dimensions.

Here we introduce a *shell fragment* approach for high-dimensional OLAP, based on the following observation: Although a data cube may contain many dimensions, *most OLAP operations are performed only on a small number of dimensions relevant to some query-selected conditions at a time*. In other words, an OLAP query is likely to ignore many dimensions (i.e., treating them as irrelevant), constrain certain conditions in some dimensions (e.g., using query constants), and leave only a few to be manipulated (for drilling, pivoting, etc.). This is because it is neither realistic nor fruitful for anyone to comprehend thousands of cells involving dozens of dimensions simultaneously in a high-dimensional space at the same time.

Based on this observation, it is natural to first locate some low-dimensional cuboids of interest within a high-dimension cube and then conduct OLAP on such low-dimensional cuboids. This implies that if multidimensional aggregates can be computed quickly on a *small number of dimensions inside a high-dimensional space*, we may still achieve fast OLAP without materializing the original high-dimensional data cube. This leads to a *semionline computation approach*, called *shell fragment* as follows: First, given a base cuboid, we can precompute (i.e., offline) cube shell fragments. Then, when query comes, one can quickly assembly a low-dimensional cube online using the preprocessed data, and conduct OLAP operations. The shell fragment approach can handle databases of high dimensionality and can quickly compute small local cubes online. It explores the *inverted index* data structure, which is popular in information retrieval and Web-based information systems.

The basic idea is as follows. Given a high-dimensional data set, we partition the dimensions into a set of disjoint dimension *fragments*, convert each fragment into its corresponding inverted index repre-



TID	A	B	C	D	E
1	$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
2	$a_1$	$b_2$	$c_1$	$d_2$	$e_1$
3	$a_1$	$b_2$	$c_1$	$d_1$	$e_2$
4	$a_2$	$b_1$	$c_1$	$d_1$	$e_2$
5	$a_2$	$b_1$	$c_1$	$d_1$	$e_3$

sensation, and then construct *cube shell fragments* while keeping the inverted indices associated with the cube cells. Using the precomputed cubes' shell fragments, we can dynamically assemble and compute cuboid cells of the required data cube online. This is made efficient by set intersection operations on the inverted indices.

To illustrate the shell fragment approach, we use a tiny database of Table 3.4 as a running example. Let the cube measure be `count()`. Other measures will be discussed later. We first look at how to construct the inverted index for the given database.

**Example 3.15. Construct the inverted index.** For each attribute value in each dimension, list the tuple identifiers (*TIDs*) of all the tuples that have that value. For example, attribute value  $a_2$  appears in tuples 4 and 5. The TID list for  $a_2$  then contains exactly two items, namely 4 and 5. The resulting inverted index table is shown in Table 3.5. It retains all the information of the original database.  $\square$

“How do we compute shell fragments of a data cube?” We first partition all the dimensions of the given data set into independent groups of dimensions, called *fragments*. We scan the base cuboid and construct an inverted index for each attribute. For each fragment, we compute the full *local* (i.e., fragment-based) data cube while retaining the inverted indices. Consider a database of 60 dimensions, namely,  $A_1, A_2, \dots, A_{60}$ . We can first partition the 60 dimensions into 20 fragments of size 3, such as  $(A_1, A_2, A_3)$ ,  $(A_4, A_5, A_6)$ ,  $\dots$ ,  $(A_{58}, A_{59}, A_{60})$ . For each fragment, we compute its full data cube while recording the inverted indices. For example, in fragment  $(A_1, A_2, A_3)$ , we would compute seven

Attribute Value	TID List	List Size
$a_1$	{1, 2, 3}	3
$a_2$	{4, 5}	2
$b_1$	{1, 4, 5}	3
$b_2$	{2, 3}	2
$c_1$	{1, 2, 3, 4, 5}	5
$d_1$	{1, 3, 4, 5}	4
$d_2$	{2}	1
$e_1$	{1, 2}	2
$e_2$	{3, 4}	2
$e_3$	{5}	1

cuboids:  $A_1, A_2, A_3, A_1A_2, A_2A_3, A_1A_3, A_1A_2A_3$ . Furthermore, an inverted index is retained for each cell in the cuboids. That is, for each cell, its associated TID list is recorded.

The benefit of computing local cubes of each shell fragment instead of computing the complete cube shell can be seen by a simple calculation. For a base cuboid of 60 dimensions, there are only  $7 \times 20 = 140$  cuboids to be computed according to the preceding shell fragment partitioning. This is in contrast to the 36,050 cuboids computed for the cube shell of size 3! Notice that the above fragment partitioning is based simply on the grouping of consecutive dimensions. A more desirable approach would be to partition based on popular dimension groupings. This information can be obtained from domain experts or the past history of OLAP queries.

Let's return to our running example to see how shell fragments are computed.

**Example 3.16. Compute shell fragments.** Suppose we are to compute the shell fragments of size 3. We first divide the five dimensions into two fragments, namely  $(A, B, C)$  and  $(D, E)$ . For each fragment, we compute the full local data cube by intersecting the TID lists in Table 3.5 in a top-down depth-first order in the cuboid lattice. For example, to compute the cell  $(a_1, b_2, *)$ , we intersect the TID lists of  $a_1$  and  $b_2$  to obtain a new list of  $\{2, 3\}$ . Cuboid  $AB$  is shown in Table 3.6.

After computing cuboid  $AB$ , we can then compute cuboid  $ABC$  by intersecting all pairwise combinations between Table 3.6 and the row  $c_1$  in Table 3.5. Notice that because cell  $(a_2, b_2)$  is empty, it can be effectively discarded in subsequent computations, based on the downward antimonotonicity property. The same process can be applied to compute fragment  $(D, E)$ , which is completely independent from computing  $(A, B, C)$ . Cuboid  $DE$  is shown in Table 3.7.  $\square$

If the measure in the iceberg condition is `count()` (as in tuple counting), there is no need to reference the original database because the *length* of the TID list is equivalent to the tuple count. “*Do we need to reference the original database if computing other measures such as average()?*” Actually, we can build and reference an *ID\_measure* array instead, which stores what we need to compute other measures. For example, to compute `average()`, we let the *ID\_measure* array hold three elements, namely,  $(TID, \text{item\_count}, \text{sum})$ , for each cell. The `average()` measure for each aggregate cell can then be computed using `sum()/item_count()`, by accessing only this *ID\_measure* array. Since *ID\_measure* array is a

**Table 3.6 Cuboid  $AB$ .**

Cell	Intersection	TID List	List Size
$(a_1, b_1)$	$\{1, 2, 3\} \cap \{1, 4, 5\}$	$\{1\}$	1
$(a_1, b_2)$	$\{1, 2, 3\} \cap \{2, 3\}$	$\{2, 3\}$	2
$(a_2, b_1)$	$\{4, 5\} \cap \{1, 4, 5\}$	$\{4, 5\}$	2
$(a_2, b_2)$	$\{4, 5\} \cap \{2, 3\}$	$\{\}$	0

**Table 3.7 Cuboid  $DE$ .**

Cell	Intersection	TID List	List Size
$(d_1, e_1)$	$\{1, 3, 4, 5\} \cap \{1, 2\}$	$\{1\}$	1
$(d_1, e_2)$	$\{1, 3, 4, 5\} \cap \{3, 4\}$	$\{3, 4\}$	2
$(d_1, e_3)$	$\{1, 3, 4, 5\} \cap \{5\}$	$\{5\}$	1
$(d_2, e_1)$	$\{2\} \cap \{1, 2\}$	$\{2\}$	1

more compact data structure than the corresponding high-dimensional database, it is more likely to fit in memory.

“Once we have computed the shell fragments, how can they be used to answer OLAP queries?” Given the precomputed shell fragments, the cube space can be viewed as a virtual cube to support OLAP queries. In general, two types of queries are possible: (1) *point query* and (2) *subcube query*. In a point query, all of the *relevant* dimensions in the cube have been instantiated and only the corresponding measure is inquired, whereas in a subcube query, at least one of the *relevant* dimensions in the cube is *inquired*. Let’s examine only the subcube query. In an  $n$ -dimensional data cube  $A_1 A_2 \dots A_n$ , a subcube query could be in the form  $\langle A_1, A_5?, A_9, A_{21}? : M? \rangle$ , where  $A_1 = \{a_{11}, a_{18}\}$  and  $A_9 = a_{94}$ ,  $A_5$  and  $A_{21}$  are the inquired dimensions, and  $M$  is the inquired measure.

A subcube query returns a local data cube based on the instantiated and inquired dimensions. Such a data cube needs to be aggregated in a multidimensional way so that online analytical processing (drilling, dicing, pivoting, etc.) can be made available to users for flexible manipulation and analysis. Because instantiated dimensions usually provide highly selective constants that dramatically reduce the size of the valid TID lists, we should make maximal use of the precomputed shell fragments by finding the fragments that best fit the set of instantiated dimensions and fetching and intersecting the associated TID lists to derive the reduced TID list. This list can then be used to intersect the best-fitting shell fragments consisting of the inquired dimensions. This will generate the relevant and inquired base cuboid, which can then be used to compute the relevant subcube on-the-fly using an efficient online cubing algorithm.

Let the subcube query be of the form  $\langle \alpha_i, \alpha_j, A_k?, \alpha_p, A_q? : M? \rangle$ , where  $\alpha_i, \alpha_j$ , and  $\alpha_p$  represent a set of instantiated values of dimension  $A_i, A_j$ , and  $A_p$ , respectively, and  $A_k$  and  $A_q$  represent two inquired dimensions. First, we check the shell fragment schema to determine which dimensions among (1)  $A_i, A_j$ , and  $A_p$ , and (2)  $A_k$  and  $A_q$  are in the same fragment partition. Suppose  $A_i$  and  $A_j$  belong to the same fragment, as do  $A_k$  and  $A_q$ , but that  $A_p$  is in a different fragment. We fetch the corresponding TID lists in the precomputed 2-D fragment for  $A_i$  and  $A_j$  using the instantiations  $\alpha_i$  and  $\alpha_j$ , then fetch the TID list on the precomputed 1-D fragment for  $A_p$  using instantiation  $\alpha_p$ , and then fetch the TID lists on the precomputed 2-D fragments for  $A_k$  and  $A_q$ , respectively, using no instantiations (i.e., all possible values). The obtained TID lists are intersected to derive the final TID lists, which are used to fetch the corresponding measures from the *ID\_measure* array to derive the “base cuboid” of a 2-D subcube for two dimensions ( $A_k, A_q$ ). A fast cube computation algorithm can be applied to compute this 2-D cube based on the derived base cuboid. The computed 2-D cube is then ready for OLAP operations.

### 3.5.4 Efficient processing of OLAP queries using cuboids

The purpose of materializing cuboids and constructing OLAP index structures is to speed up query processing in data cubes. Given materialized views, query processing should proceed as follows:

- 1. Determine which operations should be performed on the available cuboids:** This involves transforming any selection, projection, roll-up (group-by), and drill-down operations specified in the query into corresponding SQL and/or OLAP operations. For example, slicing and dicing a data cube may correspond to selection and/or projection operations on a materialized cuboid.
- 2. Determine to which materialized cuboid(s) the relevant operations should be applied:** This involves identifying all of the materialized cuboids that may potentially be used to answer the query,

pruning the set using knowledge of “dominance” relationships among the cuboids, estimating the costs of using the remaining materialized cuboids, and selecting the cuboid with the least cost.

**Example 3.17. OLAP query processing.** Suppose that a data cube for a retail company is defined in the form of “*sales\_cube* [*time, item, location*]: *sum(sales\_in\_dollars)*.” The dimension hierarchies used are “*day < month < quarter < year*” for *time*; “*item\_name < brand < type*” for *item*; and “*street < city < province\_or\_state < country*” for *location*.

Suppose that the query to be processed is on {*brand, province\_or\_state*}, with the selection constant “*year = 2010*.” Also, suppose that there are four materialized cuboids available, as follows:

- cuboid 1: {*year, item\_name, city*}
- cuboid 2: {*year, brand, country*}
- cuboid 3: {*year, brand, province\_or\_state*}
- cuboid 4: {*item\_name, province\_or\_state*}, where *year = 2010*

“Which of these four cuboids should be selected to process the query?” Finer-granularity data cannot be generated from coarser-granularity data. Therefore cuboid 2 cannot be used because *country* is a more general concept than *province\_or\_state*. Cuboids 1, 3, and 4 can be used to process the query because (1) they have the same set or a superset of the dimensions in the query, (2) the selection clause in the query can imply the selection in the cuboid, and (3) the abstraction levels for the *item* and *location* dimensions in these cuboids are at a finer level than *brand* and *province\_or\_state*, respectively.

“How would the costs of each cuboid compare if used to process the query?” It is likely that using cuboid 1 would cost the most because both *item\_name* and *city* are at a lower level than the *brand* and *province\_or\_state* concepts specified in the query. If there are not many *year* values associated with *items* in the cube, but there are several *item\_names* for each *brand*, then cuboid 3 will be smaller than cuboid 4, and thus cuboid 3 should be chosen to process the query. However, if efficient indices are available for cuboid 4, then cuboid 4 may be a better choice. Therefore some cost-based estimation is required to decide which set of cuboids should be selected for query processing. □

---

## 3.6 Summary

- A **data warehouse** is a *subject-oriented, integrated, time-variant, and nonvolatile* data collection organized in support of management decision making. Several factors distinguish data warehouses from operational databases. Because the two systems provide quite different functionalities and require different kinds of data, it is necessary to maintain data warehouses separately from operational databases.
- Data warehouses often adopt a **three-tier architecture**. The bottom tier is a *warehouse database server*, which is typically a relational database system. The middle tier is an *OLAP server*, and the top tier is a *client* that contains query and reporting tools.
- A data warehouse contains **back-end tools and utilities** for populating and refreshing the warehouse. These cover data extraction, data cleaning, data transformation, loading, refreshing, and warehouse management.
- Data warehouse **metadata** are data defining the warehouse objects. A metadata repository provides details regarding the warehouse structure, data history, the algorithms used for summarization, map-

pings from the source data to the warehouse form, system performance, and business terms and issues.

- A **data lake** is a single repository of all enterprise data in the natural format. A data lake often stores both raw data copies and transformed data. Many analytical tasks can be conducted on data lakes. In enterprises and organizations, data warehouses and data lakes serve different purposes and complement with each other.
- The **layers of data storage** in data lakes include, from bottom up, the raw data layer, the optional standardized data layer, the cleansed data layer, the application data layer, and the optional sandbox data layer.
- A **multidimensional data model** is typically used for the design of corporate *data warehouses* and *departmental data marts*. Such a model can adopt a *star schema*, *snowflake schema*, or *fact constellation schema*. The core of the *multidimensional model* is the **data cube**, which consists of a large set of *facts* (or *measures*) and a number of *dimensions*. Dimensions are the entities or perspectives with respect to which an organization wants to keep records and are hierarchical in nature.
- A data cube consists of a **lattice of cuboids**, each corresponding to a different degree of summarization of the given multidimensional data.
- **Concept hierarchies** organize the values of attributes or dimensions into gradual abstraction levels. They are useful in mining at multiple abstraction levels.
- OLAP servers may adopt a **relational OLAP (ROLAP)**, a **multidimensional OLAP (MOLAP)**, or a **hybrid OLAP (HOLAP)** implementation. A ROLAP server uses an extended relational DBMS that maps OLAP operations on multidimensional data to standard relational operations. A MOLAP server maps multidimensional data views directly to array structures. A HOLAP server combines ROLAP and MOLAP. For example, it may use ROLAP for historic data while maintaining frequently accessed data in a separate MOLAP store.
- A **measure** in a data cube is a numeric function that can be evaluated at each point (i.e., cell) in the data cube space. Measures can be organized into three categories, namely **distributive**, **algebraic**, and **holistic**.
- **Online analytical processing** can be performed in data warehouses/marts using the multidimensional data model. Typical OLAP operations include *roll-up*, and *drill-(down, across, through)*, *slice-and-dice*, and *pivot (rotate)*, as well as statistical operations such as ranking and computing moving averages and growth rates. OLAP operations can be implemented efficiently using the data cube structure.
- To facilitate efficient data accessing, most data warehouse systems use index structures. **bimap index** represents a given attribute of low cardinality using bits and can substantially reduce the I/O cost and speed up the computation for many aggregate queries. **Join index** precomputes and stores identifier pairs of join results between a fact table and a dimension table, and thus can dramatically reduce I/O cost in aggregate computation.
- In many applications, a fact table may contain many attributes, but an OLAP query may only use several attributes. A **column-based database** stores the values of all records column by column instead of row by row and can save dramatic I/O cost and processing time in computing aggregates.
- A data cube consists of a **lattice of cuboids**. Each cuboid corresponds to a different degree of summarization of the given multidimensional data. **Full materialization** refers to the computation of all the cuboids in a data cube lattice. **Partial materialization** refers to the selective computation

of a subset of the cuboid cells in the lattice. Iceberg cubes and shell fragments are examples of partial materialization. A data cube may contain much redundant information. A **quotient cube** as a concise representation of data cube contains only closed cells and reduces redundant information. An **iceberg cube** is a data cube that stores only those cube cells that have an aggregate value (e.g., *count*) above some minimum support threshold. For **shell fragments** of a data cube, only some cuboids involving a small number of dimensions are computed, and queries on additional combinations of the dimensions can be computed on-the-fly.

- There are several efficient **data cube computation methods**. In this chapter, we discussed some cube computation methods in detail: (1) **MultiWay** array aggregation for materializing full data cubes in sparse-array-based, bottom-up, shared computation; (2) **BUC** for computing iceberg cubes by exploring ordering and sorting for efficient top-down computation; and (3) **shell-fragment cubing**, which supports high-dimensional OLAP by precomputing only the partitioned cube shell fragments.

---

### 3.7 Exercises

- 3.1. Consider the data about students, instructors, courses, and departments in a college setting. When such data is used as operational data, please give three example operations. If we want to build a data warehouse using such data, what may be a subject of the data warehouse?
- 3.2. Use one example to discuss how data mart, enterprise data warehouse, and machine learning applications can be connected and build up one over another.
- 3.3. Is it possible that an enterprise runs both a data warehouse and a data lake? If so, what is the relation between the data warehouse and the data lake? Can you describe one scenario where maintaining both a data warehouse and a data lake is necessary and beneficial?
- 3.4. Suppose that a data warehouse consists of the three dimensions *time*, *doctor*, and *patient*, and the two measures *count* and *charge*, where *charge* is the fee that a doctor charges a patient for a visit.
  - a. Enumerate three classes of schemas that are popularly used for modeling data warehouses.
  - b. Draw a schema diagram for the above data warehouse using one of the schema classes listed in (a).
  - c. Starting with the base cuboid [*day*, *doctor*, *patient*], what specific *OLAP operations* should be performed in order to list the total fee collected by each doctor in 2010?
  - d. To obtain the same list, write an SQL query assuming the data is stored in a relational database with the schema *fee* (*day*, *month*, *year*, *doctor*, *hospital*, *patient*, *count*, *charge*).
- 3.5. Suppose that a data warehouse for *Big\_University* consists of the four dimensions *student*, *course*, *semester*, and *instructor*, and two measures *count* and *avg\_grade*. At the lowest conceptual level (e.g., for a given student, course, semester, and instructor combination), the *avg\_grade* measure stores the actual course grade of the student. At higher conceptual levels, *avg\_grade* stores the average grade for the given combination.
  - a. Draw a *snowflake schema* diagram for the data warehouse.
  - b. Starting with the base cuboid [*student*, *course*, *semester*, *instructor*], what specific *OLAP operations* (e.g., roll-up from *semester* to *year*) should you perform in order to list the average grade of *CS* courses for each *Big\_University* student.

- c. If each dimension has five levels (including all), such as “*student < major < status < university < all*,” how many cuboids will this cube contain (including the base and apex cuboids)?
- 3.6. Suppose that a data warehouse consists of the four dimensions *date*, *spectator*, *location*, and *game*, and the two measures *count* and *charge*, where *charge* is the fare that a spectator pays when watching a game on a given date. Spectators may be students, adults, or seniors, with each category having its own charge rate.
- Draw a *star schema* diagram for the data warehouse.
  - Starting with the base cuboid [*date*, *spectator*, *location*, *game*], what specific *OLAP operations* should you perform in order to list the total charge paid by student spectators at *GM\_Place* in 2010?
  - Bitmap indexing* is useful in data warehousing. Taking this cube as an example, briefly discuss advantages and problems of using a bitmap index structure.
- 3.7. A data warehouse can be modeled by either a *star schema* or a *snowflake schema*. Briefly describe the similarities and the differences of the two models, and then analyze their advantages and disadvantages with regard to one another. Give your opinion of which might be more empirically useful and state the reasons behind your answer.
- 3.8. Design a data warehouse for a regional weather bureau. The weather bureau has about 1000 probes, which are scattered throughout various land and ocean locations in the region to collect basic weather data, including air pressure, temperature, and precipitation at each hour. All data are sent to the central station, which has collected such data for more than 10 years. Your design should facilitate efficient querying and online analytical processing and derive general weather patterns in multidimensional space.
- 3.9. A popular data warehouse implementation is to construct a multidimensional database, known as a data cube. Unfortunately, this may often generate a huge, yet very sparse, multidimensional matrix.
- Present an example illustrating such a huge and sparse data cube.
  - Design an implementation method that can elegantly overcome this sparse matrix problem. Note that you need to explain your data structures in detail and discuss the space needed, as well as how to retrieve data from your structures.
  - Modify your design in (b) to handle *incremental data updates*. Give the reasoning behind your new design.
- 3.10. Regarding the *computation of measures* in a data cube:
- Enumerate three categories of measures, based on the kind of aggregate functions used in computing a data cube.
  - For a data cube with the three dimensions *time*, *location*, and *item*, which category does the function *variance* belong to? Describe how to compute it if the cube is partitioned into many chunks.  
*Hint:* The formula for computing *variance* is  $\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x}_i)^2$ , where  $\bar{x}_i$  is the average of  $x_i$ s.
  - Suppose the function is “*top 10 sales*.” Discuss how to efficiently compute this measure in a data cube.
- 3.11. Suppose a company wants to design a data warehouse to facilitate the analysis of moving vehicles in an online analytical processing manner. The company registers huge amounts of auto

movement data in the format of (*Auto\_ID*, *location*, *speed*, *time*). Each *Auto\_ID* represents a vehicle associated with information (e.g., *vehicle\_category*, *driver\_category*), and each location may be associated with a street in a city. Assume that a street map is available for the city.

- a. Design such a data warehouse to facilitate effective online analytical processing in multidimensional space.
  - b. The movement data may contain noise. Discuss how you would develop a method to automatically discover data records that were likely erroneously registered in the data repository.
  - c. The movement data may be sparse. Discuss how you would develop a method that constructs a reliable data warehouse despite the sparsity of data.
  - d. If you want to drive from A to B starting at a particular time, discuss how a system may use the data in this warehouse to work out a fast route.
- 3.12.** Radio-frequency identification is commonly used to trace object movement and perform inventory control. An RFID reader can successfully read an RFID tag from a limited distance at any scheduled time. Suppose a company wants to design a data warehouse to facilitate the analysis of objects with RFID tags in an online analytical processing manner. The company registers huge amounts of RFID data in the format of (*RFID*, *at\_location*, *time*), and also has some information about the objects carrying the RFID tag, for example, (*RFID*, *product\_name*, *product\_category*, *producer*, *date\_produced*, *price*).
- a. Design a data warehouse to facilitate effective registration and online analytical processing of such data.
  - b. The RFID data may contain lots of redundant information. Discuss a method that maximally reduces redundancy during data registration in the RFID data warehouse.
  - c. The RFID data may contain lots of noise such as missing registration and misread IDs. Discuss a method that effectively cleans up the noisy data in the RFID data warehouse.
  - d. You may want to perform online analytical processing to determine how many TV sets were shipped from the LA seaport to BestBuy in Champaign, IL, by *month*, *brand*, and *price\_range*. Outline how this could be done efficiently if you were to store such RFID data in the warehouse.
  - e. If a customer returns a jug of milk and complains that it has spoiled before its expiration date, discuss how you can investigate such a case in the warehouse to find out what the problem is, either in shipping or in storage.
- 3.13.** In many applications, new data sets are incrementally added to the existing large data sets. Thus, an important consideration is whether a measure can be computed efficiently in an incremental manner. Use *count*, *standard deviation*, and *median* as examples to show that a distributive or algebraic measure facilitates efficient incremental computation, whereas a holistic measure does not.
- 3.14.** Suppose that we need to record three measures in a data cube: `min()`, `average()`, and `median()`. Design an efficient computation and storage method for each measure given that the cube allows data to be *deleted incrementally* (i.e., in small portions at a time) from the cube.
- 3.15.** In data warehouse technology, a multiple dimensional view can be implemented by a relational database technique (*ROLAP*), by a multidimensional database technique (*MOLAP*), or by a hybrid database technique (*HOLAP*).
- a. Briefly describe each implementation technique.
  - b. For each technique, explain how each of the following functions may be implemented:



- i. The generation of a data warehouse (including aggregation)
    - ii. Roll-up
    - iii. Drill-down
    - iv. Incremental updating
  - c. Which implementation techniques do you prefer, and why?
- 3.16.** Suppose that a data warehouse contains 20 dimensions, each with about five levels of granularity.
- a. Users are mainly interested in four particular dimensions, each having three frequently accessed levels for rolling up and drilling down. How would you design a data cube structure to support this preference efficiently?
  - b. At times, a user may want to *drill through* the cube to the raw data for one or two particular dimensions. How would you support this feature?
- 3.17.** A data cube,  $C$ , has  $n$  dimensions, and each dimension has exactly  $p$  distinct values in the base cuboid. Assume that there are no concept hierarchies associated with the dimensions.
- a. What is the *maximum number of cells* possible in the base cuboid?
  - b. What is the *minimum number of cells* possible in the base cuboid?
  - c. What is the *maximum number of cells* possible (including both base cells and aggregate cells) in the  $C$  data cube?
  - d. What is the *minimum number of cells* possible in  $C$ ?
- 3.18.** Assume that a 10-D base cuboid contains only three base cells: (1)  $(a_1, d_2, d_3, d_4, \dots, d_9, d_{10})$ , (2)  $(d_1, b_2, d_3, d_4, \dots, d_9, d_{10})$ , and (3)  $(d_1, d_2, c_3, d_4, \dots, d_9, d_{10})$ , where  $a_1 \neq d_1$ ,  $b_2 \neq d_2$ , and  $c_3 \neq d_3$ . The measure of the cube is  $\text{count}()$ .
- a. How many *nonempty* cuboids will a full data cube contain?
  - b. How many *nonempty* aggregate (i.e., nonbase) cells will a full cube contain?
  - c. How many *nonempty* aggregate cells will an iceberg cube contain if the condition of the iceberg cube is “ $\text{count} \geq 2$ ”?
  - d. A cell,  $c$ , is a *closed cell* if there exists no cell,  $d$ , such that  $d$  is a specialization of cell  $c$  (i.e.,  $d$  is obtained by replacing a  $*$  in  $c$  by a non- $*$  value) and  $d$  has the same measure value as  $c$ . A *closed cube* is a data cube consisting of only closed cells. How many closed cells are in the full cube?
- 3.19.** There are several typical cube computation methods, such as *MultiWay* [ZDN97], *BUC* [BR99], and *Star-Cubing* [XHLW03]. Briefly describe these three methods (i.e., use one or two lines to outline the key points) and compare their feasibility and performance under the following conditions:
- a. Computing a dense full cube of low dimensionality (e.g., less than eight dimensions).
  - b. Computing an iceberg cube of around 10 dimensions with a highly skewed data distribution.
  - c. Computing a sparse iceberg cube of high dimensionality (e.g., over 100 dimensions).
- 3.20.** Suppose a data cube,  $C$ , has  $D$  dimensions, and the base cuboid contains  $k$  distinct tuples.
- a. Present a formula to calculate the minimum number of cells that the cube,  $C$ , may contain.
  - b. Present a formula to calculate the maximum number of cells that  $C$  may contain.
  - c. Answer parts (a) and (b) as if the count in each cube cell must be no less than a threshold,  $v$ .
  - d. Answer parts (a) and (b) as if only closed cells are considered (with the minimum count threshold,  $v$ ).

- 3.21.** Suppose that a base cuboid has three dimensions,  $A, B, C$ , with the following number of cells:  $|A| = 1,000,000$ ,  $|B| = 100$ , and  $|C| = 1000$ . Suppose that each dimension is evenly partitioned into 10 portions for *chunking*.
- Assuming each dimension has only one level, draw the complete lattice of the cube.
  - If each cube cell stores one measure with four bytes, what is the total size of the computed cube if the cube is *dense*?
  - State the order for computing the chunks in the cube that requires the least amount of space, and compute the total amount of main memory space required for computing the 2-D planes.
- 3.22.** When computing a cube of high dimensionality, we encounter the inherent *curse of dimensionality* problem: There exists a huge number of subsets of combinations of dimensions.
- Suppose that there are only two base cells,  $\{(a_1, a_2, a_3, \dots, a_{100})$  and  $(a_1, a_2, b_3, \dots, b_{100})\}$ , in a 100-D base cuboid. Compute the number of nonempty aggregate cells. Comment on the storage space and time required to compute these cells.
  - Suppose we are to compute an iceberg cube from (a). If the minimum support count in the iceberg condition is 2, how many aggregate cells will there be in the iceberg cube? Show the cells.
  - Introducing iceberg cubes will lessen the burden of computing trivial aggregate cells in a data cube. However, even with iceberg cubes, we could still end up having to compute a large number of trivial uninteresting cells (i.e., with small counts). Suppose that a database has 20 tuples that map to (or cover) the two following base cells in a 100-D base cuboid, each with a cell count of 10:  $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10\}$ .
    - Let the minimum support be 10. How many distinct aggregate cells will there be like the following:  $\{(a_1, a_2, a_3, a_4, \dots, a_{99}, *) : 10, \dots, (a_1, a_2, *, a_4, \dots, a_{99}, a_{100}) : 10, \dots, (a_1, a_2, a_3, *, \dots, *, *) : 10\}$ ?
    - If we ignore all the aggregate cells that can be obtained by replacing some constants with  $*$ 's while keeping the same measure value, how many distinct cells remain? What are the cells?
- 3.23.** Propose an algorithm that computes *closed iceberg cubes* efficiently.
- 3.24.** Suppose that we want to compute an iceberg cube for the dimensions,  $A, B, C, D$ , where we wish to materialize all cells that satisfy a minimum support count of at least  $v$ , and where  $cardinality(A) < cardinality(B) < cardinality(C) < cardinality(D)$ . Show the *BUC* processing tree (which shows the order in which the *BUC* algorithm explores a data cube's lattice, starting from all) for the construction of this iceberg cube.
- 3.25.** Discuss how you might extend the *Star-Cubing* algorithm to compute iceberg cubes where the iceberg condition tests for an *avg* that is no bigger than some value,  $v$ .
- 3.26.** A flight data warehouse for a travel agent consists of six dimensions: *traveler*, *departure (city)*, *departure\_time*, *arrival*, *arrival\_time*, and *flight*; and two measures: *count()* and *avg\_fare()*, where *avg\_fare()* stores the concrete fare at the lowest level but the average fare at other levels.
- Suppose the cube is fully materialized. Starting with the *base cuboid* [*traveler*, *departure*, *departure\_time*, *arrival*, *arrival\_time*, *flight*], what specific OLAP operations (e.g., roll-up *flight* to *airline*) should one perform to list the average fare per month for *each business traveler* who flies American Airlines (AA) from Los Angeles in 2009?

- b. Suppose we want to compute a data cube where the condition is that the minimum number of records is 10 and the average fare is over \$500. Outline an efficient cube computation method (based on common sense about flight data distribution).
- 3.27. (Implementation project)** There are four typical data cube computation methods: MultiWay [ZDN97], BUC [BR99], H-Cubing [HPDW01], and Star-Cubing [XHLW03].
- a. Implement any one of these cube computation algorithms and describe your implementation, experimentation, and performance. Find another student who has implemented a different algorithm on the same platform (e.g., C++ on Linux) and compare your algorithm performance with his or hers.
- Input:
- i. An  $n$ -dimensional base cuboid table (for  $n < 20$ ), which is essentially a relational table with  $n$  attributes.
  - ii. An iceberg condition:  $\text{count}(C) \geq k$ , where  $k$  is a positive integer as a parameter.
- Output:
- i. The set of computed cuboids that satisfy the iceberg condition, in the order of your output generation.
  - ii. Summary of the set of cuboids in the form of “*cuboid ID*: the number of nonempty cells,” sorted in alphabetical order of cuboids (e.g.,  $A$ : 155,  $AB$ : 120,  $ABC$ : 22,  $ABCD$ : 4,  $ABCE$ : 6,  $ABD$ : 36), where the number after  $:$  represents the number of nonempty cells. (This is used to quickly check the correctness of your results.)
- b. Based on your implementation, discuss the following:
- i. What challenging computation problems are encountered as the number of dimensions grows large?
  - ii. How can iceberg cubing solve the problems of part (a) for some data sets (and characterize such data sets)?
  - iii. Give one simple example to show that sometimes iceberg cubes cannot provide a good solution.
- c. Instead of computing a high-dimensionality data cube, we may choose to materialize the cuboids that have only a small number of dimension combinations. For example, for a 30-D data cube, we may only compute the 5-D cuboids for every possible 5-D combination. The resulting cuboids form a *shell cube*. Discuss how easy or hard it is to modify your cube computation algorithm to facilitate such computation.
- 3.28.** The *sampling cube* was proposed for multidimensional analysis of sampling data (e.g., survey data). In many real applications, sampling data can be of high dimensionality (e.g., it is not unusual to have more than 50 dimensions in a survey data set).
- a. How can we construct an efficient and scalable high-dimensional sampling cube in large sampling data sets?
  - b. Design an efficient incremental update algorithm for such a high-dimensional sampling cube.
  - c. Discuss how to support quality drill-down given that some low-level cells may be empty or contain too few data for reliable analysis.
- 3.29.** The ranking cube was designed to support top- $k$  (ranking) queries in relational database systems. However, ranking queries are also posed to data warehouses, where ranking is on multidimensional aggregates instead of on measures of base facts. For example, consider a product manager

who is analyzing a sales database that stores the nationwide sales history, organized by location and time. To make investment decisions, the manager may pose the following query: “*What are the top-10 (state, year) cells having the largest total product sales?*” He may further drill down and ask, “*What are the top-10 (city, month) cells?*” Suppose the system can perform such partial materialization to derive two types of materialized cuboids: a *guiding cuboid* and a *supporting cuboid*, where the former contains a number of guiding cells that provide concise, high-level data statistics to guide the ranking query processing, whereas the latter provides inverted indices for efficient online aggregation.

- a. Derive an efficient method for computing such aggregate ranking cubes.
  - b. Extend your framework to handle more advanced measures. One such example could be as follows. Consider an organization donation database, where donors are grouped by “age,” “income,” and other attributes. Interesting questions include: “*Which age and income groups have made the top-k average amount of donation (per donor)?*” and “*Which income group of donors has the largest standard deviation in the donation amount?*”
- 3.30.** Recently, researchers have proposed another kind of query, called a *skyline query*. A *skyline query* returns all the objects  $p_i$  such that  $p_i$  is not dominated by any other object  $p_j$ , where dominance is defined as follows. Let the value of  $p_i$  on dimension  $d$  be  $v(p_i, d)$ . We say  $p_i$  is dominated by  $p_j$  if and only if for each preference dimension  $d$ ,  $v(p_j, d) \leq v(p_i, d)$ , and there is at least one  $d$  where the equality does not hold.
- a. Design a ranking cube (see the previous question) so that skyline queries can be processed efficiently.
  - b. Skyline queries are sometimes too strict to be desirable to some users. One may generalize the concept of skyline into *generalized skyline* as follows: *Given a  $d$ -dimensional database and a query  $q$ , the **generalized skyline** is the set of the following objects: (1) the skyline objects and (2) the nonskyline objects that are  $\epsilon$ -neighbors of a skyline object, where  $r$  is an  $\epsilon$ -neighbor of an object  $p$  if the distance between  $p$  and  $r$  is no more than  $\epsilon$ .* Design a ranking cube to process generalized skyline queries efficiently.
- 3.31.** The *prediction cube* is a good example of multidimensional data mining in cube space.
- a. Propose an efficient algorithm that computes prediction cubes in a given multidimensional database.
  - b. For what kind of classification models can your algorithm be applied? Explain.
- 3.32.** *Multifeature cubes* allow us to construct interesting data cubes based on rather sophisticated query conditions. Can you construct the following multifeature cube by translating the following user requests into queries using the form introduced in this textbook?
- a. Construct a smart shopper cube where a shopper is smart if at least 10% of the goods she buys in each shopping trip are on sale.
  - b. Construct a data cube for best-deal products where best-deal products are those products for which the price is the lowest for this product in the given month.
- 3.33.** *Discovery-driven cube exploration* is a desirable way to mark interesting points among a large number of cells in a data cube. Individual users may have different views on whether a point should be considered interesting enough to be marked. Suppose one would like to mark those objects of which the absolute value of  $z$  score is over 2 in every row and column in a  $d$ -dimensional plane.

- a. Derive an efficient computation method to identify such points during the data cube computation.
- b. Suppose a partially materialized cube has  $(d - 1)$ -dimensional and  $(d + 1)$ -dimensional cuboids materialized but not the  $d$ -dimensional one. Derive an efficient method to mark those  $(d - 1)$ -dimensional cells with  $d$ -dimensional children that contain such marked points.

---

### 3.8 Bibliographic notes

There are a good number of introductory-level textbooks on data warehousing and OLAP technology—for example, Kimball et al. [KRTM08]; Imhoff, Galemno, and Geiger [IGG03]; and Inmon [Inm96]. Chaudhuri and Dayal [CD97] provide an early overview of data warehousing and OLAP technology. A set of research papers on materialized views and data warehouse implementations are collected in *Materialized Views: Techniques, Implementations, and Applications* by Gupta and Mumick [GM99].

The history of decision support systems can be traced back to the 1960s. However, the proposal to construct large data warehouses for multidimensional data analysis is credited to Codd [CCS93] who coined the term *OLAP* for *online analytical processing*. The OLAP Council was established in 1995. Widom [Wid95] identifies several research problems in data warehousing. Kimball and Ross [KR02] provide an overview of the deficiencies of SQL regarding the ability to support comparisons that are common in the business world, and present a good set of application cases that require data warehousing and OLAP technology. For an overview of OLAP systems vs. statistical databases, see Shoshani [Sho97].

Gray et al. [GCB<sup>+</sup>97] propose the data cube as a relational aggregation operator generalizing group-by, crosstabs, and subtotals. Harinarayan, Rajaraman, and Ullman [HRU96] propose a greedy algorithm for the partial materialization of cuboids in the computation of a data cube. Data cube computation methods are investigated by numerous studies such as Sarawagi and Stonebraker [SS94]; Agarwal et al. [AAD<sup>+</sup>96]; Zhao, Deshpande, and Naughton [ZDN97]; Ross and Srivastava [RS97]; Beyer and Ramakrishnan [BR99]; Han, Pei, Dong, and Wang [HPDW01]; and Xin, Han, Li, and Wah [XHLW03].

The concept of iceberg queries is first introduced in Fang et al. [FSGM<sup>+</sup>98]. The use of join indices to speed up relational query processing is proposed by Valduriez [Val87]. O’Neil and Graefe [OG95] propose a bitmapped join index method to speed up OLAP-based query processing. A discussion of the performance of bitmapping and other nontraditional index techniques is given in O’Neil and Quass [OQ97].

For work regarding the selection of materialized cuboids for efficient OLAP query processing, see, for example, Chaudhuri and Dayal [CD97]; Harinarayan, Rajaraman, and Ullman [HRU96]; and Srivastava et al. [SDJL96]. Methods for cube size estimation are discussed by Deshpande et al. [DNR<sup>+</sup>97], Ross and Srivastava [RS97], and Beyer and Ramakrishnan [BR99]. Agrawal, Gupta, and Sarawagi [AGS97] propose operations for modeling multidimensional databases. Methods for answering queries quickly by online aggregation are described in Hellerstein, Haas, and Wang [HHW97] and Hellerstein et al. [HAC<sup>+</sup>99]. Techniques for estimating the top  $N$  queries are proposed in Carey and Kossman [CK98] and Donjerkovic and Ramakrishnan [DR99].

Efficient computation of multidimensional aggregates in data cubes is studied by many researchers. Gray et al. [GCB<sup>+</sup>97] propose *cube-by* as a relational aggregation operator generalizing group-by,

cross-tabs, and subtotals, and categorized data cube measures into three categories: *distributive*, *algebraic*, and *holistic*. Harinarayan, Rajaraman, and Ullman [HRU96] propose a greedy algorithm for the partial materialization of cuboids in the computation of a data cube. Sarawagi and Stonebraker [SS94] develop a chunk-based computation technique for the efficient organization of large multidimensional arrays. Agarwal et al. [AAD<sup>+</sup>96] propose several guidelines for efficient computation of multidimensional aggregates for ROLAP servers.

The chunk-based MultiWay array aggregation method for data cube computation in MOLAP is proposed by Zhao, Deshpande, and Naughton [ZDN97]. Ross and Srivastava [RS97] develop a method for computing sparse data cubes. Iceberg queries are first described in Fang et al. [FSGM<sup>+</sup>98]. BUC, a scalable method that computes iceberg cubes from the apex cuboid downwards, is introduced by Beyer and Ramakrishnan [BR99]. Han, Pei, Dong, and Wang [HPDW01] introduce an H-Cubing method for computing iceberg cubes with complex measures using an H-tree structure.

The Star-Cubing method for computing iceberg cubes with a dynamic star-tree structure is introduced by Xin, Han, Li, and Wah [XHLW03]. MM-Cubing, an efficient iceberg cube computation method that factorizes the lattice space is developed by Shao, Han, and Xin [SHX04]. The shell-fragment-based cubing approach for efficient high-dimensional OLAP is proposed by Li, Han, and Gonzalez [LHG04].

Aside from computing iceberg cubes, another way to reduce data cube computation is to materialize condensed, dwarf, or quotient cubes, which are variants of closed cubes. Wang, Feng, Lu, and Yu propose computing a reduced data cube, called a *condensed cube* [WLFY02]. Sismanis, Deligiannakis, Roussopoulos, and Kotidis propose computing a compressed data cube, called a *dwarf cube* [SDRK02]. Lakshmanan, Pei, and Han propose a *quotient cube* structure to summarize a data cube's semantics [LPH02], which is further extended to a *qc-tree structure* by Lakshmanan, Pei, and Zhao [LPZ03]. An *aggregation-based* approach, called C-Cubing (i.e., *Closed-Cubing*), is developed by Xin, Han, Shao, and Liu [XHSL06], which performs efficient closed-cube computation by taking advantage of a new algebraic measure *closedness*.

There are also various studies on the computation of compressed data cubes by approximation, such as *quasicubes* by Barbara and Sullivan [BS97]; *wavelet cubes* by Vitter, Wang, and Iyer [VWI98]; *compressed cubes* for query approximation on continuous dimensions by Shanmugasundaram, Fayyad, and Bradley [SFB99]; using log-linear models to compress data cubes by Barbara and Wu [BW00]; and OLAP over uncertain and imprecise data by Burdick et al. [BDJ<sup>+</sup>05].

For works regarding the selection of materialized cuboids for efficient OLAP query processing, see Chaudhuri and Dayal [CD97]; Harinarayan, Rajaraman, and Ullman [HRU96]; Srivastava, Dar, Jagadish, and Levy [SDJL96]; Gupta [Gup97]; Baralis, Paraboschi, and Teniente [BPT97]; and Shukla, Deshpande, and Naughton [SDN98]. Methods for cube size estimation are discussed by Deshpande et al. [DNR<sup>+</sup>97], Ross and Srivastava [RS97], and Beyer and Ramakrishnan [BR99]. Agrawal, Gupta, and Sarawagi [AGS97] propose operations for modeling multidimensional databases.

Data cube modeling and computation have been extended well beyond relational data. Computation of *stream cubes* for multidimensional stream data analysis is studied by Chen et al. [CDH<sup>+</sup>02]. Efficient computation of *spatial data cubes* is examined by Stefanovic, Han, and Koperski [SHK00], efficient OLAP in spatial data warehouses is studied by Papadias, Kalnis, Zhang, and Tao [PKZT01], and a map cube for visualizing spatial data warehouses is proposed by Shekhar et al. [SLT<sup>+</sup>01]. A multimedia data cube is constructed in MultiMediaMiner by Zaiane et al. [ZHL<sup>+</sup>98]. For analysis of multidimensional text databases, *TextCube*, based on the vector space model, is proposed by Lin et al. [LDH<sup>+</sup>08], and

*TopicCube*, based on a topic modeling approach, is proposed by Zhang, Zhai, and Han [ZZH09]. *RFID Cube* and *FlowCube* for analyzing RFID data are proposed by Gonzalez et al. [GHLK06,GHL06].

The *sampling cube* is introduced for analyzing sampling data by Li et al. [LHY<sup>+</sup>08]. The *ranking cube* is proposed by Xin, Han, Cheng, and Li [XHCL06] for efficient processing of ranking (top-*k*) queries in databases. This methodology has been extended by Wu, Xin, and Han [WXH08] to *ARCCube*, which supports the ranking of aggregate queries in partially materialized data cubes. It has also been extended by Wu, Xin, Mei, and Han [WXMH09] to *PromoCube*, which supports promotion query analysis in multidimensional space.

The discovery-driven exploration of OLAP data cubes is proposed by Sarawagi, Agrawal, and Megiddo [SAM98]. Further studies on integration of OLAP with data mining capabilities for intelligent exploration of multidimensional OLAP data are done by Sarawagi and Sathe [SS01]. The construction of multifeature data cubes is described by Ross, Srivastava, and Chatziantoniou [RSC98]. Methods for answering queries quickly by online aggregation are described by Hellerstein, Haas, and Wang [HHW97] and Hellerstein et al. [HAC<sup>+</sup>99]. A cube-gradient analysis problem, called *cubegrade*, is first proposed by Imielinski, Khachiyan, and Abdulghani [IKA02]. An efficient method for multidimensional constrained gradient analysis in data cubes is studied by Dong et al. [DHL<sup>+</sup>01].

Mining cube space, or integration of knowledge discovery and OLAP cubes, has been studied by many researchers. The concept of online analytical mining (OLAM), or OLAP mining, is introduced by Han [Han98]. Chen et al. develop a *regression cube* for regression-based multidimensional analysis of time-series data [CDH<sup>+</sup>02,CDH<sup>+</sup>06]. Fagin et al. [FGK<sup>+</sup>05] studied data mining in *multistructured databases*. Chen, Chen, Lin, and Ramakrishnan [CCLR05] propose *prediction cubes*, which integrate prediction models with data cubes to discover interesting data subspaces for facilitated prediction. Chen, Ramakrishnan, Shavlik, and Tamma [CRST06] study the use of data mining models as building blocks in a multistep mining process and the use of cube space to intuitively define the space of interest for predicting global aggregates from local regions. Ramakrishnan and Chen [RC07] present an organized picture of exploratory mining in cube space.

The innovation of data lake is mainly driven by industry. Some general introduction to the concept of data lakes include [Fan15]. Inmon [Inm16] discusses data lake architecture. Some examples of data lake systems include Azure Data Lake Store [RSD<sup>+</sup>17], Google Goods [HKN<sup>+</sup>16], Constance [HGQ16], and CoreDB [BBN<sup>+</sup>17].

# Pattern mining: basic concepts and methods

**Frequent patterns** are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a smartphone, then a smart TV, and then a smart home device, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

In this chapter, we introduce the basic concepts of frequent patterns, associations, and correlations (Section 4.1) and study how they can be mined efficiently (Section 4.2). We also discuss how to judge whether the patterns found are interesting (Section 4.3). In the subsequent chapter, we extend our discussion to advanced frequent pattern mining, including mining more complex forms of frequent patterns, and their applications.

## 4.1 Basic concepts

Frequent pattern mining uncovers recurring relationships in a given data set. This section introduces the basic concepts of frequent pattern mining for the discovery of interesting associations and correlations between itemsets in transactional and relational databases. We begin in Section 4.1.1 by presenting an example of market basket analysis, the earliest form of frequent pattern mining for association rules. The basic concepts of mining frequent patterns and associations are discussed in Section 4.1.2.

### 4.1.1 Market basket analysis: a motivating example

A set of items is referred to as an **itemset**.<sup>1</sup> Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts

<sup>1</sup> In the data mining research literature, “itemset” is more commonly used than “item set.”



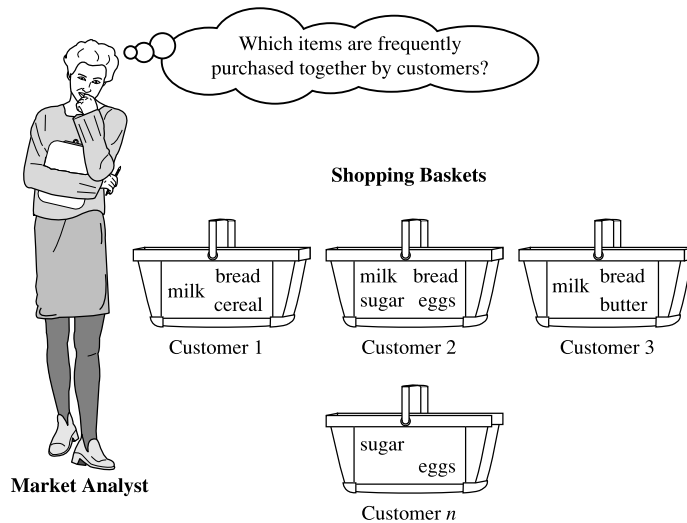


FIGURE 4.1

Market basket analysis.

of business transaction records can help in many business decision-making processes such as catalog design, cross-marketing, and customer shopping behavior analysis.

A typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by finding associations between the different items that customers place in their “shopping baskets” (Fig. 4.1). The discovery of these associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? This information can lead to increased sales, revenue, and customer acquisition by helping retailers do selective marketing and planned shelf space.

Let’s look at an example of how market basket analysis can be useful.

**Example 4.1. Market basket analysis.** Suppose, as manager of a retail company, you would like to learn more about the buying habits of your customers. Specifically, you wonder, “Which groups or sets of items are customers likely to purchase on a given trip to the store?” To answer your question, market basket analysis may be performed on the retail data of customer transactions at your store. You can then use these results to choose marketing strategies and help create a new catalog. For instance, market basket analysis may help you design different store layouts. In one strategy, items that are frequently purchased together can be placed in proximity to further encourage the combined sale of such items. If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of both items.

In an alternative strategy, placing hardware and software at opposite ends of the store may entice customers who purchase such items to pick up other items along the way. For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading toward the software display to purchase antivirus software and may decide to purchase a home security system as

well. Market basket analysis can also help retailers plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then reducing the prices on printers may encourage the sale of printers *as well as* computers.  $\square$

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed to extract buying patterns that reflect items that are frequently *associated* or purchased together. These patterns can be represented in the form of **association rules**. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in the following association rule:

$$\text{computer} \Rightarrow \text{antivirus\_software} \quad [\text{support} = 2\%, \text{confidence} = 60\%]. \quad (4.1)$$

Rule **support** and **confidence** are two measures of rule interestingness. They reflect the usefulness and certainty of discovered rules, respectively. A support of 2% for Rule (4.1) means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy a **minimum support threshold** and a **minimum confidence threshold**. These thresholds can be set by users or domain experts. Additional analysis can be performed to discover interesting statistical correlations between associated items.

### 4.1.2 Frequent itemsets, closed itemsets, and association rules

Let  $\mathcal{I} = \{I_1, I_2, \dots, I_m\}$  be an itemset. Let  $D$ , the task-relevant data, be a set of database transactions where each transaction  $T$  is a nonempty itemset such that  $T \subseteq \mathcal{I}$ . Each transaction is associated with an identifier, called a *TID*. Let  $A$  be a set of items. A transaction  $T$  is said to contain  $A$  if  $A \subseteq T$ . An association rule is an implication of the form  $A \Rightarrow B$ , where  $A \subset \mathcal{I}$ ,  $B \subset \mathcal{I}$ ,  $A \neq \emptyset$ ,  $B \neq \emptyset$ , and  $A \cap B = \emptyset$ . The rule  $A \Rightarrow B$  holds in the transaction set  $D$  with **support**  $s$ , where  $s$  is the percentage of transactions in  $D$  that contain  $A \cup B$  (i.e., the *union* of sets  $A$  and  $B$  say, or, both  $A$  and  $B$ ). This is taken to be the probability,  $P(A \cup B)$ .<sup>2</sup> The rule  $A \Rightarrow B$  has **confidence**  $c$  in the transaction set  $D$ , where  $c$  is the percentage of transactions in  $D$  containing  $A$  that also contain  $B$ . This is taken to be the conditional probability,  $P(B|A)$ . That is,

$$\text{support}(A \Rightarrow B) = P(A \cup B) \quad (4.2)$$

$$\text{confidence}(A \Rightarrow B) = P(B|A). \quad (4.3)$$

Rules that satisfy both a minimum support threshold (*min\_sup*) and a minimum confidence threshold (*min\_conf*) are called **strong**. By convention, support and confidence values are represented as percentages.

---

<sup>2</sup> Notice that the notation  $P(A \cup B)$  indicates the probability that a transaction contains the *union* of sets  $A$  and  $B$  (i.e., it contains every item in  $A$  and  $B$ ). This should not be confused with  $P(A \text{ or } B)$ , which indicates the probability that a transaction contains either  $A$  or  $B$ .

An itemset that contains  $k$  items is a  **$k$ -itemset**. The set  $\{\text{computer}, \text{antivirus\_software}\}$  is a 2-itemset. The **occurrence frequency of an itemset** is the number of transactions that contain the itemset. Occurrence frequency is also referred as the **frequency**, **support count**, or **count** of the itemset. Note that the itemset support defined in Eq. (4.2) is sometimes referred to as *relative support*, whereas the occurrence frequency is called the **absolute support**. If the relative support of an itemset  $I$  satisfies a prespecified **minimum support threshold** (i.e., the absolute support of  $I$  satisfies the corresponding **minimum support count threshold**), then  $I$  is a **frequent** itemset.<sup>3</sup> The set of frequent  $k$ -itemsets is commonly denoted by  $L_k$ .<sup>4</sup>

From Eq. (4.3), we have

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}. \quad (4.4)$$

Eq. (4.4) shows that the confidence of rule  $A \Rightarrow B$  can be easily derived from the support counts of  $A$  and  $A \cup B$ . That is, once the support counts of  $A$ ,  $B$ , and  $A \cup B$  are found, it is straightforward to derive the corresponding association rules  $A \Rightarrow B$  and  $B \Rightarrow A$  and check whether they are strong. Thus the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. **Find all frequent itemsets.** By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min\_sup*.
2. **Generate strong association rules from the frequent itemsets.** By definition, these rules must satisfy minimum support and minimum confidence.

Additional interestingness measures that can be applied for the discovery of correlation relationships between associated items will be discussed in Section 4.3. The overall performance of mining association rules is determined by the first step since the second step is much less costly than the first.

A major challenge in mining frequent itemsets from a large data set is the fact that such mining often generates a huge number of itemsets satisfying the minimum support (*min\_sup*) threshold, especially when *min\_sup* is set low. This is because if an itemset is frequent, each of its subsets is frequent as well. A long itemset will contain a combinatorial number of shorter frequent subitemsets. For example, a frequent itemset of length 100, such as  $\{a_1, a_2, \dots, a_{100}\}$ , contains  $\binom{100}{1} = 100$  frequent 1-itemsets:  $\{a_1\}, \{a_2\}, \dots, \{a_{100}\}$ ;  $\binom{100}{2}$  frequent 2-itemsets:  $\{a_1, a_2\}, \{a_1, a_3\}, \{a_1, a_4\}, \dots, \{a_2, a_3\}, \{a_2, a_4\}, \dots, \{a_{99}, a_{100}\}$ ; and so on. The total number of frequent itemsets that it contains is thus

$$\binom{100}{1} + \binom{100}{2} + \dots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \quad (4.5)$$

This is too huge a number of itemsets for any computer to compute or store. To overcome this difficulty, we introduce the concepts of *closed frequent itemset* and *maximal frequent itemset*.

<sup>3</sup> In early work, itemsets satisfying minimum support were referred to as **large**. This term, however, is somewhat confusing as it has connotations of the number of items in an itemset rather than the frequency of occurrence of the set. Hence, we use the more recent term **frequent**.

<sup>4</sup> Although the term **frequent** is preferred over **large**, for historic reasons frequent  $k$ -itemsets are still denoted as  $L_k$ .

An itemset  $X$  is **closed** in a data set  $D$  if there exists no proper superitemset  $Y^5$  such that  $Y$  has the same support count as  $X$  in  $D$ . An itemset  $X$  is a **closed frequent itemset** in set  $D$  if  $X$  is both closed and frequent in  $D$ . An itemset  $X$  is a **maximal frequent itemset** (or **max-itemset**) in a data set  $D$  if  $X$  is frequent, and there exists no superitemset  $Y$  such that  $X \subset Y$  and  $Y$  is frequent in  $D$ .

Let  $\mathcal{C}$  be the set of closed frequent itemsets for a data set  $D$  satisfying a minimum support threshold,  $min\_sup$ . Let  $\mathcal{M}$  be the set of maximal frequent itemsets for  $D$  satisfying  $min\_sup$ . Suppose that we have the support count of each itemset in  $\mathcal{C}$  and  $\mathcal{M}$ . Notice that  $\mathcal{C}$  and its count information can be used to derive the whole set of frequent itemsets. Thus we say that  $\mathcal{C}$  contains complete information regarding its corresponding frequent itemsets. On the other hand,  $\mathcal{M}$  registers only the support of the maximal itemsets. It usually does not contain the complete support information regarding its corresponding frequent itemsets. We illustrate these concepts with Example 4.2.

**Example 4.2. Closed and maximal frequent itemsets.** Suppose that a transaction database has only two transactions:  $\{\langle a_1, a_2, \dots, a_{100} \rangle; \langle a_1, a_2, \dots, a_{50} \rangle\}$ . Let the minimum support count threshold be  $min\_sup = 1$ . We find two closed frequent itemsets and their support counts, that is,  $\mathcal{C} = \{\{a_1, a_2, \dots, a_{100}\} : 1; \{a_1, a_2, \dots, a_{50}\} : 2\}$ . There is only one maximal frequent itemset:  $\mathcal{M} = \{\{a_1, a_2, \dots, a_{100}\} : 1\}$ . Notice that we cannot include  $\{a_1, a_2, \dots, a_{50}\}$  as a maximal frequent itemset because it has a frequent superset,  $\{a_1, a_2, \dots, a_{100}\}$ . Compare this to the preceding where we determined that there are  $2^{100} - 1$  frequent itemsets, which are too many to be enumerated!

The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from  $\mathcal{C}$ , we can derive, say, (1)  $\{a_2, a_{45} : 2\}$  since  $\{a_2, a_{45}\}$  is a subitemset of the itemset  $\{a_1, a_2, \dots, a_{50} : 2\}$ ; and (2)  $\{a_8, a_{55} : 1\}$  since  $\{a_8, a_{55}\}$  is not a subitemset of the previous itemset but of the itemset  $\{a_1, a_2, \dots, a_{100} : 1\}$ . However, from the maximal frequent itemset, we can only assert that both itemsets ( $\{a_2, a_{45}\}$  and  $\{a_8, a_{55}\}$ ) are frequent, but we cannot assert their actual support counts.  $\square$

---

## 4.2 Frequent itemset mining methods

In this section, you will learn methods for mining the simplest form of frequent patterns such as those discussed for market basket analysis in Section 4.1.1. We begin by presenting **Apriori**, the basic algorithm for finding frequent itemsets in Section 4.2.1. In Section 4.2.2, we look at how to generate strong association rules from frequent itemsets. Section 4.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability. Section 4.2.4 presents pattern-growth methods for mining frequent itemsets that confine the subsequent search space to only the data sets containing the current frequent itemsets. Section 4.2.5 presents methods for mining frequent itemsets that take advantage of the vertical data format.

---

<sup>5</sup>  $Y$  is a proper superitemset of  $X$  if  $X$  is a proper subitemset of  $Y$ , that is, if  $X \subset Y$ . In other words, every item of  $X$  is contained in  $Y$ , but there is at least one item of  $Y$  that is not in  $X$ .

### 4.2.1 Apriori algorithm: finding frequent itemsets by confined candidate generation

**Apriori** is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules [AS94b]. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where  $k$ -itemsets are used to explore  $(k + 1)$ -itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by  $L_1$ . Next,  $L_1$  is used to find  $L_2$ , the set of frequent 2-itemsets, which is used to find  $L_3$ , and so on, until no more frequent  $k$ -itemsets can be found. The finding of each  $L_k$  requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the **Apriori property** is used to reduce the search space.

**Apriori property:** *all nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset  $I$  does not satisfy the minimum support threshold,  $min\_sup$ , then  $I$  is not frequent, that is,  $P(I) < min\_sup$ . If an item  $A$  is added to the itemset  $I$ , then the resulting itemset (i.e.,  $I \cup A$ ) cannot occur more frequently than  $I$ . Therefore  $I \cup A$  is not frequent either, that is,  $P(I \cup A) < min\_sup$ .

This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotonicity* because the property is monotonic in the context of failing a test.

“How is the Apriori property used in the algorithm?” To understand this, let us look at how  $L_{k-1}$  is used to find  $L_k$  for  $k \geq 2$ . A two-step process is followed, consisting of **join** and **prune** actions.

- 1. The join step.** To find  $L_k$ , a set of **candidate**  $k$ -itemsets is generated by joining  $L_{k-1}$  with itself. This set of candidates is denoted  $C_k$ . Let  $l_1$  and  $l_2$  be itemsets in  $L_{k-1}$ . The notation  $l_i[j]$  refers to the  $j$ th item in  $l_i$  (e.g.,  $l_1[k - 2]$  refers to the second to the last item in  $l_1$ ). For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the  $(k - 1)$ -itemset,  $l_i$ , this means that the items are sorted such that  $l_i[1] < l_i[2] < \dots < l_i[k - 1]$ . The join,  $L_{k-1} \bowtie L_{k-1}$ , is performed, where members of  $L_{k-1}$  are joinable if their first  $(k - 2)$  items are in common. That is, members  $l_1$  and  $l_2$  of  $L_{k-1}$  are joined if  $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \dots \wedge (l_1[k - 2] = l_2[k - 2]) \wedge (l_1[k - 1] < l_2[k - 1])$ . The condition  $l_1[k - 1] < l_2[k - 1]$  simply ensures that no duplicates are generated. The resulting itemset formed by joining  $l_1$  and  $l_2$  is  $\{l_1[1], l_1[2], \dots, l_1[k - 2], l_1[k - 1], l_2[k - 1]\}$ .
- 2. The prune step.**  $C_k$  is a superset of  $L_k$ , that is, its members may or may not be frequent, but all of the frequent  $k$ -itemsets are included in  $C_k$ . A database scan to determine the count of each candidate in  $C_k$  would result in the determination of  $L_k$  (i.e., all candidates having a count no less than the minimum support count are frequent by definition and therefore belong to  $L_k$ ).  $C_k$ , however, can be huge, and so this could involve heavy computation. To reduce the size of  $C_k$ , the Apriori property is used as follows. Any  $(k - 1)$ -itemset that is not frequent cannot be a subset of a frequent  $k$ -itemset. Hence, if any  $(k - 1)$ -subset of a candidate  $k$ -itemset is not in  $L_{k-1}$ , then the candidate cannot be frequent either and so can be removed from  $C_k$ . This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

**Example 4.3. Apriori.** Let's look at a concrete example, based on the transaction database,  $D$ , of Table 4.1. There are nine transactions in this database, that is,  $|D| = 9$ . We use Fig. 4.2 to illustrate the Apriori algorithm for finding frequent itemsets in  $D$ .

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets,  $C_1$ . The algorithm simply scans all of the transactions to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is,  $min\_sup = 2$ . (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is  $2/9 = 22\%$ .) The set of frequent 1-itemsets,  $L_1$ , can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in  $C_1$  satisfy minimum support.
3. To discover the set of frequent 2-itemsets,  $L_2$ , the algorithm uses the join  $L_1 \bowtie L_1$  to generate a candidate set of 2-itemsets,  $C_2$ .<sup>6</sup>  $C_2$  consists of  $\binom{L_1}{2}$  2-itemsets. Note that no candidates are removed from  $C_2$  during the prune step because each subset of the candidates is also frequent.
4. Next, the transactions in  $D$  are scanned and the support count of each candidate itemset in  $C_2$  is accumulated, as shown in the middle table of the second row in Fig. 4.2.
5. The set of frequent 2-itemsets,  $L_2$ , is then determined, consisting of those candidate 2-itemsets in  $C_2$  having minimum support.
6. The generation of the set of the candidate 3-itemsets,  $C_3$ , is detailed in Fig. 4.3. From the join step, we first get  $C_3 = L_2 \bowtie L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$ . Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from  $C_3$ , thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of  $D$  to determine  $L_3$ . Note that when given a candidate  $k$ -itemset, we only need to check if its  $(k - 1)$ -subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of  $C_3$  is shown in the first table of the bottom row of Fig. 4.2.
7. The transactions in  $D$  are scanned to determine  $L_3$ , consisting of those candidate 3-itemsets in  $C_3$  having minimum support (Fig. 4.2).

**Table 4.1 A transactional data set.**

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

<sup>6</sup>  $L_1 \bowtie L_1$  is equivalent to  $L_1 \times L_1$ , since the definition of  $L_k \bowtie L_k$  requires the two joining itemsets to share  $k - 1 = 0$  items.

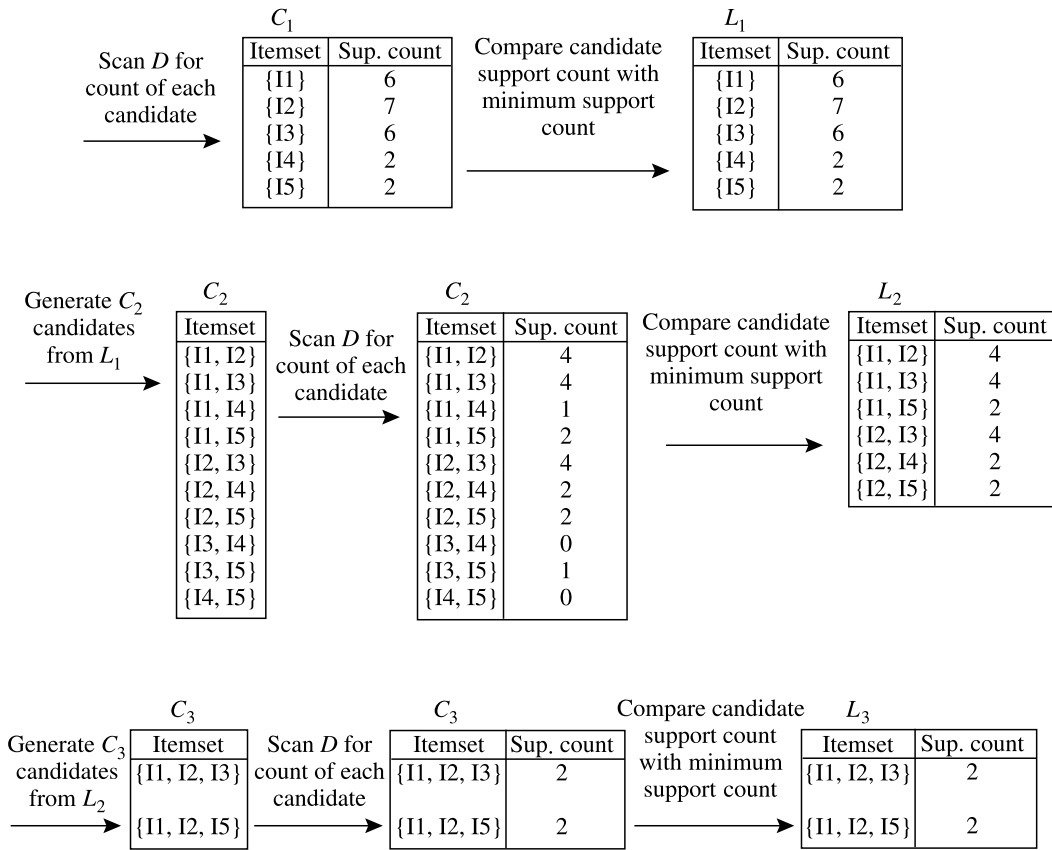


FIGURE 4.2

Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

8. The algorithm uses  $L_3 \bowtie L_3$  to generate a candidate set of 4-itemsets,  $C_4$ . Although the join results in  $\{\{I1, I2, I3, I5\}\}$ , itemset  $\{I1, I2, I3, I5\}$  is pruned because its subset  $\{I2, I3, I5\}$  is not frequent. Thus,  $C_4 = \phi$ , and the algorithm terminates, having found all of the frequent itemsets.  $\square$

Fig. 4.4 shows pseudocode for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets,  $L_1$ . In steps 2 through 10,  $L_{k-1}$  is used to generate candidates  $C_k$  to find  $L_k$  for  $k \geq 2$ . The `apriori_gen` procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3). Once all of the candidates have been generated, the database is scanned (step 4). For each transaction, a subset function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6 and 7). Finally, all the candidates satisfying the minimum support (step 9) form the set of frequent itemsets,  $L$  (step 11). A procedure can then be called to generate association rules from the frequent itemsets. Such a procedure is described in Section 4.2.2.

- a. Join:  $C_3 = L_2 \times L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$   
 $\times \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$   
 $= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}.$
- b. Prune using the Apriori property: *all nonempty subsets of a frequent itemset must also be frequent*. Do any of the candidates have a subset that is not frequent?
- The 2-item subsets of  $\{I1, I2, I3\}$  are  $\{I1, I2\}$ ,  $\{I1, I3\}$ , and  $\{I2, I3\}$ . All 2-item subsets of  $\{I1, I2, I3\}$  are members of  $L_2$ . Therefore, keep  $\{I1, I2, I3\}$  in  $C_3$ .
  - The 2-item subsets of  $\{I1, I2, I5\}$  are  $\{I1, I2\}$ ,  $\{I1, I5\}$ , and  $\{I2, I5\}$ . All 2-item subsets of  $\{I1, I2, I5\}$  are members of  $L_2$ . Therefore, keep  $\{I1, I2, I5\}$  in  $C_3$ .
  - The 2-item subsets of  $\{I1, I3, I5\}$  are  $\{I1, I3\}$ ,  $\{I1, I5\}$ , and  $\{I3, I5\}$ .  $\{I3, I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I1, I3, I5\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2, I3, I4\}$  are  $\{I2, I3\}$ ,  $\{I2, I4\}$ , and  $\{I3, I4\}$ .  $\{I3, I4\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2, I3, I4\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2, I3, I5\}$  are  $\{I2, I3\}$ ,  $\{I2, I5\}$ , and  $\{I3, I5\}$ .  $\{I3, I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2, I3, I5\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2, I4, I5\}$  are  $\{I2, I4\}$ ,  $\{I2, I5\}$ , and  $\{I4, I5\}$ .  $\{I4, I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2, I4, I5\}$  from  $C_3$ .
- c. Therefore,  $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$  after pruning.

FIGURE 4.3

Generation and pruning of candidate 3-itemsets,  $C_3$ , from  $L_2$  using the Apriori property.

The `apriori_gen` procedure performs two kinds of actions, namely, **join** and **prune**, as described before. In the join component,  $L_{k-1}$  is joined with  $L_{k-1}$  to generate potential candidates (steps 1–4). The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure `has_infrequent_subset`.

## 4.2.2 Generating association rules from frequent itemsets

Once the frequent itemsets from transactions in a database  $D$  have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum confidence). This can be done using Eq. (4.4) for confidence, which we show again here for completeness:

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}.$$

The conditional probability is expressed in terms of itemset support count, where  $\text{support\_count}(A \cup B)$  is the number of transactions containing the itemsets  $A \cup B$ , and  $\text{support\_count}(A)$  is the number of transactions containing the itemset  $A$ . Based on this equation, association rules can be generated as follows.

- For each frequent itemset  $l$ , generate all nonempty subsets of  $l$ .
- For every nonempty subset  $s$  of  $l$ , output the rule “ $s \Rightarrow (l - s)$ ” if  $\frac{\text{support\_count}(l)}{\text{support\_count}(s)} \geq \text{min\_conf}$ , where  $\text{min\_conf}$  is the minimum confidence threshold.



**Algorithm: Apriori.** Find frequent itemsets using an iterative level-wise approach based on candidate generation.

**Input:**

- $D$ , a database of transactions;
- $min\_sup$ , the minimum support count threshold.

**Output:**  $L$ , frequent itemsets in  $D$ .

**Method:**

```

(1)   $L_1 = \text{find\_frequent\_1-itemsets}(D)$ ;
(2)  for ( $k = 2$ ;  $L_{k-1} \neq \phi$ ;  $k++$ ) {
(3)     $C_k = \text{apriori\_gen}(L_{k-1})$ ;
(4)    for each transaction  $t \in D$  { // scan  $D$  for counts
(5)       $C_t = \text{subset}(C_k, t)$ ; // get the subsets of  $t$  that are candidates
(6)      for each candidate  $c \in C_t$ 
(7)         $c.\text{count}++$ ;
(8)    }
(9)     $L_k = \{c \in C_k | c.\text{count} \geq min\_sup\}$ 
(10) }
(11) return  $L = \cup_k L_k$ ;

procedure apriori_gen( $L_{k-1}$ :frequent ( $k - 1$ )-itemsets)
(1)  for each itemset  $l_1 \in L_{k-1}$ 
(2)    for each itemset  $l_2 \in L_{k-1}$ 
(3)      if ( $l_1[1] = l_2[1] \wedge (l_1[2] = l_2[2])$ 
           $\wedge \dots \wedge (l_1[k - 2] = l_2[k - 2]) \wedge (l_1[k - 1] < l_2[k - 1])$ ) then {
(4)         $c = l_1 \bowtie l_2$ ; // join step: generate candidates
(5)        if has_infrequent_subset( $c, L_{k-1}$ ) then
(6)          delete  $c$ ; // prune step: remove unfruitful candidate
(7)        else add  $c$  to  $C_k$ ;
(8)      }
(9)  return  $C_k$ ;

procedure has_infrequent_subset( $c$ : candidate  $k$ -itemset;
                                $L_{k-1}$ : frequent ( $k - 1$ )-itemsets); // use prior knowledge
(1)  for each ( $k - 1$ )-subset  $s$  of  $c$ 
(2)    if  $s \notin L_{k-1}$  then
(3)      return TRUE;
(4)  return FALSE;

```

**FIGURE 4.4**

Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

**Example 4.4. Generating association rules.** Let's try an example based on the transactional data shown before in Table 4.1. The data contain frequent itemset  $X = \{I1, I2, I5\}$ . What are the association rules that can be generated from  $X$ ? The nonempty subsets of  $X$  are  $\{I1, I2\}$ ,  $\{I1, I5\}$ ,  $\{I2, I5\}$ ,  $\{I1\}$ ,  $\{I2\}$ , and  $\{I5\}$ . The resulting association rules are as shown below, each listed with its confidence:

- $\{I1, I2\} \Rightarrow I5, \text{ confidence} = 2/4 = 50\%$
- $\{I1, I5\} \Rightarrow I2, \text{ confidence} = 2/2 = 100\%$
- $\{I2, I5\} \Rightarrow I1, \text{ confidence} = 2/2 = 100\%$
- $I1 \Rightarrow \{I2, I5\}, \text{ confidence} = 2/6 = 33\%$
- $I2 \Rightarrow \{I1, I5\}, \text{ confidence} = 2/7 = 29\%$
- $I5 \Rightarrow \{I1, I2\}, \text{ confidence} = 2/2 = 100\%$

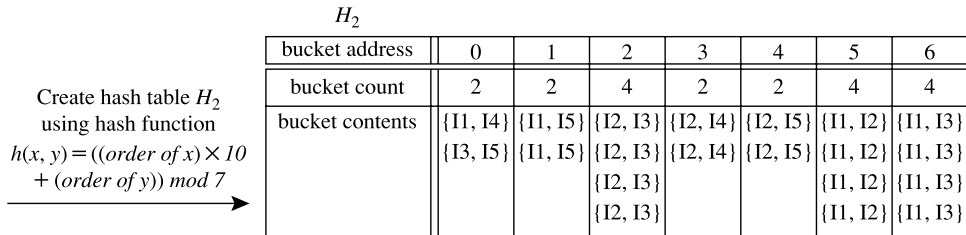
If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are strong. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right side of the rule. □

### 4.2.3 Improving the efficiency of Apriori

“How can we further improve the efficiency of Apriori-based mining?” Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized as follows.

**Hash-based technique** (hashing itemsets into corresponding buckets). A hash-based technique can be used to reduce the size of the candidate  $k$ -itemsets,  $C_k$ , for  $k > 1$ . For example, when scanning each transaction (e.g., let  $t = \{i_1, i_2, i_4\}$ ) in the database to generate the frequent 1-itemsets,  $L_1$ , we can generate all the 2-itemsets for each transaction (e.g., three 2-itemsets  $\{i_1, i_2\}$ ,  $\{i_1, i_4\}$ , and  $\{i_2, i_4\}$  for transaction  $t$ ), hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts as shown in Fig. 4.5. A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of candidate  $k$ -itemsets examined (especially when  $k = 2$ ).

**Transaction reduction** (reducing the number of transactions scanned in future iterations). A transaction that does not contain any frequent  $k$ -itemsets cannot contain any frequent  $(k + 1)$ -itemsets. Therefore such a transaction can be marked or removed from further consideration because subsequent database scans for  $j$ -itemsets, where  $j > k$ , will not need to consider such a transaction.



**FIGURE 4.5**

Hash table,  $H_2$ , for candidate 2-itemsets. This hash table was generated by scanning Table 4.1’s transactions while determining  $L_1$ . If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in  $C_2$ .

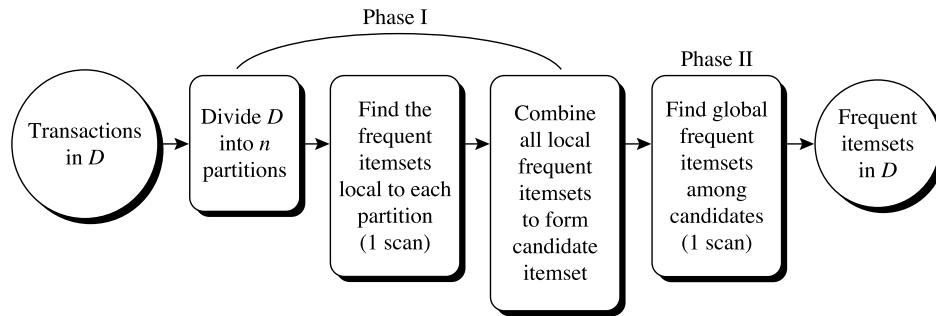


FIGURE 4.6

Mining by partitioning the data.

**Partitioning** (partitioning the data to find candidate itemsets). A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Fig. 4.6). It consists of two phases. In phase I, the algorithm divides the transactions of  $D$  into  $n$  nonoverlapping partitions. If the minimum relative support threshold for transactions in  $D$  is  $min\_sup$ , then the minimum support count for a partition is  $min\_sup \times \text{the number of transactions in that partition}$ . For each partition, all the *local frequent itemsets* (i.e., the itemsets frequent within the partition) are found. A local frequent itemset may or may not be frequent with respect to the entire database,  $D$ . However, *any itemset that is potentially frequent with respect to  $D$  must occur as a frequent itemset in at least one of the partitions.*<sup>7</sup> Therefore all local frequent itemsets are candidate itemsets with respect to  $D$ . The collection of frequent itemsets from all partitions forms the *global candidate itemsets* with respect to  $D$ . In phase II, a second scan of  $D$  is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

**Sampling** (mining on a subset of the given data). The basic idea of the sampling approach is to pick a random sample  $S$  of the given data  $D$ , and then search for frequent itemsets in  $S$  instead of  $D$ . In this way, we trade off some degree of accuracy against efficiency. The  $S$  sample size is such that the search for frequent itemsets in  $S$  can be done in main memory, and so only one scan of the transactions in  $S$  is required overall. Because we are searching for frequent itemsets in  $S$  rather than in  $D$ , it is possible that we will miss some of the global frequent itemsets.

To reduce this possibility, we use a lower support threshold than the minimum support to find the frequent itemsets local to  $S$  (denoted  $L_S$ ). The rest of the database is then used to compute the actual frequencies of each itemset in  $L_S$ . A mechanism is used to determine whether all the global frequent itemsets are included in  $L_S$ . If  $L_S$  actually contains all the frequent itemsets in  $D$ , then only one scan of  $D$  is required. Otherwise, a second pass can be done to find the frequent itemsets that were missed in the first pass. The sampling approach is especially beneficial when efficiency is of utmost importance such as in computationally intensive applications that must be run frequently.

<sup>7</sup> The proof of this property is left as an exercise (see Exercise 4.3d).

**Dynamic itemset counting** (adding candidate itemsets at different points during a scan). A dynamic itemset counting technique is proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only after each complete database scan. The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Other variations are discussed in the next chapter or left as exercises.

#### 4.2.4 A pattern-growth approach for mining frequent itemsets

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs.

- *It may still need to generate a huge number of candidate sets.* For example, if there are  $10^4$  frequent 1-itemsets, the Apriori algorithm will need to generate more than  $10^7$  candidate 2-itemsets.
- *It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching.* It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

“Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?” An interesting method in this attempt is called **frequent pattern growth**, or simply **FP-growth**, which adopts a *divide-and-conquer* strategy as follows. First, it compresses the database representing frequent items into a **frequent pattern tree**, or **FP-tree**, which retains the itemset association information. It then divides the compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one itemset found so far, or “pattern fragment,” and mines each database separately. For each “pattern fragment,” only its associated data sets need to be examined. Therefore this approach may substantially reduce the size of the data sets to be searched, along with the “growth” of patterns being examined. You will see how it works in Example 4.5.

**Example 4.5. FP-growth (finding frequent itemsets without candidate generation).** We reexamine the mining of transaction database,  $D$ , of Table 4.1 in Example 4.3 using the frequent pattern growth approach.

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by  $L$ . Thus, we have  $L = \{\{I2: 7\}, \{I1: 6\}, \{I3: 6\}, \{I4: 2\}, \{I5: 2\}\}$ .

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with “null.” Scan database  $D$  a second time. The items in each transaction are processed in  $L$  order (i.e., sorted according to descending support count), and a branch is created for each transaction. For example, the scan of the first transaction, “T100: I1, I2, I5,” which contains three items (I2, I1, I5 in  $L$  order), leads to the construction of the first branch of the tree with three nodes,  $\langle I2: 1 \rangle$ ,  $\langle I1: 1 \rangle$ , and  $\langle I5: 1 \rangle$ , where I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, T200,

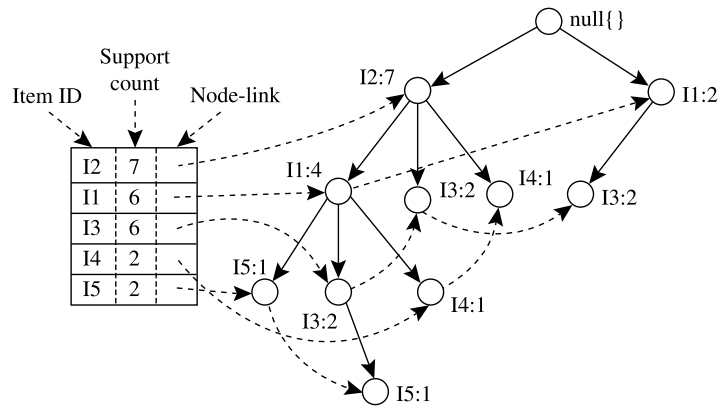


FIGURE 4.7

An FP-tree registers compressed frequent pattern information.

contains the items I2 and I4 in  $L$  order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common **prefix**, I2, with the existing path for T100. Therefore, we instead increment the count of the I2 node by 1, and create a new node,  $\langle I4: 1 \rangle$ , which is linked as a child to  $\langle I2: 2 \rangle$ . In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links**. The tree obtained after scanning all the transactions is shown in Fig. 4.7 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed into that of mining the FP-tree.

The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a “subdatabase,” which consists of the set of *prefix paths* in the FP-tree cooccurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining of the FP-tree is summarized in Table 4.2 and detailed as follows.

- We first consider I5, which is the last item in  $L$ , rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two FP-tree branches of Fig. 4.7. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are  $\langle I2, I1, I5: 1 \rangle$  and  $\langle I2, I1, I3, I5: 1 \rangle$ . Therefore, considering I5 as a suffix, its corresponding two prefix paths are  $\langle I2, I1: 1 \rangle$  and  $\langle I2, I1, I3: 1 \rangle$ , which form its conditional pattern base. Using this conditional pattern base as a transaction database, we build an I5-conditional FP-tree, which contains only a single path,  $\langle I2: 2, I1: 2 \rangle$ ; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns:  $\{I2, I5: 2\}$ ,  $\{I1, I5: 2\}$ ,  $\{I2, I1, I5: 2\}$ .

**Table 4.2 Mining the FP-tree by creating conditional (sub-)pattern bases.**

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	(I2: 2, I1: 2)	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	(I2: 2)	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	(I2: 4, I1: 2), (I1: 2)	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	(I2: 4)	{I2, I1: 4}

**Algorithm: FP\_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

**Input:**

- $D$ , a transaction database;
- $min\_sup$ , the minimum support count threshold.

**Output:** The complete set of frequent patterns.

**Method:**

1. The FP-tree is constructed in the following steps:
  - a. Scan the transaction database  $D$  once. Collect  $F$ , the set of frequent items, and their support counts. Sort  $F$  in support count descending order as  $L$ , the list of frequent items.
  - b. Create the root of an FP-tree, and label it as “null.” For each transaction  $Trans$  in  $D$  do the following. Select and sort the frequent items in  $Trans$  according to the order of  $L$ . Let the sorted frequent item list in  $Trans$  be  $[p|P]$ , where  $p$  is the first element and  $P$  is the remaining list. Call  $insert\_tree([p|P], T)$ , which is performed as follows. If  $T$  has a child  $N$  such that  $N.item-name = p.item-name$ , then increment  $N$ 's count by 1; else create a new node  $N$ , and let its count be 1, its parent link be linked to  $T$ , and its node-link to the nodes with the same  $item-name$  via the node-link structure. If  $P$  is nonempty, call  $insert\_tree(P, N)$  recursively.
2. The FP-tree is mined by calling  $FP\_growth(FP\_tree, null)$ , which is implemented as follows.

procedure  $FP\_growth(Tree, \alpha)$

- (1) **if**  $Tree$  contains a single path  $P$  **then**
- (2)     **for each** combination (denoted as  $\beta$ ) of the nodes in the path  $P$
- (3)         generate pattern  $\beta \cup \alpha$  with  $support\_count = minimum\ support\ count\ of\ nodes\ in\ \beta$ ;
- (4) **else for each**  $a_i$  in the header of  $Tree$  **{**
- (5)     generate pattern  $\beta = a_i \cup \alpha$  with  $support\_count = a_i.support\_count$ ;
- (6)     construct  $\beta$ 's conditional pattern base and then  $\beta$ 's conditional FP-tree  $Tree_\beta$ ;
- (7)     **if**  $Tree_\beta \neq \emptyset$  **then**
- (8)         call  $FP\_growth(Tree_\beta, \beta)$ ; **}**

#### FIGURE 4.8

FP-growth algorithm for discovering frequent itemsets without candidate generation.

- For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, (I2: 2), and derives one frequent pattern, {I2, I4: 2}.
- Similar to the preceding analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}. Its conditional FP-tree has two branches, (I2: 4, I1: 2) and (I1: 2), as shown in Fig. 4.9, which generates the set of patterns {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}.
- Finally, I1's conditional pattern base is {{I2: 4}}, with an FP-tree that contains only one node, (I2: 4), which generates one frequent pattern, {I2, I1: 4}.

This mining process is summarized in Fig. 4.8. □

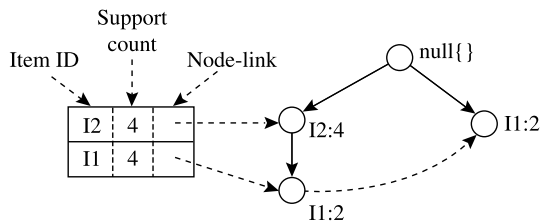


FIGURE 4.9

The conditional FP-tree associated with the conditional node I3.

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first partition the database into a set of projected databases and then construct an FP-tree and mine it in each projected database. This process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

#### 4.2.5 Mining frequent itemsets using the vertical data format

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (i.e.,  $\{TID : itemset\}$ ), where *TID* is a transaction ID and *itemset* is the set of items bought in transaction *TID*. This is known as the **horizontal data format**. Alternatively, data can be presented in *item-TID\_set* format (i.e.,  $\{item : TID\_set\}$ ), where *item* is an item name, and *TID\_set* is the set of transaction identifiers containing the item. This is known as the **vertical data format**.

In this subsection, we look at how frequent itemsets can also be mined efficiently using vertical data format, which is the essence of the **Eclat** (Equivalence Class Transformation) algorithm.

**Example 4.6. Mining frequent itemsets using the vertical data format.** Consider the horizontal data format of the transaction database, *D*, of Table 4.1 in Example 4.3. This can be transformed into the vertical data format shown in Table 4.3 by scanning the data set once.

**Table 4.3 The vertical data format of the transaction data set *D* of Table 4.1.**

itemset	TID_set
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

**Table 4.4 2-Itemsets in vertical data format.**

itemset	TID_set
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

**Table 4.5 3-Itemsets in vertical data format.**

itemset	TID_set
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

Mining can be performed on this data set by intersecting the TID\_sets of every pair of frequent single items. The minimum support count is 2. Because every single item is frequent in Table 4.3, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table 4.4. Notice that because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}. By intersecting the TID\_sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table 4.5, where there are only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}. □

Example 4.6 illustrates the process of mining frequent itemsets by exploring the vertical data format. First, we transform the horizontally formatted data into the vertical format by scanning the data set once. The support count of an itemset is simply the length of the TID\_set of the itemset. Starting with  $k = 1$ , the frequent  $k$ -itemsets can be used to construct the candidate  $(k + 1)$ -itemsets based on the Apriori property. The computation is done by intersection of the TID\_sets of the frequent  $k$ -itemsets to compute the TID\_sets of the corresponding  $(k + 1)$ -itemsets. This process repeats, with  $k$  incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate  $(k + 1)$ -itemset from frequent  $k$ -itemsets, another merit of this method is that there is no need to scan the database to find the support of  $(k + 1)$ -itemsets (for  $k \geq 1$ ). This is because the TID\_set of each  $k$ -itemset carries the complete information required for counting such support. However, the TID\_sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.

To further reduce the cost of registering long TID\_sets, as well as the subsequent costs of intersections, we can use a technique called *diffset*, which keeps track of only the differences of the TID\_sets of



a  $(k + 1)$ -itemset and a corresponding  $k$ -itemset. For instance, in Example 4.6 we have  $\{I1\} = \{T100, T400, T500, T700, T800, T900\}$  and  $\{I1, I2\} = \{T100, T400, T800, T900\}$ . The *diffset* between the two is  $\text{diffset}(\{I1, I2\}, \{I1\}) = \{T500, T700\}$ . Thus rather than recording the four TIDs that make up the intersection of  $\{I1\}$  and  $\{I2\}$ , we can instead use *diffset* to record just two TIDs, indicating the difference between  $\{I1\}$  and  $\{I1, I2\}$ . With such compressed bookkeeping, itemset frequency can still be calculated correctly. Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

### 4.2.6 Mining closed and max patterns

In Section 4.1.2 we saw how frequent itemset mining may generate a huge number of frequent itemsets, especially when the *min\_sup* threshold is set low or when there exist long patterns in the data set. Example 4.2 showed that closed frequent itemsets<sup>8</sup> can substantially reduce the number of patterns generated in frequent itemset mining while preserving the complete information regarding the set of frequent itemsets. That is, from the set of closed frequent itemsets, we can easily derive the set of frequent itemsets and their support. Thus in practice, it is more desirable to mine the set of closed frequent itemsets rather than the set of all frequent itemsets in most cases.

“How can we mine closed frequent itemsets?” A naïve approach would be to first mine the complete set of frequent itemsets and then remove every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset. However, this is quite costly. As shown in Example 4.2, this method would have to first derive  $2^{100} - 1$  frequent itemsets to obtain a length-100 frequent itemset, all before it could begin to eliminate redundant itemsets. This is prohibitively expensive. In fact, there exist only a very small number of closed frequent itemsets in Example 4.2’s data set.

A recommended methodology is to prune the search space as soon as we can identify the case of closed itemsets during mining. For example, an itemset merging method is introduced as follows.

**Itemset merging.** *If every transaction containing a frequent itemset  $X$  also contains an itemset  $Y$  but not any proper superset of  $Y$ , then  $X \cup Y$  forms a frequent closed itemset and there is no need to search for any itemset containing  $X$  but no  $Y$ .*

For example, in Table 4.2 of Example 4.5, the projected conditional database for prefix itemset  $\{I5:2\}$  is  $\{\{I2, I1\}, \{I2, I1, I3\}\}$ , from which we can see that each of its transactions contains itemset  $\{I2, I1\}$  but no proper superset of  $\{I2, I1\}$ . Itemset  $\{I2, I1\}$  can be merged with  $\{I5\}$  to form the closed itemset  $\{I5, I2, I1: 2\}$ , and we do not need to mine for closed itemsets that contain  $I5$  but not  $\{I2, I1\}$ .

Many search space pruning and closure checking methods have been developed for mining frequent closed itemsets. Moreover, because maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed for mining closed itemset can be extended to mining maximal frequent itemsets. Interested readers may like to dig deeper by studying related research papers.

---

<sup>8</sup> Remember that  $X$  is a *closed frequent* itemset in a data set  $S$  if there exists no proper superitemset  $Y$  such that  $Y$  has the same support count as  $X$  in  $S$ , and  $X$  satisfies minimum support.

## 4.3 Which patterns are interesting?—Pattern evaluation methods

Most association rule mining algorithms employ a support–confidence framework. Although minimum support and confidence thresholds *help* weed out or exclude the exploration of a good number of uninteresting rules, many of the rules generated are still not interesting to many users. This is especially true *when mining at low support thresholds or mining for long patterns*. This has been a major bottleneck for successful application of association rule mining.

In this section, we first look at how even strong association rules can be uninteresting and misleading (Section 4.3.1). We then discuss how the support–confidence framework can be supplemented with additional interestingness measures based on *correlation analysis* (Section 4.3.2). Section 4.3.3 presents additional pattern evaluation measures. It then provides an overall comparison of all the measures discussed here. By the end, you will learn which pattern evaluation measures are most effective for the discovery of only interesting rules.

### 4.3.1 Strong rules are not necessarily interesting

The interestingness of a rule can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics “behind” the data, can be used as one step toward the goal of weeding out uninteresting rules that would otherwise be presented to the user.

“*How can we tell which strong association rules are really interesting?*” Let’s examine the following example.

**Example 4.7. A misleading “strong” association rule.** Suppose we are interested in analyzing transactions with respect to the purchase of computer games and videos. Let *game* refer to the transactions containing computer games, and *video* refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, whereas 7500 included videos, and 4000 included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered:

$$\begin{aligned} & \text{buys}(X, \text{“computer games”}) \Rightarrow \text{buys}(X, \text{“videos”}) \\ & [\text{support} = 40\%, \text{confidence} = 66\%]. \end{aligned} \quad (4.6)$$

Rule (4.6) is a strong association rule and would therefore be reported, since its support value of  $\frac{4000}{10,000} = 40\%$  and confidence value of  $\frac{4000}{6000} = 66\%$  satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (4.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (4.6).  $\square$

Example 4.7 also illustrates that the confidence of a rule  $A \Rightarrow B$  can be deceiving. It does not measure the *real strength* (or lack of strength) of the *correlation* and *implication* between  $A$  and  $B$ . Hence, alternatives to the support–confidence framework can be useful in mining interesting data relationships.

### 4.3.2 From association analysis to correlation analysis

As we have seen so far, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be augmented to the support–confidence framework for association rules. This leads to *correlation rules* of the form

$$A \Rightarrow B \text{ [support, confidence, correlation]}. \quad (4.7)$$

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets  $A$  and  $B$ . There are many different correlation measures for us to choose. In this subsection, we study several correlation measures to determine which would be good for mining large data sets.

**Lift** is a simple correlation measure that is given as follows. The occurrence of itemset  $A$  is **independent** of the occurrence of itemset  $B$  if  $P(A \cup B) = P(A)P(B)$ ; otherwise, itemsets  $A$  and  $B$  are **dependent** and **correlated**. This definition can easily be extended to more than two itemsets. The **lift** between the occurrence of  $A$  and  $B$  can be measured by computing

$$\text{lift}(A, B) = \frac{P(A \cup B)}{P(A)P(B)}. \quad (4.8)$$

If the resulting value of Eq. (4.8) is less than 1, then the occurrence of  $A$  is *negatively correlated* with the occurrence of  $B$ , meaning that the occurrence of one likely leads to the absence of the other one. If the resulting value is greater than 1, then  $A$  and  $B$  are *positively correlated*, meaning that the occurrence of one implies the occurrence of the other. If the resulting value is equal to 1, then  $A$  and  $B$  are *independent*, and there is no correlation between them.

Eq. (4.8) is equivalent to  $P(B|A)/P(B)$ , or  $\text{conf}(A \Rightarrow B)/P(B)$ , which is also referred to as the *lift* of the association (or correlation) rule  $A \Rightarrow B$ . In other words, it assesses the degree to which the occurrence of one “lifts” the occurrence of the other. For example, if  $A$  corresponds to the sale of computer games and  $B$  corresponds to the sale of videos, then given the current market conditions, the sale of games is said to increase or “lift” the likelihood of the sale of videos by a factor of the value returned by Eq. (4.8).

Let’s go back to the computer game and video data of Example 4.7.

**Example 4.8. Correlation analysis using lift.** To help filter out misleading “strong” associations of the form  $A \Rightarrow B$  from the data of Example 4.7, we need to study how the two itemsets,  $A$  and  $B$ , are correlated. Let *game* refer to the transactions of Example 4.7 that do not contain computer games, and *video* refer to those that do not contain videos. The transactions can be summarized in a *contingency table*, as shown in Table 4.6.

From the table, we can see that the probability of purchasing a computer game is  $P(\{\text{game}\}) = 0.60$ , the probability of purchasing a video is  $P(\{\text{video}\}) = 0.75$ , and the probability of purchasing both is  $P(\{\text{game}, \text{video}\}) = 0.40$ . By Eq. (4.8), the lift of Rule (4.6) is  $P(\{\text{game}, \text{video}\})/(P(\{\text{game}\}) \times P(\{\text{video}\})) = 0.40/(0.60 \times 0.75) = 0.89$ . Because this value is less than 1, there is a negative correlation between the occurrence of  $\{\text{game}\}$  and  $\{\text{video}\}$ . The numerator is the likelihood of a customer purchasing both, whereas the denominator is what the likelihood would have been if the two purchases were completely independent. Such a negative correlation cannot be identified by a support–confidence framework.  $\square$

**Table 4.6** 2 × 2 contingency table summarizing the transactions with respect to game and video purchases.

	<i>game</i>	$\overline{\text{game}}$	$\Sigma_{row}$
<i>video</i>	4000	3500	7500
$\overline{\text{video}}$	2000	500	2500
$\Sigma_{col}$	6000	4000	10,000

**Table 4.7** Table 4.6 contingency table, now with the expected values.

	<i>game</i>	$\overline{\text{game}}$	$\Sigma_{row}$
<i>video</i>	4000 (4500)	3500 (3000)	7500
$\overline{\text{video}}$	2000 (1500)	500 (1000)	2500
$\Sigma_{col}$	6000	4000	10,000

The second correlation measure that we study is the  $\chi^2$  measure, which was introduced in Chapter 3 (Eq. (3.1)). To compute the  $\chi^2$  value, we take the squared difference between the observed and expected value for a slot ( $A$  and  $B$  pair) in the contingency table, divided by the expected value. This amount is summed for all slots of the contingency table. Let's perform a  $\chi^2$  analysis of Example 4.8.

**Example 4.9. Correlation analysis using  $\chi^2$ .** To compute the correlation using  $\chi^2$  analysis for nominal data, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 4.7. From the table, we can compute the  $\chi^2$  value as follows:

$$\begin{aligned} \chi^2 = \Sigma \frac{(\text{observed} - \text{expected})^2}{\text{expected}} &= \frac{(4000 - 4500)^2}{4500} + \frac{(3500 - 3000)^2}{3000} \\ &+ \frac{(2000 - 1500)^2}{1500} + \frac{(500 - 1000)^2}{1000} = 555.6. \end{aligned}$$

Because the  $\chi^2$  value is greater than 1, and the observed value of the slot ( $\text{game}, \text{video}$ ) = 4000, which is less than the expected value of 4500, *buying game* and *buying video* are *negatively correlated*. This is consistent with the conclusion derived from the analysis of the *lift* measure in Example 4.8.  $\square$

### 4.3.3 A comparison of pattern evaluation measures

The above discussion shows that instead of using the simple support–confidence framework to evaluate frequent patterns, other measures, such as *lift* and  $\chi^2$ , often disclose more intrinsic pattern relationships. How effective are these measures? Should we also consider other alternatives?

Researchers have studied many pattern evaluation measures even before the start of in-depth research on scalable methods for mining frequent patterns. In the data mining community, several other pattern evaluation measures have attracted interest. In this subsection, we present four such measures: *all\_confidence*, *max\_confidence*, *Kulczynski*, and *cosine*. Each of these four measures has an interesting property: the value of each measure is only influenced by the supports of  $A$ ,  $B$ , and  $A \cup B$ , or more

exactly, by the conditional probabilities of  $P(A|B)$  and  $P(B|A)$ , but not by the total number of transactions. Another common property is that each measure ranges from 0 to 1, and the higher the value, the closer the relationship between  $A$  and  $B$ .

Given two itemsets,  $A$  and  $B$ , the **all\_confidence** measure of  $A$  and  $B$  is defined as

$$all\_conf(A, B) = \frac{sup(A \cup B)}{\max\{sup(A), sup(B)\}} = \min\{P(A|B), P(B|A)\}, \quad (4.9)$$

where  $\max\{sup(A), sup(B)\}$  is the maximum support of the itemsets  $A$  and  $B$ . Thus  $all\_conf(A, B)$  is also the minimum confidence of the two association rules related to  $A$  and  $B$ , namely, “ $A \Rightarrow B$ ” and “ $B \Rightarrow A$ .”

Given two itemsets,  $A$  and  $B$ , the **max\_confidence** measure of  $A$  and  $B$  is defined as

$$max\_conf(A, B) = \max\{P(A|B), P(B|A)\}. \quad (4.10)$$

The  $max\_conf$  measure is the maximum confidence of the two association rules, “ $A \Rightarrow B$ ” and “ $B \Rightarrow A$ .”

Given two itemsets,  $A$  and  $B$ , the **Kulczynski** measure of  $A$  and  $B$  (abbreviated as **Kulc**) is defined as

$$Kulc(A, B) = \frac{1}{2}(P(A|B) + P(B|A)). \quad (4.11)$$

It was proposed in 1927 by Polish mathematician S. Kulczynski. It can be viewed as an average of two confidence measures. That is, it is the average of two conditional probabilities: the probability of itemset  $B$  given itemset  $A$ , and the probability of itemset  $A$  given itemset  $B$ .

Finally, given two itemsets,  $A$  and  $B$ , the **cosine** measure of  $A$  and  $B$  is defined as

$$\begin{aligned} cosine(A, B) &= \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}} \\ &= \sqrt{P(A|B) \times P(B|A)}. \end{aligned} \quad (4.12)$$

The *cosine* measure can be viewed as a *harmonized lift* measure. The two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of  $A$  and  $B$ . This is an important difference, however, because by taking the square root, the cosine value is only influenced by the supports of  $A$ ,  $B$ , and  $A \cup B$ , and not by the total number of transactions.

Now, together with *lift* and  $\chi^2$ , we have introduced in total six pattern evaluation measures. You may wonder, “Which is the best in assessing the discovered pattern relationships?” To answer this question, we examine their performance on some typical data sets.

**Example 4.10. Comparison of six pattern evaluation measures on typical data sets.** The relationships between the purchases of two items, *milk* and *coffee*, can be examined by summarizing their purchase history in Table 4.8, a  $2 \times 2$  contingency table, where an entry such as *mc* represents the number of transactions containing both milk and coffee.

**Table 4.8**  $2 \times 2$  contingency table for two items.

	<i>milk</i>	$\overline{milk}$	$\Sigma_{row}$
<i>coffee</i>	<i>mc</i>	$\overline{mc}$	<i>c</i>
$\overline{coffee}$	$m\overline{c}$	$\overline{m\overline{c}}$	$\overline{c}$
$\Sigma_{col}$	<i>m</i>	$\overline{m}$	$\Sigma$

**Table 4.9** Comparison of six pattern evaluation measures using contingency tables for a variety of data sets.

Data Set	<i>mc</i>	$\overline{mc}$	$m\overline{c}$	$\overline{m\overline{c}}$	$\chi^2$	<i>lift</i>	<i>all_conf.</i>	<i>max_conf.</i>	<i>Kulc.</i>	<i>cosine</i>
$D_1$	10,000	1000	1000	100,000	90,557	9.26	0.91	0.91	0.91	0.91
$D_2$	10,000	1000	1000	100	0	1	0.91	0.91	0.91	0.91
$D_3$	100	1000	1000	100,000	670	8.44	0.09	0.09	0.09	0.09
$D_4$	1000	1000	1000	100,000	24,740	25.75	0.5	0.5	0.5	0.5
$D_5$	1000	100	10,000	100,000	8173	9.18	0.09	0.91	0.5	0.29
$D_6$	1000	10	100,000	100,000	965	1.97	0.01	0.99	0.5	0.10

Table 4.9 shows a set of transactional data sets with their corresponding contingency tables and the associated values for each of the six evaluation measures. Let’s first examine the first four data sets,  $D_1$  through  $D_4$ . From the table, we see that *m* and *c* are positively associated in  $D_1$  and  $D_2$ , negatively associated in  $D_3$ , and neutral in  $D_4$ . For  $D_1$  and  $D_2$ , *m* and *c* are positively associated because *mc* (10,000) is considerably greater than  $\overline{mc}$  (1000) and  $m\overline{c}$  (1000). Intuitively, for people who bought milk ( $m = 10,000 + 1000 = 11,000$ ), it is very likely that they also bought coffee ( $mc/m = 10/11 = 91\%$ ), and vice versa.

The results of the four newly introduced measures show that *m* and *c* are strongly positively associated in both data sets by producing a measure value of 0.91. However, *lift* and  $\chi^2$  generate dramatically different measure values for  $D_1$  and  $D_2$  due to their sensitivity to  $\overline{m\overline{c}}$ . In fact, in many real-world scenarios,  $\overline{m\overline{c}}$  is usually huge and unstable. For example, in a market basket database, the total number of transactions could fluctuate on a daily basis and overwhelmingly exceed the number of transactions containing any particular itemset. Therefore a good interestingness measure should not be affected by transactions that do not contain the itemsets of interest; otherwise, it would generate unstable results, as illustrated in  $D_1$  and  $D_2$ .

Similarly, in  $D_3$ , the four new measures correctly show that *m* and *c* are strongly negatively associated because the *mc* to *c* ratio equals the *mc* to *m* ratio, that is,  $100/1100 = 9.1\%$ . However, *lift* and  $\chi^2$  both contradict this in an incorrect way: their values for  $D_2$  are between those for  $D_1$  and  $D_3$ .

For data set  $D_4$ , both *lift* and  $\chi^2$  indicate a highly positive association between *m* and *c*, whereas the others indicate a “neutral” association because the ratio of *mc* to  $\overline{mc}$  equals the ratio of *mc* to  $m\overline{c}$ , which is 1. This means that if a customer buys coffee (or milk), the probability that he or she will also purchase milk (or coffee) is exactly 50%. □

“Why are *lift* and  $\chi^2$  so poor at distinguishing pattern association relationships in the previous transactional data sets?” To answer this, we have to consider the *null-transactions*. A **null-transaction** is a transaction that does not contain any of the itemsets being examined. In our example,  $\overline{m\overline{c}}$  rep-

resents the number of null-transactions. *Lift* and  $\chi^2$  have difficulty distinguishing interesting pattern association relationships because they are both strongly influenced by  $\overline{mc}$ . Typically, the number of null-transactions can outweigh the number of individual purchases because, for example, many people may buy neither milk nor coffee. On the other hand, the other four measures are good indicators of interesting pattern associations because their definitions remove the influence of  $\overline{mc}$  (i.e., they are not influenced by the number of null-transactions).

This discussion shows that it is highly desirable to have a measure that is independent of the number of null-transactions. A measure is **null-invariant** if its value is free from the influence of null-transactions. Null-invariance is an important property for measuring association patterns in large transaction databases. Among the six discussed measures in this subsection, only *lift* and  $\chi^2$  are not null-invariant measures.

“Among the *all\_confidence*, *max\_confidence*, *Kulczynski*, and *cosine measures*, which is best at indicating interesting pattern relationships?”

To answer this question, we introduce the **imbalance ratio (IR)**, which assesses the imbalance of two itemsets,  $A$  and  $B$ , in rule implications. It is defined as

$$IR(A, B) = \frac{|sup(A) - sup(B)|}{sup(A) + sup(B) - sup(A \cup B)}, \quad (4.13)$$

where the numerator is the absolute value of the difference between the support of the itemsets  $A$  and  $B$ , and the denominator is the number of transactions containing  $A$  or  $B$ . If the two directional implications between  $A$  and  $B$  are the same, then  $IR(A, B)$  will be zero. Otherwise, the larger the difference between the two, the larger the imbalance ratio. This ratio is independent of the number of null-transactions and independent of the total number of transactions.

Let’s continue examining the remaining data sets in Example 4.10.

**Example 4.11. Comparing null-invariant measures in pattern evaluation.** Although the four measures introduced in this section are null-invariant, they may present dramatically different values on some subtly different data sets. Let’s examine data sets  $D_5$  and  $D_6$ , shown earlier in Table 4.9, where the two events  $m$  and  $c$  have unbalanced conditional probabilities. That is, the ratio of  $mc$  to  $c$  is greater than 0.9. This means that knowing that  $c$  occurs should strongly suggest that  $m$  occurs also. The ratio of  $mc$  to  $m$  is less than 0.1, indicating that  $m$  implies that  $c$  is quite unlikely to occur. The *all\_confidence* and *cosine* measures view both cases as negatively associated and the *Kulc* measure views both as neutral. The *max\_confidence* measure claims strong positive associations for these cases. The measures give very diverse results!

“Which measure intuitively reflects the true relationship between the purchase of milk and coffee?” Actually, in this case, it is difficult to argue whether the two data sets have positive or negative association. From one point of view, only  $mc/(mc + \overline{mc}) = 1000/(1000 + 10,000) = 9.09\%$  of milk-related transactions contain coffee in  $D_5$ , and this percentage is  $1000/(1000 + 100,000) = 0.99\%$  in  $D_6$ , both indicating a negative association. On the other hand, 90.9% of transactions in  $D_5$  (i.e.,  $mc/(mc + \overline{m}) = 1000/(1000 + 100)$ ) and 9% in  $D_6$  (i.e.,  $1000/(1000 + 10)$ ) containing coffee contain milk as well, which indicates a positive association between milk and coffee, a very different conclusion.

In this case, it is fair to treat it as neutral, as *Kulc* does. In the meantime, it will be good to also indicate its skewness using the *imbalance ratio (IR)*. According to Eq. (4.13), for  $D_4$  we have

$IR(m, c) = 0$ , a perfectly balanced case; for  $D_5$ ,  $IR(m, c) = 0.89$ , a rather imbalanced case; whereas for  $D_6$ ,  $IR(m, c) = 0.99$ , a very skewed case. Therefore the two measures, *Kulc* and *IR*, work together, presenting a clear picture for all three data sets,  $D_4$  through  $D_6$ .  $\square$

In summary, the use of only support and confidence measures to mine associations may generate a large number of rules, many of which can be uninteresting to users. Instead, we can augment the support–confidence framework with a pattern interestingness measure, which helps focus the mining toward rules with strong pattern relationships. The added measure substantially reduces the number of rules generated and leads to the discovery of more meaningful rules. Besides those introduced in this section, many other interestingness measures have been studied in the literature. Unfortunately, most of them do not have the null-invariance property. Because large data sets typically have many null-transactions, it is important to consider the null-invariance property when selecting appropriate interestingness measures for pattern evaluation. Among the four null-invariant measures studied here, namely *all\_confidence*, *max\_confidence*, *Kulc*, and *cosine*, we recommend using *Kulc* in conjunction with the imbalance ratio.

---

## 4.4 Summary

- The discovery of frequent patterns, associations, and correlation relationships among huge amounts of data is useful in selective marketing, decision analysis, and business management. A popular area of application is **market basket analysis**, which studies customers' buying habits by searching for itemsets that are frequently purchased together (or in sequence).
- **Association rule mining** consists of first finding **frequent itemsets** (sets of items, such as  $A$  and  $B$ , satisfying a *minimum support threshold*, or percentage of the task-relevant tuples), from which **strong** association rules in the form of  $A \Rightarrow B$  are generated. These rules also satisfy a *minimum confidence threshold* (a prespecified probability of satisfying  $B$  under the condition that  $A$  is satisfied). Associations can be further analyzed to uncover **correlation rules**, which convey statistical correlations between itemsets  $A$  and  $B$ .
- Many efficient and scalable algorithms have been developed for **frequent itemset mining**, from which association and correlation rules can be derived. These algorithms can be classified into three categories: (1) *Apriori-like algorithms*, (2) *frequent pattern growth-based algorithms* such as FP-growth, and (3) *algorithms that use the vertical data format*.
- The **Apriori algorithm** is a seminal algorithm for mining frequent itemsets for Boolean association rules. It explores the level-wise mining Apriori property that *all nonempty subsets of a frequent itemset must also be frequent*. At the  $k$ th iteration (for  $k \geq 2$ ), it forms frequent  $k$ -itemset candidates based on the frequent  $(k - 1)$ -itemsets, and scans the database once to find the *complete* set of frequent  $k$ -itemsets,  $L_k$ .  
Variations involving hashing and transaction reduction can be used to make the procedure more efficient. Other variations include partitioning the data (mining on each partition and then combining the results) and sampling the data (mining on a data subset). These variations can reduce the number of data scans required to as little as two or even one.
- **Frequent pattern growth** is a method of mining frequent itemsets without candidate generation. It constructs a highly compact data structure (an *FP-tree*) to compress the original transaction database. Rather than employing the generate-and-test strategy of Apriori-like methods, it focuses



on frequent pattern (fragment) growth, which avoids costly candidate generation, resulting in greater efficiency.

- **Mining frequent itemsets using the vertical data format (Eclat)** is a method that transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical data format of *item-TID\_set*. It mines the transformed data set by *TID\_set* intersections based on the Apriori property and additional optimization techniques such as *diffset*.
- Not all strong association rules are interesting. Therefore, the support–confidence framework should be augmented with a pattern evaluation measure, which promotes the mining of *interesting* rules. A measure is **null-invariant** if its value is free from the influence of **null-transactions** (i.e., the *transactions that do not contain any of the itemsets being examined*). Among many pattern evaluation measures, we examined *lift*,  $\chi^2$ , *all\_confidence*, *max\_confidence*, *Kulczynski*, and *cosine*, and showed that only the latter four are null-invariant. We suggest using the Kulczynski measure, together with the imbalance ratio, to present pattern relationships among itemsets.

---

## 4.5 Exercises

- 4.1. Suppose you have the set  $\mathcal{C}$  of all frequent closed itemsets on a data set  $D$ , as well as the support count for each frequent closed itemset. Describe an algorithm to determine whether a given itemset  $X$  is frequent or not, and the support of  $X$  if it is frequent.
- 4.2. An itemset  $X$  is called a *generator* on a data set  $D$  if there does not exist a proper subitemset  $Y \subset X$  such that  $support(X) = support(Y)$ . A generator  $X$  is a *frequent generator* if  $support(X)$  passes the minimum support threshold. Let  $\mathcal{G}$  be the set of all frequent generators on a data set  $D$ .
  - a. Can you determine whether an itemset  $A$  is frequent and the support of  $A$ , if it is frequent, using only  $\mathcal{G}$  and the support counts of all frequent generators? If yes, present your algorithm. Otherwise, what other information is needed? Can you give an algorithm assuming the information needed is available?
  - b. What is the relationship between closed itemsets and generators?
- 4.3. The Apriori algorithm makes use of *prior knowledge* of subset support properties.
  - a. Prove that all nonempty subsets of a frequent itemset must also be frequent.
  - b. Prove that the support of any nonempty subset  $s'$  of itemset  $s$  must be at least as great as the support of  $s$ .
  - c. Given frequent itemset  $l$  and subset  $s$  of  $l$ , prove that the confidence of the rule “ $s' \Rightarrow (l - s')$ ” cannot be more than the confidence of “ $s \Rightarrow (l - s)$ ,” where  $s'$  is a subset of  $s$ .
  - d. A *partitioning* variation of Apriori subdivides the transactions of a database  $D$  into  $n$  nonoverlapping partitions. Prove that any itemset that is frequent in  $D$  must be frequent in at least one partition of  $D$ .
- 4.4. Let  $c$  be a candidate itemset in  $C_k$  generated by the Apriori algorithm. How many length- $(k - 1)$  subsets do we need to check in the prune step? Per your previous answer, can you give an improved version of procedure `has_infrequent_subset` in Fig. 4.4?
- 4.5. Section 4.2.2 describes a method for *generating association rules* from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed there. (*Hint*: consider incorporating the properties of Exercises 4.3(b), (c) into your design.)

4.6. A database has five transactions. Let  $min\_sup = 60\%$  and  $min\_conf = 80\%$ .

TID	items_bought
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

- Find all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.
- List all the *strong* association rules (with support  $s$  and confidence  $c$ ) matching the following metarule, where  $X$  is a variable representing customers, and  $item_i$  denotes variables representing items (e.g., “A,” “B,”):

$$\forall X \in transaction, buys(X, item_1) \wedge buys(X, item_2) \Rightarrow buys(X, item_3) \quad [s, c]$$

4.7. (**Implementation project**) Using a programming language that you are familiar with, such as C++ or Java, implement three *frequent itemset mining* algorithms introduced in this chapter: (1) Apriori [AS94b], (2) FP-growth [HPY00], and (3) Eclat [Zak00] (mining using the vertical data format). Compare the performance of each algorithm with various kinds of large data sets. Write a report to analyze the situations (e.g., data size, data distribution, minimal support threshold setting, and pattern density) where one algorithm may perform better than the others, and state why.

4.8. A database has four transactions. Let  $min\_sup = 60\%$  and  $min\_conf = 80\%$ .

cust_ID	TID	items_bought (in the form of brand-item_category)
01	T100	{King’s-Crab, Sunset-Milk, Dairyland-Cheese, Best-Bread}
02	T200	{Best-Cheese, Dairyland-Milk, Goldenfarm-Apple, Tasty-Pie, Wonder-Bread}
01	T300	{Westcoast-Apple, Dairyland-Milk, Wonder-Bread, Tasty-Pie}
03	T400	{Wonder-Bread, Sunset-Milk, Dairyland-Cheese}

- At the granularity of *item\_category* (e.g.,  $item_i$  could be “Milk”), for the rule template,

$$\forall X \in transaction, buys(X, item_1) \wedge buys(X, item_2) \Rightarrow buys(X, item_3) \quad [s, c],$$

list the frequent  $k$ -itemset for the largest  $k$ , and *all* the *strong* association rules (with their support  $s$  and confidence  $c$ ) containing the frequent  $k$ -itemset for the largest  $k$ .

- At the granularity of *brand-item\_category* (e.g.,  $item_i$  could be “Sunset-Milk”), for the rule template,

$$\forall X \in customer, buys(X, item_1) \wedge buys(X, item_2) \Rightarrow buys(X, item_3),$$

list the frequent  $k$ -itemset for the largest  $k$  (but do not print any rules).

4.9. Suppose that a large store has a transactional database that is *distributed* among four locations. Transactions in each component database have the same format, namely  $T_j : \{i_1, \dots, i_m\}$ , where  $T_j$  is a transaction identifier, and  $i_k$  ( $1 \leq k \leq m$ ) is the identifier of an item purchased in the transaction. Propose an efficient algorithm to mine global association rules. Your algorithm should not require shipping all the data to one site and should not cause excessive network communication overhead.

- 4.10. Suppose that frequent itemsets are saved for a large transactional database,  $DB$ . Discuss how to efficiently mine the (global) association rules under the same minimum support threshold, if a set of new transactions, denoted as  $\Delta DB$ , is (*incrementally*) added in?
- 4.11. Most frequent pattern mining algorithms consider only distinct items in a transaction. However, multiple occurrences of an item in the same shopping basket, such as four cakes and three jugs of milk, can be important in transactional data analysis. How can one mine frequent itemsets efficiently considering multiple occurrences of items? Propose modifications to the well-known algorithms, such as Apriori and FP-growth, to adapt to such a situation.
- 4.12. (**Implementation project**) Many techniques have been proposed to further improve the performance of frequent itemset mining algorithms. Taking FP-tree-based frequent pattern growth algorithms (e.g., FP-growth) as an example, implement one of the following optimization techniques. Compare the performance of your new implementation with the unoptimized version.
- The frequent pattern mining method of Section 4.2.4 uses an FP-tree to generate conditional pattern bases using a bottom-up projection technique (i.e., project onto the prefix path of an item  $p$ ). However, one can develop a *top-down projection* technique, that is, project onto the suffix path of an item  $p$  in the generation of a conditional pattern base. Design and implement such a top-down FP-tree mining method. Compare its performance with the bottom-up projection method.
  - Nodes and pointers are used uniformly in an FP-tree in the FP-growth algorithm design. However, such a structure may consume a lot of space when the data are sparse. One possible alternative design is to explore *array- and pointer-based hybrid implementation*, where a node may store multiple items when it contains no splitting point to multiple subbranches. Develop such an implementation and compare it with the original one.
  - It is time and space consuming to generate numerous conditional pattern bases during pattern-growth mining. An interesting alternative is to *push right* the branches that have been mined for a particular item  $p$ , that is, to push them to the remaining branch(es) of the FP-tree. This is done so that fewer conditional pattern bases have to be generated and additional sharing can be explored when mining the remaining FP-tree branches. Design and implement such a method and conduct a performance study on it.
- 4.13. Give a short example to show that items in a strong association rule actually may be *negatively correlated*.
- 4.14. The following contingency table summarizes supermarket transaction data, where *hot dogs* refers to the transactions containing hot dogs,  $\overline{\text{hot dogs}}$  refers to the transactions that do not contain hot dogs, *hamburgers* refers to the transactions containing hamburgers, and  $\overline{\text{hamburgers}}$  refers to the transactions that do not contain hamburgers.

	<i>hotdogs</i>	$\overline{\text{hot dogs}}$	$\Sigma_{row}$
<i>hamburgers</i>	2000	500	2500
$\overline{\text{hamburgers}}$	1000	1500	2500
$\Sigma_{col}$	3000	2000	5000

- Suppose that the association rule “*hot dogs*  $\Rightarrow$  *hamburgers*” is mined. Given a minimum support threshold of 25% and a minimum confidence threshold of 50%, is this association rule strong?

- b. Based on the given data, is the purchase of *hot dogs* independent of the purchase of *ham-burgers*? If not, what kind of *correlation* relationship exists between the two?
  - c. Compare the use of the *all\_confidence*, *max\_confidence*, *Kulczynski*, and *cosine* measures with *lift* and *correlation* on the given data.
- 4.15. (Implementation project)** The DBLP data set (<https://dblp.uni-trier.de/xml/>) consists of over three million entries of research papers published in computer science conferences and journals. Among these entries, there are a good number of authors that have coauthor relationships.
- a. Propose a method to efficiently mine a set of coauthor relationships that are closely correlated (e.g., often coauthoring papers together).
  - b. Based on the mining results and the pattern evaluation measures discussed in this chapter, discuss which measure may convincingly uncover close collaboration patterns better than others.
  - c. Based on the study in (a), develop a method that can roughly predict advisor and advisee relationships and the approximate period for such advisory supervision.

---

## 4.6 Bibliographic notes

Association rule mining was first proposed by Agrawal, Imielinski, and Swami [AIS93]. The Apriori algorithm discussed in Section 4.2.1 for frequent itemset mining was presented in Agrawal and Srikant [AS94b]. A variation of the algorithm using a similar pruning heuristic was developed independently by Mannila, Tiovonen, and Verkamo [MTV94]. A joint publication combining these works later appeared in Agrawal et al. [AMS<sup>+</sup>96]. A method for generating association rules from frequent itemsets is described in Agrawal and Srikant [AS94a].

References for the variations of Apriori described in Section 4.2.3 include the following. The use of hash tables to improve association mining efficiency was studied by Park, Chen, and Yu [PCY95a]. The partitioning technique was proposed by Savasere, Omiecinski, and Navathe [SON95]. The sampling approach is discussed in Toivonen [Toi96]. A dynamic itemset counting approach is given in Brin, Motwani, Ullman, and Tsur [BMUT97]. An efficient incremental updating of mined association rules was proposed by Cheung, Han, Ng, and Wong [CHNW96]. Parallel and distributed association data mining under the Apriori framework was studied by Park, Chen, and Yu [PCY95b]; Agrawal and Shafer [AS96]; and Cheung et al. [CHN<sup>+</sup>96]. Another parallel association mining method, which explores itemset clustering using a vertical database layout, was proposed in Zaki, Parthasarathy, Ogihara, and Li [ZPOL97].

Other scalable frequent itemset mining methods have been proposed as alternatives to the Apriori-based approach. FP-growth, a pattern-growth approach for mining frequent itemsets without candidate generation, was proposed by Han, Pei, and Yin [HPY00] (Section 4.2.4). An exploration of hyper structure mining of frequent patterns, called H-Mine, was proposed by Pei et al. [PHL01]. A method that integrates top-down and bottom-up traversal of FP-trees in pattern-growth mining was proposed by Liu, Pan, Wang, and Han [LPWH02]. An array-based implementation of prefix-tree structure for efficient pattern growth mining was proposed by Grahne and Zhu [GZ03b]. Eclat, an approach for mining frequent itemsets by exploring the vertical data format, was proposed by Zaki [Zak00]. A depth-first generation of frequent itemsets by a tree projection technique was proposed by Agarwal, Aggarwal,

and Prasad [AAP01]. An integration of association mining with relational database systems was studied by Sarawagi, Thomas, and Agrawal [STA98].

The mining of frequent closed itemsets was proposed in Pasquier, Bastide, Taouil, and Lakhal [PRTL99], where an Apriori-based algorithm called A-Close for such mining was presented. CLOSET, an efficient closed itemset mining algorithm based on the frequent pattern growth method, was proposed by Pei, Han, and Mao [PHM00]. CHARM by Zaki and Hsiao [ZH02] developed a compact vertical TID list structure called *diffset*, which records only the difference in the TID list of a candidate pattern from its prefix pattern. A fast hash-based approach is also used in CHARM to prune nonclosed patterns. CLOSET+ by Wang, Han, and Pei [WHP03] integrates previously proposed effective strategies as well as newly developed techniques such as hybrid tree-projection and item skipping. AFOPT, a method that explores a *right push* operation on FP-trees during the mining process, was proposed by Liu, Lu, Lou, and Yu [LLLY03]. Grahne and Zhu [GZ03b] proposed a prefix-tree-based algorithm integrated with array representation, called FPClose, for mining closed itemsets using a pattern-growth approach.

Pan et al. [PCT<sup>+</sup>03] proposed CARPENTER, a method for finding closed patterns in long biological data sets, which integrates the advantages of vertical data formats and pattern growth methods. Mining max-patterns was first studied by Bayardo [Bay98], where MaxMiner, an Apriori-based, level-wise, breadth-first search method, was proposed to find *max-itemset* by performing *superset frequency pruning* and *subset infrequency pruning* for search space reduction. Another efficient method, MAFIA, developed by Burdick, Calimlim, and Gehrke [BCG01], uses vertical bitmaps to compress TID lists, thus improving the counting efficiency. A FIMI (Frequent Itemset Mining Implementation) workshop dedicated to implementation methods for frequent itemset mining was reported by Goethals and Zaki [GZ03a].

The problem of mining interesting rules has been studied by many researchers. The statistical independence of rules in data mining was studied by Piatetski-Shapiro [PS91]. The interestingness problem of strong association rules is discussed in Chen, Han, and Yu [CHY96]; Brin, Motwani, and Silverstein [BMS97]; and Aggarwal and Yu [AY99], which cover several interestingness measures, including *lift*. An efficient method for generalizing associations to correlations is given in Brin, Motwani, and Silverstein [BMS97]. Other alternatives to the support–confidence framework for assessing the interestingness of association rules are proposed in Brin, Motwani, Ullman, and Tsur [BMUT97] and Ahmed, El-Makky, and Taha [AEMT00].

A method for mining strong gradient relationships among itemsets was proposed by Imielinski, Khachiyani, and Abdulghani [IKA02]. Silverstein, Brin, Motwani, and Ullman [SBMU98] studied the problem of mining causal structures over transaction databases. Some comparative studies of different interestingness measures were done by Hilderman and Hamilton [HH01]. The notion of null transaction invariance was introduced, together with a comparative analysis of interestingness measures, by Tan, Kumar, and Srivastava [TKS02]. The use of *all\_confidence* as a correlation measure for generating interesting association rules was studied by Omiecinski [Omi03] and by Lee, Kim, Cai, and Han [LKCH03]. Wu, Chen, and Han [WCH10] introduced the Kulczynski measure for associative patterns and performed a comparative analysis of a set of measures for pattern evaluation.

# Pattern mining: advanced methods

# 5

**Frequent pattern mining** has reached far beyond the basics due to substantial research, numerous extensions of the problem scope, and broad application studies. In this chapter, we will learn methods for advanced pattern mining. We first introduce methods for mining various kinds of patterns, including mining multilevel patterns, multidimensional patterns, patterns in continuous data, rare patterns, negative patterns, and frequent patterns in high-dimensional data. We also examine methods for mining compressed and approximate patterns. Then we examine the methodologies of utilizing constraints to reduce the cost of frequent pattern mining. Since sequential patterns and structural patterns are popularly encountered but they need rather different mining methods, we introduce concepts and methods for mining sequential patterns in sequence data sets and mining subgraph patterns in graph data sets. To get the flavor on how to extend pattern mining methods to facilitate diverse applications, we examine one example on mining copy-and-paste bugs in large software programs. Notice that *pattern mining* is a more general term than *frequent pattern mining* since the former covers rare and negative patterns as well. However, when there is no ambiguity, the two terms are used interchangeably.

## 5.1 Mining various kinds of patterns

In the last chapter we have studied methods for mining patterns and associations at a single concept level and single dimensional space (e.g., products purchased). However, in many applications, people may like to uncover more complex patterns from massive data. For example, one may like to find *multilevel associations* that involve concepts at different abstraction levels, *multidimensional associations* that involve more than one dimension or predicate (e.g., rules that relate what a customer *buys* to his or her *age*), *quantitative association rules* that involve numeric attributes (e.g., *age*, *salary*), *rare patterns* that may suggest interesting although rare item combinations, and *negative patterns* that show negative correlations between items.

In this section we examine methods for mining patterns and associations at multiple abstraction levels (Section 5.1.1) and at multidimensional spaces (Section 5.1.2), handling data with quantitative attributes (Section 5.1.3), mining patterns in high-dimensional space (Section 5.1.4), and mining rare patterns and negative patterns (Section 5.1.5).

### 5.1.1 Mining multilevel associations

For many applications, strong associations discovered at high abstraction levels, though often having high support, could be commonsense knowledge (e.g., buying bread and milk frequently together). We may want to drill down to find novel patterns at more detailed levels (e.g., buying what kind of

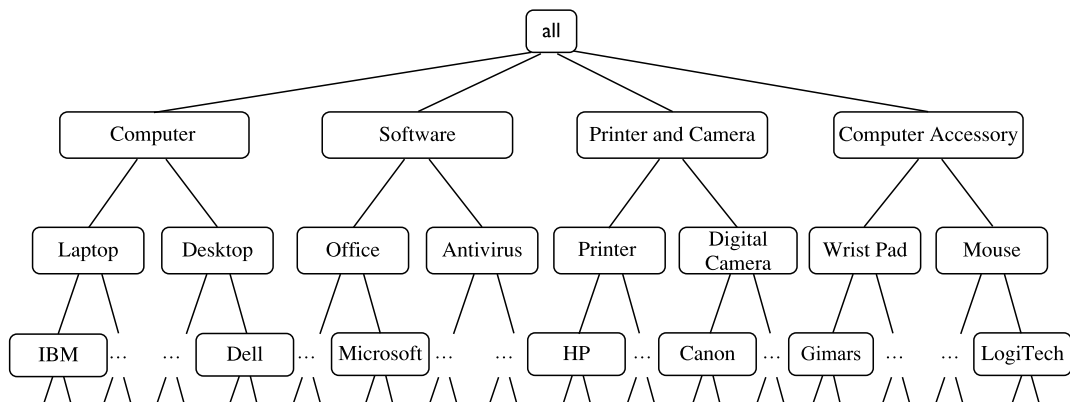
bread and what kind of milk frequently together). On the other hand, there could be too many scattered patterns at low or primitive abstraction levels, some of which are just trivial specializations of patterns at higher levels. Therefore, it is interesting to examine how to develop effective methods for mining meaningful patterns at multiple abstraction levels, with sufficient flexibility for easy traversal among different abstraction spaces.

**Example 5.1. Mining multilevel association rules.** Suppose we are given the task-relevant set of transactional data in Table 5.1 for sales in an e-store, showing the items purchased for each transaction. The concept hierarchy for the items is shown in Fig. 5.1. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to a higher-level, more general concept set. Data can be generalized by replacing low-level concepts within the data by their corresponding higher-level concepts, or *ancestors*, from a concept hierarchy.

The concept hierarchy in Fig. 5.1 has five levels, respectively, referred to as levels 0 through 4, starting with level 0 at the root node for all (the most general abstraction level). Here, level 1 includes *computer*, *software*, *printer and camera*, and *computer accessory*; level 2 includes *laptop computer*, *desktop computer*, *office software*, *antivirus software*, etc.; and level 3 includes *Dell desktop computer*, ..., *Microsoft office software*, etc. Level 4 is the most specific abstraction level of this hierarchy. It consists of concrete products.

**Table 5.1 Task-relevant data,  $D$ .**

TID	Items Purchased
T100	Apple 15" MacBook Pro, HP Photosmart 7520 printer
T200	Microsoft Office Professional 2020, Microsoft Surface Mobile Mouse
T300	Logitech MX Master 2S Wireless Mouse, Gimars GEL Wrist Rest
T400	Dell Studio XPS 16 Notebook, Canon PowerShot SX70 HS Digital Camera
T500	Apple iPad Air (10.5-inch, Wi-Fi, 256GB), Norton Security Premium
...	...



**FIGURE 5.1**

Concept hierarchy for computer items of an e-store.

Concept hierarchies for nominal attributes may be specified by users familiar with the data such as store managers. Alternatively, they can be generated from data, based on the analysis of product specifications, attribute values, or data distributions. Concept hierarchies for numeric attributes can be generated using discretization techniques, such as those introduced in Chapter 2. For our example, the concept hierarchy of Fig. 5.1 is provided.

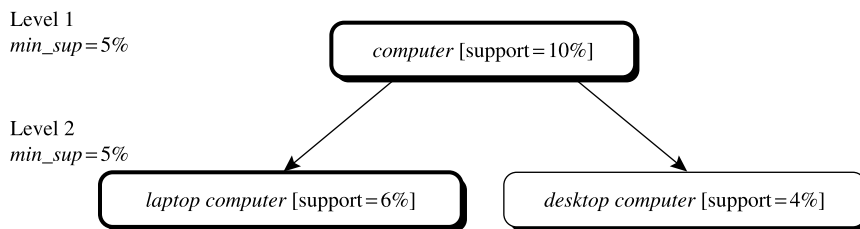
The items in Table 5.1 are at the lowest level of Fig. 5.1's concept hierarchy. It is difficult to find interesting purchase patterns in such primitive-level data. For instance, if “*Dell Studio XPS 16 Notebook*” or “*Logitech VX Nano Cordless Laser Mouse*” occurs in a very small fraction of the transactions, then it can be difficult to find strong associations involving these specific items. Few people may buy these items together, making it unlikely that the itemset will satisfy minimum support. However, we would expect that it is easier to find strong associations between generalized abstractions of these items, such as between “*Dell Notebook*” and “*Cordless Mouse*.” □

Association rules generated from mining data at multiple abstraction levels are called **multiple-level** or **multilevel association rules**. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy can be employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at concept level 1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations.

A number of variations to this approach are described next, where each variation involves “playing” with the support threshold in a slightly different way. The variations are illustrated in Figs. 5.2 and 5.3, where nodes indicate an item or itemset that has been examined, and nodes with thick borders indicate that an examined item or itemset is frequent.

- **Using uniform minimum support for all levels** (referred to as **uniform support**): The same minimum support threshold is used when mining at each abstraction level. For example, in Fig. 5.2, a minimum support threshold of 5% is used throughout (e.g., for mining from “*computer*” downward to “*laptop computer*”). Both “*computer*” and “*laptop computer*” are found to be frequent, whereas “*desktop computer*” is not.

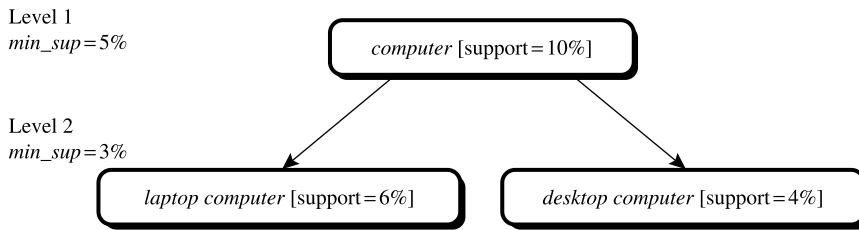
When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of



**FIGURE 5.2**

Multilevel mining with uniform support.



**FIGURE 5.3**

Multilevel mining with reduced support.

its descendants: The search avoids examining itemsets containing any item or itemset of which the ancestors do not have minimum support.

The uniform support approach, however, has some drawbacks. It is unlikely that items at lower abstraction levels will occur as frequently as those at higher abstraction levels. If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the next approach.

- **Using reduced minimum support at lower levels** (referred to as **reduced support**): Each abstraction level has its own minimum support threshold. The deeper the abstraction level, the smaller the corresponding threshold. For example, in Fig. 5.3, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, “*computer*,” “*laptop computer*,” and “*desktop computer*” are all considered frequent.

For mining multilevel patterns with reduced support, the minimum support threshold at the lowest abstraction level should be used during the mining process to allow mining to penetrate down to the lowest abstraction level. However, for the final pattern/rule extraction, thresholds associated with the corresponding items should be enforced to print out only interesting associations.

- **Using item or group-based minimum support** (referred to as **group-based support**): Because users or experts often have insights as to which groups are more important than others, it is sometimes desirable to set up user-specific, item-based, or group-based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price or on items of interest, such as by setting particularly low support thresholds for “*camera with price over \$ 1000*,” to pay particular attention to the association patterns containing items in these categories.

For mining patterns with mixed items from groups with different support thresholds, usually the lowest support threshold among all the participating groups is taken as the support threshold in mining. This will avoid filtering out valuable patterns containing items from the group with the lowest support threshold. In the meantime, the minimal support threshold for each individual group should be kept to avoid generating uninteresting itemsets from each group. Other interestingness measures can be used after the itemset mining to extract truly interesting rules.

A serious side effect of mining multilevel association rules is its generation of many redundant rules across multiple abstraction levels due to the “ancestor” relationships among items. For example,

consider the following rules where “*laptop computer*” is an ancestor of “*Dell laptop computer*” based on the concept hierarchy of Fig. 5.1, and where  $X$  is a variable representing customers who purchased items.

$$\begin{aligned} \text{buys}(X, \text{“laptop computer”}) &\Rightarrow \text{buys}(X, \text{“HP printer”}) \\ &[\text{support} = 8\%, \text{confidence} = 70\%] \end{aligned} \quad (5.1)$$

$$\begin{aligned} \text{buys}(X, \text{“Dell laptop computer”}) &\Rightarrow \text{buys}(X, \text{“HP printer”}) \\ &[\text{support} = 2\%, \text{confidence} = 72\%] \end{aligned} \quad (5.2)$$

“If Rules (5.1) and (5.2) are both mined, does Rule (5.2) provide any novel information?” We say a rule  $R1$  is an **ancestor** of a rule  $R2$ , if  $R1$  can be obtained by replacing the items in  $R2$  by their ancestors in a concept hierarchy. For example, Rule (5.1) is an ancestor of Rule (5.2) because “*laptop computer*” is an ancestor of “*Dell laptop computer*.” Based on this definition, a rule can be considered redundant if its support and confidence are close to their “expected” values, based on an ancestor of the rule.

**Example 5.2. Checking redundancy among multilevel association rules.** Suppose that about one-quarter of all “*laptop computer*” sales are for “*Dell laptop computers*.” Since Rule (5.1) has a 70% confidence and 8% support, we may expect Rule (5.2) to have a confidence of around 70% (since all data samples of “*Dell laptop computer*” are also samples of “*laptop computer*”) and a support of around 2% (i.e.,  $8\% \times \frac{1}{4}$ ). If this is indeed the case, then Rule (5.2) is not interesting because it does not offer any additional information and is less general than Rule (5.1).  $\square$

### 5.1.2 Mining multidimensional associations

So far, we have studied association rules that imply a single predicate, that is, the predicate *buys*. For instance, at mining a data set, we may discover the Boolean association rule

$$\text{buys}(X, \text{“Apple iPad air”}) \Rightarrow \text{buys}(X, \text{“HP printer”}). \quad (5.3)$$

Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to Rule (5.3) as a **single-dimensional** or **intradimensional association rule** because it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule). Such rules are commonly mined from transactional data.

Instead of considering transactional data only, sales and related information are often linked with relational data or integrated into a data warehouse. Such data stores are multidimensional in nature. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items and/or transactions such as the item description or the branch location of the sale. Additional relational information regarding the customers who purchased the items (e.g., customer age, occupation, credit rating, income, and address) may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates such as

$$\text{age}(X, \text{“18...25”}) \wedge \text{occupation}(X, \text{“student”}) \Rightarrow \text{buys}(X, \text{“laptop”}). \quad (5.4)$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (5.4) contains three predicates (*age*, *occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has **no repeated predicates**. Multidimensional association rules with no repeated predicates are called **interdimensional association rules**. We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimensional association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, "18 \dots 25") \wedge buys(X, "laptop") \Rightarrow buys(X, "HP printer"). \quad (5.5)$$

Database attributes can be nominal or quantitative. The values of **nominal** (or categorical) attributes are “names of things.” Nominal attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). **Quantitative** attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*). Techniques for mining multidimensional association rules can be categorized into two basic approaches regarding the treatment of quantitative attributes.

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels such as “0..20K,” “21K..30K,” “31K..40K,” and so on. Here, discretization is *static* and predetermined. Chapter 2 on data preprocessing gave several techniques for discretizing numeric attributes. The discretized numeric attributes, with their interval labels, can then be treated as nominal attributes (where each interval is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes**.

In the second approach, *quantitative attributes are discretized or clustered into “bins” based on the data distribution*. These bins may be further combined during the mining process. The discretization process is *dynamic* and established to satisfy some mining criteria such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as **(dynamic) quantitative association rules**.

Let’s study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to interdimensional association rules. Note that rather than searching for frequent itemsets (as is done for single-dimensional association rule mining), in multidimensional association rule mining we search for frequent *predicate sets*. A **k-predicate set** is a set containing *k* conjunctive predicates. For instance, the set of predicates {*age*, *occupation*, *buys*} from Rule (5.4) is a 3-predicate set.

### 5.1.3 Mining quantitative association rules

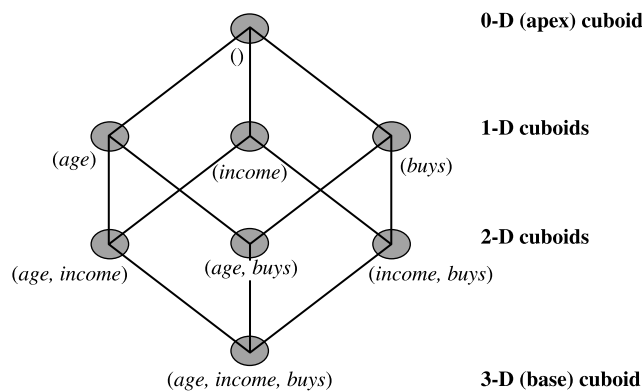
As discussed earlier, relational and data warehouse data often involve quantitative attributes or measures. We can discretize quantitative attributes into multiple intervals and then treat them as nominal data in association mining. However, such simple discretization may lead to the generation of an enormous number of rules, many of which may not be useful. Here we introduce three methods that can help overcome this difficulty to discover novel association relationships: (1) a data cube method, (2) a clustering-based method, and (3) a statistical analysis method to uncover exceptional behaviors.

### Data cube–based mining of quantitative associations

In many cases quantitative attributes can be discretized before mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. Nominal attributes may also be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then any of the frequent itemset mining algorithms we have discussed can easily be modified so as to find all frequent predicate sets. In particular, instead of searching on only one attribute like *buys*, we need to search through all of the relevant attributes, treating each attribute-value pair as an itemset.

Alternatively, the transformed multidimensional data may be used to construct a *data cube*. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (e.g., counts) in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules. An overview of data cube technology and data cube computation algorithms were presented in Chapter 3. Fig. 5.4 shows the lattice of cuboids defining a data cube for the dimensions *age*, *income*, and *buys*. The cells of an  $n$ -dimensional cuboid can be used to store the support counts of the corresponding  $n$ -predicate sets. The base cuboid aggregates the task-relevant data by *age*, *income*, and *buys*; the 2-D cuboid, (*age*, *income*), aggregates by *age* and *income*, and so on; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data.

Due to the ever-increasing use of data warehouse and OLAP technology, it is possible that a data cube containing the dimensions that are of interest to the user may already exist, fully or partially materialized. If this is the case, we can simply fetch the corresponding aggregate values or compute them using lower-level materialized aggregates, and return the rules needed using a rule generation algorithm. Notice that even in this case, the Apriori property can still be used to prune the search space. If a given  $k$ -predicate set has support *sup*, which does not satisfy minimum support, then further exploration of this set should be terminated. This is because any more-specialized version of the  $k$ -itemset will have support no greater than *sup* and, therefore, will not satisfy minimum support either. In cases where no relevant data cube exists for the mining task, we must create one on-the-fly. This



**FIGURE 5.4**

Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age*, *income*, and *buys*.

becomes an iceberg cube computation problem, where the minimum support threshold is taken as the iceberg condition (Chapter 3).

### ***Mining clustering-based quantitative associations***

Besides using discretization-based or data cube-based data sets to generate quantitative association rules, we can also generate *quantitative association rules* by clustering data in the quantitative dimensions. (Recall that objects within a cluster are similar to one another and dissimilar to those in other clusters.) The general assumption is that interesting frequent patterns or association rules are in general found at relatively dense clusters of quantitative attributes. Here, we describe a top-down approach and a bottom-up approach to clustering that finds quantitative associations.

A typical top-down approach for finding clustering-based quantitative frequent patterns is as follows. For each quantitative dimension, a standard clustering algorithm (e.g.,  $k$ -means or a density-based clustering algorithm, as described in Chapter 8) can be applied to find clusters in this dimension that satisfy the minimum support threshold. For each cluster, we then examine the 2-D spaces generated by combining the cluster with a cluster or nominal value of another dimension to see if such a combination passes the minimum support threshold. If it does, we continue to search for clusters in this 2-D region and progress to even higher-dimensional combinations. The Apriori pruning still applies in this process: If, at any point, the support of a combination does not have minimum support, its further partitioning or combination with other dimensions cannot have minimum support either.

A bottom-up approach for finding clustering-based frequent patterns works by first clustering in high-dimensional space to form clusters with support that satisfies the minimum support threshold, and then projecting and merging those clusters in the space containing fewer dimensional combinations. However, for high-dimensional data sets, finding high-dimensional clustering itself is a tough problem. Thus, this approach is less realistic.

### ***Using statistical theory to disclose exceptional behavior***

It is possible to discover quantitative association rules that disclose exceptional behavior, where “exceptional” is defined based on a statistical theory. For example, the following association rule may indicate exceptional behavior:

$$gender = female \Rightarrow mean\ wage = \$7.90/hr \text{ (overall\_mean\_wage} = \$9.02/hr). \quad (5.6)$$

This rule states that the average wage for females is only \$7.90/hr. This rule is (subjectively) interesting because it reveals a group of people earning a significantly lower wage than the average wage of \$9.02/hr.

An integral aspect of our definition involves applying statistical tests to confirm the validity of our rules. That is, Rule (5.6) is only accepted if a statistical test (in this case, a Z-test) confirms that with high confidence it can be inferred that the mean wage of the female population is indeed lower than the mean wage of the rest of the population.<sup>1</sup>

---

<sup>1</sup> The above rule was mined from a real database based on a 1985 U.S. census.

### 5.1.4 Mining high-dimensional data

Our discussions of mining multidimensional patterns in the above two subsections are confined to patterns involving a small number of dimensions. However, some applications may need to mine *high-dimensional data* (i.e., data with hundreds or thousands of dimensions). However, it is not easy to extend the previous multidimensional pattern mining methods to mine high-dimensional data because the search spaces of such methods grow exponentially with the number of dimensions.

One interesting direction to handle high-dimensional data is to extend a pattern growth approach by exploring the vertical data format to handle data sets with a large number of *dimensions* (also called *features* or *items*, e.g., genes) but a *small* number of *rows* (also called *transactions* or *tuples*, e.g., samples). This is useful in applications like the analysis of gene expressions in bioinformatics, for example, where we often need to analyze microarray data that contain a *large* number of genes (e.g., 10,000 to 100,000) but only a *small* number of samples (e.g., dozens to hundreds).

Another direction is to develop a new methodology that focuses its mining effort on *colossal patterns*, that is, patterns of rather long length, instead of the *complete set* of patterns. One interesting such method is called *Pattern-Fusion*, which takes leaps in the pattern search space, leading to a good approximation of the complete set of colossal frequent patterns. We briefly outline the idea of pattern-fusion here and refer interested readers to the detailed technical paper.

In some applications (e.g., bioinformatics), a researcher can be more interested in finding *colossal* patterns (e.g., long DNA and protein sequences) than finding small (i.e., short) ones since colossal patterns usually carry more significant meanings. Finding colossal patterns is challenging because incremental mining tends to get “trapped” by an explosive number of midsize patterns before it can even reach candidate patterns of large size.

All of the pattern mining strategies we have studied so far, such as Apriori and FP-growth, use an incremental growth strategy by nature, that is, they increase the length of candidate patterns by one at a time. Breadth-first search methods like Apriori cannot bypass the generation of an explosive number of midsize patterns generated, making it impossible to reach colossal patterns. Even depth-first search methods like FP-growth can be easily trapped in a huge number of subtrees before reaching colossal patterns. Clearly, a completely new mining methodology is needed to overcome such a hurdle.

As we have observed in Fig. 5.5, there could be a small number of colossal patterns (e.g., patterns of size close to 100) but such patterns may generate an exponential number of mid-sized patterns. Instead of mining a complete set of mid-sized patterns, *Pattern-Fusion* fuses a small number of shorter patterns into bigger colossal pattern candidates, and checks against the data set to see which of such candidates are the true frequent patterns, which can be further fused to generate even larger colossal pattern candidates. Such step-by-step fusing takes leaps in the pattern search space and avoids the pitfalls of both breadth-first and depth-first searches, as shown in Fig. 5.6.

Note that a colossal pattern such as  $\{a_1, a_2, \dots, a_{100}\} : 55$  implies that the data set contains many, many short subpatterns like  $\{a_1, a_2, a_9, \dots, a_{30}\} : 55+$ ;  $\dots$ ,  $\{a_1, a_9, \dots, a_{40}\} : 55+$ ;  $\dots$ ), where  $55+$  means with support count of at least 55. That is, a colossal pattern should generate far more small patterns than smaller patterns do. Thus, a colossal pattern is more robust in the sense that *if a small number of items are removed from the pattern, the resulting pattern would have a similar support set*. The larger the pattern size, the more prominent this robustness. Such a robustness relationship between a colossal pattern and its corresponding short patterns can be extended to multiple levels.

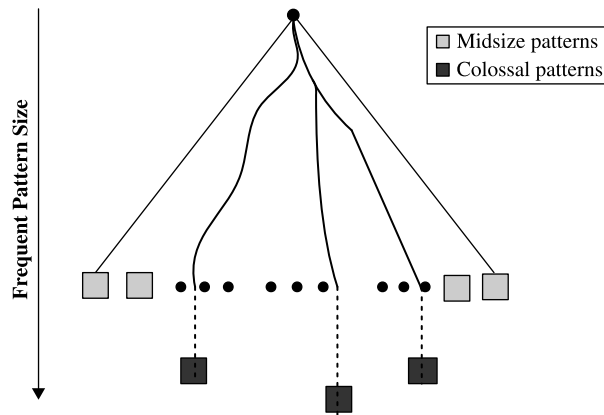


FIGURE 5.5

A high-dimensional data set may contain a small set of colossal patterns but exponentially many midsized patterns.

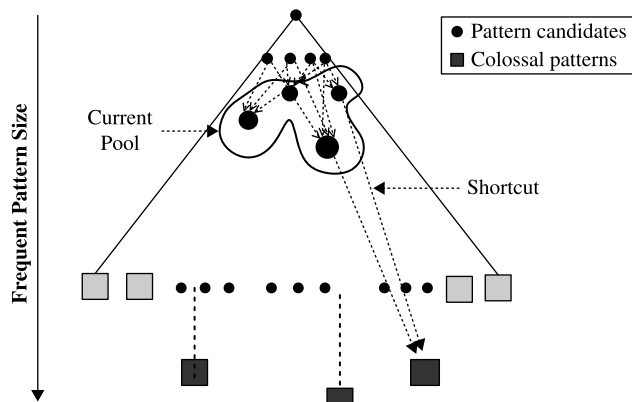


FIGURE 5.6

Pattern tree traversal: Candidates are taken from a pool of patterns, which results in shortcuts through pattern space to the colossal patterns.

Thus Pattern-Fusion has the capability to identify good merging candidates, which are the patterns that share some subpatterns and have some similar support sets. This does help the search leaps through pattern space more directly toward colossal patterns.

It has been theoretically shown that Pattern-Fusion leads to a good approximation of colossal patterns (see [ZYH<sup>+</sup>07]). The method was tested on synthetic and real data sets constructed from program tracing data and microarray data. Experiments show that the method can find most of the colossal patterns with high efficiency.

### 5.1.5 Mining rare patterns and negative patterns

All the methods presented so far in this chapter have been for mining frequent patterns. Sometimes, however, it is interesting to find patterns that are rare instead of frequent, or patterns that reflect a negative correlation between items. These patterns are respectively referred to as rare patterns and negative patterns. In this subsection, we consider various ways of defining rare patterns and negative patterns, which are also useful to mine.

**Example 5.3. Rare patterns and negative patterns.** In jewelry sales data, sales of diamond watches are rare; however, patterns involving the selling of diamond watches could be interesting. In supermarket data, if we find that customers frequently buy Coca-Cola Classic or Diet Coke but not both, then buying Coca-Cola Classic and buying Diet Coke together is considered a negative (correlated) pattern. In car sales data, a dealer sells a few fuel-thirsty vehicles (e.g., SUVs) to a given customer, and then later sells electric cars to the same customer. Even though buying SUVs and buying electric cars may be negatively correlated events, it can be interesting to discover and examine such exceptional cases.  $\square$

An **infrequent** (or **rare**) **pattern** is a pattern with a frequency support that is *below* (or *far below*) a user-specified (relative) minimum support threshold. However, since the occurrence frequencies of the majority of itemsets are usually below or even far below the minimum support threshold, it is desirable in practice for users to specify other conditions for rare patterns. For example, if we want to find patterns containing at least one item with a value that is over \$500, we should specify such a constraint explicitly. Efficient mining of such itemsets is discussed under mining multidimensional associations (Section 5.1.1), where the strategy is to adopt multiple (e.g., item- or group-based) minimum support thresholds. Other applicable methods are discussed under constraint-based pattern mining (Section 5.3), where user-specified constraints are pushed deep into the iterative mining process.

There are various ways we could define a negative pattern. We will consider three such definitions.

**Definition 5.1.** If itemsets  $X$  and  $Y$  are both frequent but rarely occur together (i.e.,  $\text{sup}(X \cup Y) < \text{sup}(X) \times \text{sup}(Y)$ ), then itemsets  $X$  and  $Y$  are **negatively correlated**, and the pattern  $X \cup Y$  is a **negatively correlated pattern**. If  $\text{sup}(X \cup Y) \ll \text{sup}(X) \times \text{sup}(Y)$ , then  $X$  and  $Y$  are **strongly negatively correlated**, and the pattern  $X \cup Y$  is a **strongly negatively correlated pattern**.

This definition can easily be extended for patterns containing  $k$ -itemsets for  $k > 2$ .

A problem with the definition, however, is that it is not *null-invariant*. That is, its value can be misleadingly influenced by null transactions, where a *null-transaction* is a transaction that does not contain any of the itemsets being examined (Section 4.3.3). This is illustrated in Example 5.4.

**Example 5.4. Null-transaction problem with Definition 5.1.** If there are a lot of null-transactions in the data set, then the number of null-transactions rather than the patterns observed may strongly influence a measure's assessment as to whether a pattern is negatively correlated. For example, suppose a sewing store sells needle packages  $A$  and  $B$ . The store sold 100 packages each of  $A$  and  $B$ , but only one transaction contains both  $A$  and  $B$ . Intuitively,  $A$  is negatively correlated with  $B$  since the purchase of one does not seem to encourage the purchase of the other.

Let's see how the above definition handles this scenario. If there are 200 transactions, we have  $\text{sup}(A \cup B) = 1/200 = 0.005$  and  $\text{sup}(A) \times \text{sup}(B) = 100/200 \times 100/200 = 0.25$ . Thus,  $\text{sup}(A \cup B) \ll \text{sup}(A) \times \text{sup}(B)$ , and so Definition 5.1 indicates that  $A$  and  $B$  are strongly negatively correlated. What if, instead of only 200 transactions in the database, there are  $10^6$ ? In this case, there



are many null-transactions, that is, many contain neither  $A$  nor  $B$ . How does the definition hold up? It computes  $\text{sup}(A \cup B) = 1/10^6$  and  $\text{sup}(X) \times \text{sup}(Y) = 100/10^6 \times 100/10^6 = 1/10^8$ . Thus,  $\text{sup}(A \cup B) \gg \text{sup}(X) \times \text{sup}(Y)$ , which contradicts the earlier finding even though the number of occurrences of  $A$  and  $B$  has not changed. The measure in Definition 5.1 is not null-invariant, where *null-invariance* is essential for quality interestingness measures as discussed in Section 4.3.3.  $\square$

**Definition 5.2.** If  $X$  and  $Y$  are strongly negatively correlated, then

$$\text{sup}(X \cup \bar{Y}) \times \text{sup}(\bar{X} \cup Y) \gg \text{sup}(X \cup Y) \times \text{sup}(\bar{X} \cup \bar{Y}).$$

Intuitively, it says that two itemsets  $X$  and  $Y$  are strongly negatively correlated if the probability of a transaction contains either  $X$  or  $Y$  is far bigger than the probability that it contains both  $X$  and  $Y$  or it contains neither  $X$  nor  $Y$ .

**Example 5.5. Null-transaction problem with Definition 5.2.** Given our needle package example, when there are in total 200 transactions in the database, we have

$$\begin{aligned} \text{sup}(A \cup \bar{B}) \times \text{sup}(\bar{A} \cup B) &= 99/200 \times 99/200 \approx 0.245 \\ &\gg \text{sup}(A \cup B) \times \text{sup}(\bar{A} \cup \bar{B}) = 1/200 \times (200 - 199)/200 \approx 0.25 \times 10^{-4}, \end{aligned}$$

which, according to Definition 5.2, indicates that  $A$  and  $B$  are strongly negatively correlated. However, if there are  $10^6$  transactions in the database, the measure would compute

$$\begin{aligned} \text{sup}(A \cup \bar{B}) \times \text{sup}(\bar{A} \cup B) &= 99/10^6 \times 99/10^6 = 9.8 \times 10^{-9} \\ &\ll \text{sup}(A \cup B) \times \text{sup}(\bar{A} \cup \bar{B}) = 1/10^6 \times (10^6 - 199)/10^6 \approx 10^{-6}. \end{aligned}$$

This time, the measure indicates that  $A$  and  $B$  are positively correlated, hence, a contradiction. The measure is not null-invariant.  $\square$

As a third alternative, consider Definition 5.3, which is based on the Kulczynski measure (i.e., the average of conditional probabilities). It follows the spirit of interestingness measures introduced in Section 4.3.3.

**Definition 5.3.** Suppose that itemsets  $X$  and  $Y$  are both frequent, that is,  $\text{sup}(X) \geq \text{min\_sup}$  and  $\text{sup}(Y) \geq \text{min\_sup}$ , where  $\text{min\_sup}$  is the minimum support threshold. If  $(P(X|Y) + P(Y|X))/2 < \epsilon$ , where  $\epsilon$  is a negative pattern threshold, then pattern  $X \cup Y$  is a **negatively correlated pattern**.

**Example 5.6. Negatively correlated patterns using Definition 5.3, based on the Kulczynski measure.** Let's reexamine our needle package example. Let  $\text{min\_sup}$  be 0.01% and  $\epsilon = 0.02$ . When there are 200 transactions in the database, we have  $\text{sup}(A) = \text{sup}(B) = 100/200 = 0.5 > 0.01\%$  and  $(P(B|A) + P(A|B))/2 = (0.01 + 0.01)/2 < 0.02$ ; thus  $A$  and  $B$  are negatively correlated. Does this still hold true if we have many more transactions? When there are  $10^6$  transactions in the database, the measure computes  $\text{sup}(A) = \text{sup}(B) = 100/10^6 = 0.01\% \geq 0.01\%$  and  $(P(B|A) + P(A|B))/2 = (0.01 + 0.01)/2 < 0.02$ , again indicating that  $A$  and  $B$  are negatively correlated. This matches our intuition. The measure does not have the null-invariance problem of the first two definitions considered.

Let's examine another case: Suppose that among 100,000 transactions, the store sold 1000 needle packages of  $A$  but only 10 packages of  $B$ ; however, every time package  $B$  is sold, package  $A$  is also sold

(i.e., they appear in the same transaction). In this case, the measure computes  $(P(B|A) + P(A|B))/2 = (0.01 + 1)/2 = 0.505 \gg 0.02$ , which indicates that  $A$  and  $B$  are positively correlated instead of negatively correlated. This also matches our intuition.  $\square$

With this new definition of negative correlation, efficient methods can easily be derived for mining negative patterns in large databases. This is left as an exercise for interested readers.

---

## 5.2 Mining compressed or approximate patterns

A major challenge in frequent pattern mining is the huge number of discovered patterns. Using a minimum support threshold to control the number of patterns found has limited effect. Too low a value can lead to the generation of an explosive number of output patterns, whereas too high a value can lead to the discovery of only commonsense patterns.

To reduce the huge set of frequent patterns generated in mining while maintaining high-quality patterns, we can instead mine a compressed or approximate set of frequent patterns. *Top-k most frequent patterns* were proposed to make the mining process concentrate on only the set of  $k$  most frequent patterns. Although interesting, they usually do not epitomize the  $k$  most representative patterns because of the uneven frequency distribution among itemsets. *Constraint-based mining* of frequent patterns (Section 5.3) incorporates user-specified constraints to filter out uninteresting patterns. Measures of pattern/rule *interestingness* and *correlation* (Section 5.3) can also be used to help confine the search to patterns/rules of interest.

Recall in the last chapter, we introduced two preliminary forms of “compression” of frequent patterns: *closed pattern*, which is a lossless compression of the set of frequent patterns, and *max-pattern*, which is a lossy compression. In this section, we examine two advanced forms of “compression” of frequent patterns that build on the concepts of closed patterns and max-patterns. Section 5.2.1 explores *clustering-based compression of frequent patterns*, which groups patterns together based on their similarity and frequency support. Section 5.2.2 takes a “*summarization*” approach, where the aim is to derive redundancy-aware top- $k$  representative patterns that cover the whole set of (closed) frequent itemsets. The approach considers not only the representativeness of patterns but also their mutual independence to avoid redundancy in the set of generated patterns. The  $k$  representatives provide compact compression over the collection of frequent patterns, making them easier to interpret and use.

### 5.2.1 Mining compressed patterns by pattern clustering

Pattern compression can be achieved by pattern clustering. Clustering techniques are described in detail in Chapters 8 and 9. In this section, it is not necessary to know the fine details of clustering. Rather, you will learn how the concept of clustering can be applied to compress frequent patterns. Clustering is the automatic process of grouping similar objects together, so that objects within a cluster are similar to one another and dissimilar to objects in other clusters. In this case, the objects are frequent patterns. The frequent patterns are clustered using a tightness measure called  $\delta$ -cluster. A representative pattern is selected for each cluster, thereby offering a compressed version of the set of frequent patterns.

Before we begin, let’s review some definitions. An itemset  $X$  is a **closed frequent itemset** in a data set  $D$  if  $X$  is frequent and there exists no proper superitemset  $Y$  of  $X$  such that  $Y$  has the same support

ID	Itemsets	Support
$P_1$	$\{b, c, d, e\}$	205,227
$P_2$	$\{b, c, d, e, f\}$	205,211
$P_3$	$\{a, b, c, d, e, f\}$	101,758
$P_4$	$\{a, c, d, e, f\}$	161,563
$P_5$	$\{a, c, d, e\}$	161,576

count as  $X$  in  $D$ . An itemset  $X$  is a **maximal frequent itemset** in data set  $D$  if  $X$  is frequent, and there exists no superitemset  $Y$  such that  $X \subset Y$  and  $Y$  is frequent in  $D$ . Using these concepts alone is not enough to obtain a good representative compression of a data set, as we see in Example 5.7.

**Example 5.7. Shortcomings of closed itemsets and maximal itemsets for compression.** Table 5.2 shows a subset of frequent itemsets on a large data set, where  $a, b, c, d, e, f$  represent individual items. There is no nonclosed itemset here; therefore we cannot use closed frequent itemsets to compress the data. The only maximal frequent itemset is  $P_3$ . However, we observe that itemsets  $P_2, P_3$ , and  $P_4$  are significantly different with respect to their support counts. If we were to use  $P_3$  to represent a compressed version of the data, we would lose this support count information entirely. Consider the two pairs  $(P_1, P_2)$  and  $(P_4, P_5)$ . From visual inspection, the patterns within each pair are very similar with respect to their support and expression. Therefore intuitively,  $P_2, P_3$ , and  $P_4$ , collectively, should serve as a better compressed version of the data.  $\square$

Let's see if we can find a way of clustering frequent patterns as a means of obtaining a compressed representation of them. We will need to define a good similarity measure, cluster patterns according to this measure, and then select and output only a *representative pattern* for each cluster. Since the set of closed frequent patterns is a lossless compression over the original frequent patterns set, it is a good idea to discover representative patterns around the collection of *approximately closed* patterns.

We can use the following distance measure between closed patterns. Let  $P_1$  and  $P_2$  be two closed patterns. Their supporting transaction sets are  $T(P_1)$  and  $T(P_2)$ , respectively. The **pattern distance** of  $P_1$  and  $P_2$ ,  $Pat\_Dist(P_1, P_2)$ , is defined as

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}. \quad (5.7)$$

Pattern distance is a distance metric defined on the set of transactions. It incorporates the *support* information of patterns, as desired previously.

**Example 5.8. Pattern distance.** Suppose  $P_1$  and  $P_2$  are two patterns such that  $T(P_1) = \{t_1, t_2, t_3, t_4, t_5\}$  and  $T(P_2) = \{t_1, t_2, t_3, t_4, t_6\}$ , where  $t_i$  is a transaction in the database. The distance between  $P_1$  and  $P_2$  is  $Pat\_Dist(P_1, P_2) = 1 - \frac{4}{6} = \frac{1}{3}$ .  $\square$

Now, let's consider the *expression* of patterns. Given two patterns  $A$  and  $B$ , we say  $B$  can be **expressed** by  $A$  if  $O(B) \subset O(A)$ , where  $O(A)$  is the corresponding itemset of pattern  $A$ . Following this definition, assume patterns  $P_1, P_2, \dots, P_k$  are in the same cluster. The representative pattern

$P_r$  of the cluster should be able to *express* all the other patterns in the cluster. Clearly, we have  $\bigcup_{i=1}^k O(P_i) \subseteq O(P_r)$ .

Using the distance measure, we can simply apply a clustering method, such as  $k$ -means (Section 9.2), on the collection of frequent patterns. However, this introduces two problems. First, the quality of the clusters cannot be guaranteed; second, it may not be able to find a representative pattern for each cluster (i.e., the pattern  $P_r$  may not belong to the same cluster). To overcome these problems, this is where the concept of  $\delta$ -cluster comes in, where  $\delta$  ( $0 \leq \delta \leq 1$ ) measures the tightness of a cluster.

A pattern  $P$  is  **$\delta$ -covered** by another pattern  $P'$  if  $O(P) \subseteq O(P')$  and  $Pat\_Dist(P, P') \leq \delta$ . A set of patterns form a  **$\delta$ -cluster** if there exists a representative pattern  $P_r$  such that for each pattern  $P$  in the set,  $P$  is  $\delta$ -covered by  $P_r$ .

Note that according to the concept of  $\delta$ -cluster, a pattern can belong to multiple clusters. Also, using  $\delta$ -cluster, we only need to compute the distance between each pattern and the representative pattern of the cluster. Because a pattern  $P$  is  $\delta$ -covered by a representative pattern  $P_r$  only if  $O(P) \subseteq O(P_r)$ , we can simplify the distance calculation by considering only the supports of the patterns:

$$Pat\_Dist(P, P_r) = 1 - \frac{|T(P) \cap T(P_r)|}{|T(P) \cup T(P_r)|} = 1 - \frac{|T(P_r)|}{|T(P)|}. \quad (5.8)$$

If we restrict the representative pattern to be frequent, then the number of representative patterns (i.e., clusters) is no less than the number of maximal frequent patterns. This is because a maximal frequent pattern can only be covered by itself. To achieve more succinct compression, we relax the constraints on representative patterns, that is, we allow the support of representative patterns to be *somewhat* less than  $min\_sup$ .

For any representative pattern  $P_r$ , assume its support is  $k$ . Since it has to *cover* at least one frequent pattern (i.e.,  $P$ ) with support that is at least  $min\_sup$ , we have

$$\delta \geq Pat\_Dist(P, P_r) = 1 - \frac{|T(P_r)|}{|T(P)|} \geq 1 - \frac{k}{min\_sup}. \quad (5.9)$$

That is,  $k \geq (1 - \delta) \times min\_sup$ . This is the minimum support for a representative pattern, denoted as  $min\_sup_r$ .

Based on the preceding discussion, the pattern compression problem can be defined as follows: *Given a transaction database, a minimum support  $min\_sup$ , and the cluster quality measure  $\delta$ , the pattern compression problem is to find a set of representative patterns  $R$  such that for each frequent pattern  $P$  (with respect to  $min\_sup$ ), there is a representative pattern  $P_r \in R$  (with respect to  $min\_sup_r$ ), which covers  $P$ , and the value of  $|R|$  is minimized.*

Finding a minimum set of representative patterns is an NP-Hard problem. However, efficient methods have been developed that reduce the number of closed frequent patterns generated by orders of magnitude with respect to the original collection of closed patterns. The methods succeed in finding a high-quality compression of the pattern set.

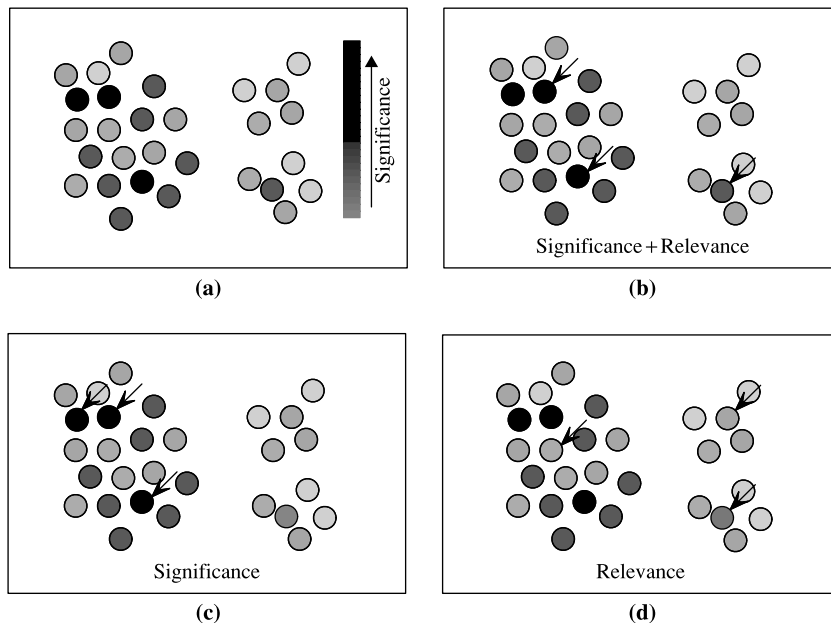
### 5.2.2 Extracting redundancy-aware top- $k$ patterns

Mining the top- $k$  most frequent patterns is a strategy for reducing the number of patterns returned during mining. However, in many cases, frequent patterns are not mutually independent but often clustered in

small regions. This is somewhat like finding 20 population centers in the world, which may result in cities clustered in a small number of countries rather than evenly distributed across the globe. Instead, most users would prefer to derive the  $k$  most interesting patterns, which are not only significant but also mutually independent and containing little redundancy. A small set of  $k$  representative patterns that have not only high significance but also low redundancy are called **redundancy-aware top- $k$  patterns**.

**Example 5.9. Redundancy-aware top- $k$  strategy vs. other top- $k$  strategies.** Fig. 5.7 illustrates the intuition behind *redundancy-aware top- $k$  patterns* vs. *traditional top- $k$  patterns* and  *$k$ -summarized patterns*. Suppose we have the frequent patterns set shown in Fig. 5.7(a), where each circle represents a pattern of which the significance is colored in grayscale. The distance between two circles reflects the redundancy of the two corresponding patterns: The closer the circles are, the more redundant the respective patterns are to one another. Let's say we want to find three patterns that will best represent the given set, that is,  $k = 3$ . Which three should we choose?

Arrows are used to show the patterns chosen if using redundancy-aware top- $k$  patterns (Fig. 5.7b), traditional top- $k$  patterns (Fig. 5.7c), or  $k$ -summarized patterns (Fig. 5.7d). In Fig. 5.7(c), the **traditional top- $k$  strategy** relies solely on significance: It selects the three most significant patterns to represent the set.



**FIGURE 5.7**

Conceptual view comparing top- $k$  methodologies (where gray levels represent pattern significance, and the closer that two patterns are displayed, the more redundant they are to one another): (a) original patterns, (b) redundancy-aware top- $k$  patterns, (c) traditional top- $k$  patterns, and (d)  $k$ -summarized patterns.

In Fig. 5.7(d), the ***k*-summarized pattern strategy** selects patterns based solely on nonredundancy. It detects three clusters and finds the most representative patterns to be the “centermost” pattern from each cluster. These patterns are chosen to represent the data. The selected patterns are considered “summarized patterns” in the sense that they represent or “provide a summary” of the clusters they stand for.

By contrast, in Fig. 5.7(b) the **redundancy-aware top-*k* patterns** make a trade-off between significance and redundancy. The three patterns chosen here have high significance and low redundancy. Observe, for example, the two highly significant patterns that, based on their redundancy, are displayed next to each other. The redundancy-aware top-*k* strategy selects only one of them, taking into consideration that two would be redundant. To formalize the definition of redundancy-aware top-*k* patterns, we need to define the concepts of significance and redundancy.  $\square$

A **significance measure**  $S$  is a function mapping a pattern  $p \in \mathcal{P}$  to a real value such that  $S(p)$  is the degree of interestingness (or usefulness) of the pattern  $p$ . In general, significance measures can be either objective or subjective. *Objective measures* depend only on the structure of the given pattern and the underlying data used in the discovery process. Commonly used objective measures include support, confidence, correlation, and *tf-idf* (or *term frequency vs. inverse document frequency*), where the latter is often used in information retrieval. *Subjective measures* are based on user beliefs in the data. They therefore depend on the users who examine the patterns. A subjective measure is usually a relative score based on user prior knowledge or a background model. It often measures the unexpectedness of a pattern by computing its divergence from the background model. Let  $S(p, q)$  be the **combined significance** of patterns  $p$  and  $q$ , and  $S(p|q) = S(p, q) - S(q)$  be the **relative significance** of  $p$  given  $q$ . Note that the combined significance,  $S(p, q)$ , means the collective significance of two individual patterns  $p$  and  $q$ , not the significance of a single super pattern  $p \cup q$ .

Given the significance measure  $S$ , the **redundancy  $R$  between two patterns**  $p$  and  $q$  is defined as  $R(p, q) = S(p) + S(q) - S(p, q)$ . Subsequently, we have  $S(p|q) = S(p) - R(p, q)$ .

We assume that the combined significance of two patterns is no less than the significance of any individual pattern (since it is a collective significance of two patterns) and does not exceed the sum of two individual significance patterns (since there exists redundancy). That is, the redundancy between two patterns should satisfy

$$0 \leq R(p, q) \leq \min(S(p), S(q)). \quad (5.10)$$

The ideal redundancy measure  $R(p, q)$  is usually hard to obtain. However, we can approximate redundancy using distance between patterns such as with the distance measure defined in Section 5.2.1.

The problem of finding redundancy-aware top-*k* patterns can thus be transformed into finding a *k*-pattern set that maximizes the marginal significance, which is a well-studied problem in information retrieval. In this field, a document has high marginal relevance if it is both relevant to the query and contains minimal marginal similarity to previously selected documents, where the marginal similarity is computed by choosing the most relevant selected document. The detailed computational method is omitted here. Experimental studies have shown that the computation based on this principle is efficient and is able to find high-significance and low-redundancy top-*k* patterns.

---

## 5.3 Constraint-based pattern mining

A pattern mining process may uncover thousands of patterns from a given data set, many of which may end up being unrelated or uninteresting to users. Often, a user has a good sense of which “direction” of

mining may lead to interesting patterns and the “form” of the patterns or rules they want to find. They may also have a sense of “conditions” for the rules, which would eliminate the discovery of certain rules that they know would not be of interest. Thus a good option is to have users specify such intuition or expectations as *constraints* to confine the search space or perform constraint refinement interactively based on the intermediate mining results. This strategy is known as **constraint-based mining**. The constraints can include the following:

- **Knowledge type constraints:** These specify the type of knowledge to be mined, such as association, correlation, classification, or clustering.
- **Data constraints:** These specify the set of task-relevant data.
- **Dimension/level constraints:** These specify the desired dimensions (or attributes) of the data, the abstraction levels, or the level of the concept hierarchies to be used in mining.
- **Interestingness constraints:** These specify thresholds on statistical measures of rule interestingness such as support, confidence, and correlation.
- **Rule/pattern constraints:** These specify the form of, or conditions on, the rules/patterns to be mined. Such constraints may be expressed as metarules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates.

These constraints can be specified using a high-level data mining query language or a template-based graphical user interface.

The first four constraint types have already been addressed earlier in the book. In this section, we discuss the use of *rule/pattern constraints* to focus on the mining task. This form of constraint-based mining allows users to describe the rules or patterns that they would like to uncover, thereby making the data mining process more *effective*. In the meantime, a sophisticated mining query optimizer can be used to exploit the constraints specified by the user, thereby making the mining process more *efficient*.

In some cases, a user may like to specify some syntactic form of rules (also called *metarules*) that she is interested in mining. Such syntactic forms help the user to express her expectation and also help the system to confine search space and improve mining efficiency.

For example, a metarule can be in the form of

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow \text{buys}(X, \text{“iPad”}), \quad (5.11)$$

where  $P_1$  and  $P_2$  are **predicate variables** that can be instantiated to attributes in a given database during the mining process,  $X$  is a variable representing a customer, and  $Y$  and  $W$  take on values of the attributes assigned to  $P_1$  and  $P_2$ , respectively. Typically, a user can specify a list of attributes to be considered for instantiation with  $P_1$  and  $P_2$ . Otherwise, a default set may be used.

A metarule forms a hypothesis regarding the relationships that the user is interested in probing or confirming. Following such a template, a system can then mine concrete rules that match the given metarule. Possibly, Rule (5.12) that complies with Metarule (5.11) will be returned as mining results

$$\text{age}(X, \text{“20..29”}) \wedge \text{income}(X, \text{“41K..60K”}) \Rightarrow \text{buys}(X, \text{“iPad”}). \quad (5.12)$$

In order to generate interesting and useful mining results, users may have multiple ways to specify rule/pattern constraints. It is desirable for a mining system to use rule/pattern constraints to prune the search space, that is, to push such constraints deeply into the mining process while still ensure the

completeness of the answer returned for a mining query. However, this is a nontrivial task, and its study leads to *constraint-based pattern mining*.

To study how to use constraints at mining frequent patterns or association rules, we examine the following running example.

**Example 5.10. Constraints on shopping transaction mining.** Suppose that a multidimensional shopping transaction database contains the following interrelated relations:

- $item(item\_ID, item\_name, description, category, price)$
- $sales(transaction\_ID, day, month, year, store\_ID, city)$
- $trans\_item(item\_ID, transaction\_ID)$

Here, the *item* table contains attributes *item\_ID*, *item\_name*, *description*, *category*, and *price*; the *sales* table contains attributes *transaction\_ID*, *day*, *month*, *year*, *store\_ID*, and *city*; and the two tables are linked via the foreign key attributes, *item\_ID* and *transaction\_ID*, in the table *trans\_item*.

A mining query may contain multiple constraints. For example, we may have a query: “From the sales in Chicago in 2020, find the patterns (i.e., item sets) that which cheap items (where the sum of the prices is less than \$10) appear in the same transaction with (hence may promote) which expensive items (where the minimum price is \$50).”

This query contains the following four constraints: (1)  $sum(I.price) < \$10$ , where *I* represents the *item\_ID* of a cheap item; (2)  $min(J.price) \geq \$50$ , where *J* represents the *item\_ID* of an expensive item; (3)  $T.city = Chicago$ ; and (4)  $T.year = 2020$ , where *T* represents a *transaction\_ID*. □

In constraint-based pattern mining, the search space can be pruned in the mining process with two strategies: *pruning pattern search space* and *pruning data search space*. The former checks candidate patterns and decides whether a pattern should be eliminated from further processing. For example, it may prune a pattern if all of its superpattern will be useless in the remaining mining process, say, based on the Apriori property. The latter checks the data set to determine whether a particular data object will not be able to contribute to the subsequent generation of satisfiable patterns in the remaining mining process (hence safely pruning the data object).

We examine these pruning strategies in the following subsections.

### 5.3.1 Pruning pattern space with pattern pruning constraints

Based on how a constraint may interact with the pattern mining process, we partition pattern mining constraints into four categories: (1) *antimonotonic*, (2) *monotonic*, (3) *convertible*, and (4) *nonconvertible*. Let’s examine them one by one.

#### **Pattern antimonotonicity**

The first group of constraints are characterized with **pattern antimonotonicity**. A constraint *C* is **pattern antimonotonic** if it has the following property: *If an itemset does not satisfy constraint C, none of its supersets will satisfy C.*

Let’s examine a constraint “ $C_1 : sum(I.price) \leq \$100$ ” and see what may happen if the constraint is added to our shopping transaction mining query. Suppose we are mining itemsets of size *k* at the *k*th iteration using the Apriori algorithm or the like. If the summation of the prices of the items in a candidate itemset  $S_1$  is greater than \$100, this itemset should be pruned from the search space, since not only the



current set cannot satisfy the constraint, but also adding more items into the set (assuming that the price of any item is no less than zero) will never be able to satisfy the constraint. Notice that the pruning of this pattern (frequent itemset) for constraint  $C_1$  is not confined to the Apriori candidate-generation-and-test framework. For example, for the same reason,  $S_1$  should be pruned in the pattern-growth framework since pattern  $S_1$  and the further growth from it can never make constraint  $C_1$  satisfiable.

This property is called *antimonotonicity* because *monotonicity* of a constraint usually means *if a pattern  $p$  satisfies a constraint  $C$ , its further expansion will always satisfy  $C$* ; however, here we claim that this constraint may have a *reverse behavior*: *once a pattern  $p_1$  violates the constraint  $C_1$ , its further growth (or expansion) will always violate  $C_1$* . Pattern pruning by antimonotonicity can be applied at each iteration of Apriori-style algorithms to help improve the efficiency of the overall mining process while guaranteeing the completeness of the data mining task.

It is interesting to note that the very basic Apriori property itself (which states that all nonempty subsets of a frequent itemset must also be frequent) is antimonotonic: *If an itemset does not satisfy the minimum support threshold, none of its supersets can*. This property has been used at each iteration of the Apriori algorithm to reduce the number of candidate itemsets to be examined, thereby reducing the search space for frequent pattern mining.

There are many constraints that are antimonotonic. For example, the constraint “ $\min(J.\text{price}) \geq \$50$ ,” and “ $\text{count}(I) \leq 10$ ,” are antimonotonic. However, there are also many constraints that are not antimonotonic. For example, the constraint “ $\text{avg}(I.\text{price}) \leq \$10$ ” is not antimonotonic. This is because even for a given itemset  $S$  that does not satisfy this constraint, a superset created by adding some (cheap) items may make it satisfy the constraint. Hence, pushing this constraint inside the mining process will not guarantee the completeness of the data mining process. A list of popularly encountered constraints is given in the first column of Table 5.3. The antimonotonicity of the constraints is indicated in the second column. To simplify our discussion, only existence operators (e.g.,  $=$ ,  $\in$ , but not  $\neq$ ,  $\notin$ ) and comparison (or containment) operators with equality (e.g.,  $\leq$ ,  $\subseteq$ ) are given.

### Pattern monotonicity

The second category of constraints is **pattern monotonicity**. A constraint  $C$  is **pattern monotonic** if it has the following property: *If an itemset satisfies constraint  $C$ , all of its supersets will satisfy  $C$* .

Let’s examine another constraint “ $C_2 : \text{sum}(I.\text{price}) \geq \$100$ ” and see what may happen if the constraint is added to our example query. Suppose we are mining itemsets of size  $k$  at the  $k$ th iteration using the Apriori algorithm or the like. If the summation of the prices of the items in a candidate itemset  $S_1$  is less than \$100, this itemset should not be pruned from the search space, since adding more items to the current set may make the itemset satisfy the constraint. However, once the sum of the prices of the items in itemset  $S$  satisfies the constraint  $C_2$ , there is no need to check this constraint for  $S$  any more since adding more items will not decrease the sum value and will always satisfy the constraint. In other words, if an itemset satisfies the constraint, so do all of its supersets. Please note that the property is independent of particular iterative pattern mining algorithms. For example, the same pruning methodology should be adopted for pattern-growth algorithms as well.

There are many pattern monotonic constraints in practice. For example, “ $\min(I.\text{price}) \leq \$10$ ” and “ $\text{count}(I) \geq 10$ ” are such constraints. The pattern monotonicity of the list of frequently encountered constraints is indicated in the third column of Table 5.3.

**Table 5.3 Characterization of commonly used pattern pruning constraints.**

Constraint	Antimonotonic	Monotonic	Succinct
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	no	yes
$\min(S) \leq v$	no	yes	yes
$\min(S) \geq v$	yes	no	yes
$\max(S) \leq v$	yes	no	yes
$\max(S) \geq v$	no	yes	yes
$\text{count}(S) \leq v$	yes	no	no
$\text{count}(S) \geq v$	no	yes	no
$\text{sum}(S) \leq v (\forall a \in S, a \geq 0)$	yes	no	no
$\text{sum}(S) \geq v (\forall a \in S, a \geq 0)$	no	yes	no
$\text{range}(S) \leq v$	yes	no	no
$\text{range}(S) \geq v$	no	yes	no
$\text{avg}(S) \theta v, \theta \in \{\leq, \geq\}$	convertible	convertible	no
$\text{support}(S) \geq \xi$	yes	no	no
$\text{support}(S) \leq \xi$	no	yes	no
$\text{all\_confidence}(S) \geq \xi$	yes	no	no
$\text{all\_confidence}(S) \leq \xi$	no	yes	no

### **Convertible constraints: ordering data in transactions**

There are constraints that are neither pattern antimonotonic nor pattern monotonic. For example, it is hard to directly push the constraint “ $C_3 : \text{avg}(I.\text{price}) \leq \$10$ ” deeply into an iterative mining process because the next item to be added to the current itemset can be more expensive or less expensive than the average price of the itemset computed so far. At the first glance, it seems to be hard to explore constraint pushing for such kind of constraints in pattern mining. However, observing that the items in a transaction can be treated as a set, and thus it is possible to arrange items in a transaction in any specific ordering. Interestingly, when the items in the itemset are arranged in a price ascending or descending order, it is possible to explore efficient pruning in frequent itemset mining as we did before. In this context, it is possible to convert such kind of constraints into monotonic or antimonotonic constraints. Hence we call such constraints as **convertible constraints**.

Let’s re-examine the constraint  $C_3$ . If the items in all the transactions are sorted in the price-ascending order (or items in any transaction are added in this order) in the pattern-growth mining process, the constraint  $C_3$  becomes *antimonotonic*, because if an itemset  $I$  violates the constraint (i.e., with an average price greater than \$10), then further addition of more expensive items into the itemset will never make it satisfy the constraint. Similarly, if items in all the transaction are sorted (or being added to the itemset being mined) in the price-descending order, it becomes *monotonic*, because if the itemset satisfies the constraint (i.e., with an average price no greater than \$10), then adding cheaper items into the current itemset will still make the average price no greater than \$10.

*Will the Apriori-like algorithm make good use of the convertible constraint to prune its search space?* Unfortunately, such a constraint satisfaction checking cannot be done easily with an Apriori-like

candidate-generation-and-test algorithm, because an Apriori-like algorithm requires all of the subsets (say,  $\{ab\}$ ,  $\{bc\}$ ,  $\{ac\}$ ) of a candidate  $\{abc\}$  must be frequent and satisfies the constraint. However, even  $\{abc\}$  itself could be a valid itemset (i.e.,  $avg(\{abc\}.price) \leq \$10$ ), the subset  $\{bc\}$  may have violated  $C_3$ , and we will never be able to generate  $\{abc\}$  since  $\{bc\}$  has been pruned.

Let  $S$  represent a set of items and its value be *price*. Besides “ $avg(S) \leq c$ ” and “ $avg(S) \geq c$ ,” there are also other convertible constraints. For example, “ $variance(S) \geq c$ ,” “ $standard\_deviation(S) \geq c$ ” are convertible constraints. However, this does not imply that every nonmonotonic or nonantimonotonic constraint is convertible. For example, if the aggregation function for item values in the set has random sampling behavior, it will be hard to arrange the items in a monotonically increasing or decreasing order. Therefore, there still exists a category of constraints that are **nonconvertible**. The good news is that although there exist some tough constraints that are not convertible, most simple and frequently used constraints belong to one of the three categories we just described, antimonotonic, monotonic, and convertible, to which efficient constraint mining methods can be applied.

### 5.3.2 Pruning data space with data pruning constraints

The second way of search space pruning in constraint-based frequent pattern mining is *pruning data space*. This strategy prunes pieces of data if they will not contribute to the subsequent generation of satisfiable patterns in the mining process. We examine *data antimonotonicity* in this section.

Interestingly, many constraints are **data-antimonotonic** in the sense that *during the mining process*, if a data entry cannot satisfy a data-antimonotonic constraint based on the current pattern, then it can be pruned. We prune it because it will not be able to contribute to the generation of any superpattern of the current pattern in the remaining mining process.

**Example 5.11. Data antimonotonicity.** We examine constraint  $C_1 : sum(I.price) \geq \$100$ , that is, the sum of the prices of the items in the mined pattern must be no less than \$100. Suppose that the current frequent itemset,  $S$ , does not satisfy constraint  $C_1$  (say, because the sum of the prices of the items in  $S$  is \$50). If the remaining frequent items in a transaction  $T_i$  cannot make  $S$  satisfy the constraint (e.g., the remaining frequent items in  $T_i$  are  $\{i_2.price = \$5, i_5.price = \$10, i_8.price = \$20\}$ ), then  $T_i$  cannot contribute to the patterns to be mined from  $S$ , and can be pruned from further mining.

Note that such pruning may not be effective by enforcing it only at the beginning of the mining process. This is because it may prune those transactions whose sum of items do not satisfy the constraint  $C_1$ . However, we may encounter a case that  $i_3.price = \$90$ , but later in the mining process,  $i_3$  becomes infrequent with  $S$  in the transaction data set, and at this point,  $T_i$  should be pruned. Therefore such checking and pruning should be enforced at each iteration to reduce the data search space.  $\square$

Notice that constraint  $C_1$  is a monotonic constraint with respect to pattern space pruning. As we have seen, this pattern monotonic constraint has very limited power for reducing the search space in pattern pruning. However, the same constraint is data antimonotonic and can be used for effective reduction of the data search space.

For a pattern antimonotonic constraint, such as  $C_2 : sum(I.price) \leq \$100$ , we can prune both pattern and data search spaces at the same time. Based on our study of pattern pruning, we already know that the current itemset can be pruned if the sum of the prices in it is over \$100 (since its further expansion can never satisfy  $C_2$ ). At the same time, we can also prune any remaining items in a transaction  $T_i$  that cannot make the constraint  $C_2$  valid. For example, if the sum of the prices of items in the current

itemset  $S$  is \$90, any item with price over \$10 in the remaining frequent items in  $T_i$  can be pruned. If none of the remaining items in  $T_i$  can make the constraint valid, the entire transaction  $T_i$  should be pruned.

Consider pattern constraints that are neither antimonotonic nor monotonic such as “ $C_3 : avg(I.price) \leq 10$ .” These can be data-antimonotonic because if the remaining items in a transaction  $T_i$  cannot make the constraint valid, then  $T_i$  can be pruned as well. Therefore data-antimonotonic constraints can be quite useful for constraint-based data space pruning.

Notice that search space pruning by data antimonotonicity is confined only to a pattern growth-based mining algorithm because the pruning of a data entry is determined based on whether it can contribute to a specific pattern. Data antimonotonicity cannot be used for pruning the data space if the Apriori algorithm is used because the data are associated with all of the currently active patterns. At any iteration, there are usually many active patterns. A data entry that cannot contribute to the formation of the superpatterns of a given pattern may still be able to contribute to the superpattern of other active patterns. Thus, the power of data space pruning can be very limited for nonpattern growth-based algorithms.

### 5.3.3 Mining space pruning with succinctness constraints

For pattern mining, there is another category of constraints called *succinct constraints*. A constraint  $c$  is **succinct** if it can be enforced by *directly pruning some data objects from the database* or by *directly enumerating all and only those sets that are guaranteed to satisfy the constraint*. The former is called **data succinct** since it enables direct data space pruning, whereas the latter is called **pattern succinct** since it enables direct pattern generation by starting with initial patterns that satisfy the constraint. Let’s examine a few examples.

First, let’s examine the constraint  $i \in S$ , that is, the pattern must contain item  $i$ . To find the patterns containing item  $i$ , one can mine only  $i$ -projected database since a transaction does not contain  $i$  will not contribute to the patterns containing  $i$ , and for those containing  $i$ , all the remaining items can participate the remaining of the mining process. This facilitates data space pruning at the beginning and thus this constraint is both data and pattern succinct. On the other hand, to find the patterns that do not contain item  $i$  (i.e.,  $i \notin S$ ), one can mine it by mining the transaction database with  $i$  removed since  $i$  in a transaction will not contribute to the pattern. This facilitates data space pruning at the beginning and also facilitate pattern space pruning (since it avoids mine any intermediate patterns containing  $i$ , thus the constraint is succinct, and it is both pattern succinct and data succinct).

As another example, a constraint “ $min(S.price) \geq \$50$ ” is data succinct since we can remove all items whose price is less than \$50 from the transactions since any item whose price is no less than \$50 will not contribute to the pattern mining process. Similarly,  $min(S.Price) \leq v$  is pattern succinct since we can start with only those items whose price is no greater than  $v$ .

Notice that not all the constraints are succinct. For example, the constraint  $sum(S.Price) \geq v$  is not succinct because it cannot be used to facilitate the pruning of any item from a transaction at the beginning of the process since the sum of the price of an itemset  $S$  will keep increasing.

The pattern succinctness of the list of SQL primitives-based constraints is indicated in the fourth column of Table 5.3.

From the above discussion, we can see that the same constraint may belong to more than one category. For example, the constraint “ $min(I.price) \leq \$10$ ” is pattern monotonic and also data succinct. In

this case, we can use data succinctness to start only with those items whose price is no more than \$10. By doing so, it has implicitly applied pattern monotonicity property already since once the constraint is used at the starting point (i.e., satisfied), we will not need to check it any more. As another example, the constraint “ $c_0 : \text{sum}(I.\text{price}) \geq \$100$ ” is both pattern monotonic and data antimonotonic, we can use the data antimonotonicity to prune those transactions whose prices of the remaining items adding together cannot reach \$100. In the meantime, once a pattern satisfies  $c_0$ , we will not need to check  $c_0$  again in the mining process.

In applications, a user may pose a mining query that may contain multiple constraints. In many cases, multiple constraints can be enforced together to jointly prune mining space, which may lead to more efficient processing. However, in some cases, different constraints may require different item-ordering for the effective constraint enforcement, especially for convertible constraints. For example, a query may contain both  $c_1 : \text{avg}(S.\text{profit}) > 20$  and  $c_2 : \text{avg}(S.\text{price}) < 50$ . Unfortunately, sorting on profit in value-descending order may not result in value-descending order of their associated item price. In this case, it is the best to estimate which ordering may lead to more effective pruning, and mining following the more effective pruning ordering will lead to more efficient processing. Suppose it is hard to find patterns satisfying  $c_1$  but easy to find pattern satisfying  $c_2$ . Then the system should sort the items in transactions in profit descending ordering. Once the average profit of the current itemset drops to below \$20, the itemset can be tossed (i.e., no further mining with it), which will lead to efficient processing.

---

## 5.4 Mining sequential patterns

A **sequence database** consists of sequences of ordered elements or events, recorded with or without a concrete notion of time. There are many applications involving sequence data. Typical examples include customer shopping sequences, Web clickstreams, biological sequences, and sequences of events in science and engineering, and in natural and social developments. In this section, we study *sequential pattern mining* in transactional databases, and with proper extensions, such mining algorithms can help find sequential patterns for many other applications, such as finding sequential patterns for Webclick streams, and for science, engineering, and social event mining. We start with the basic concepts of sequential pattern mining in Section 5.4.1. Section 5.4.2 presents several scalable methods for such mining. We will discuss constraint-based sequential pattern mining in Section 5.4.3.

### 5.4.1 Sequential pattern mining: concepts and primitives

“*What is sequential pattern mining?*” **Sequential pattern mining** is the mining of frequently occurring ordered events or subsequences as patterns. An example of a sequential pattern is “*Customers who buy an iPad Pro are likely to buy an Apple pencil within 90 days.*” For retail data, sequential patterns are useful for shelf placement and promotions. This industry, as well as telecommunications and other businesses, may also use sequential patterns for targeted marketing, customer retention, and many other tasks. Other areas in which sequential patterns can be applied include Web access pattern analysis, production processes, and network intrusion detection. Notice that most studies of sequential pattern mining concentrate on *categorical* or *symbolic patterns*, whereas numerical curve analysis usually belongs to the scope of trend analysis and forecasting in statistical time-series analysis discussed in many statistics or time-series analysis textbooks.

The sequential pattern mining problem was first introduced by Agrawal and Srikant in 1995 based on their study of customer purchase sequences, as follows: *Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items, and given a user-specified minimum support threshold of  $min\_sup$ , sequential pattern mining finds all **frequent** subsequences, that is, the subsequences whose occurrence frequency in the set of sequences is no less than  $min\_sup$ .*

Let's establish some vocabulary for our discussion of sequential pattern mining. Let  $\mathcal{I} = \{I_1, I_2, \dots, I_p\}$  be the set of all *items*. An **itemset** is a nonempty set of items. A **sequence** is an ordered list of **events**. A sequence  $s$  is denoted  $\langle e_1 e_2 e_3 \dots e_l \rangle$ , where event  $e_1$  occurs before  $e_2$ , which occurs before  $e_3$ , and so on. Event  $e_j$  is also called an **element** of  $s$ . In the case of customer purchase data, an event refers to a shopping trip in which a customer bought items at a certain store. The event is thus an itemset, that is, an unordered list of items that the customer purchased during the trip. The itemset (or event) is denoted as  $(x_1 x_2 \dots x_q)$ , where  $x_k$  is an item. For brevity, the brackets are omitted if an element has only one item, that is, element  $(x)$  is written as  $x$ . Suppose that a customer made several shopping trips to the store. These ordered events form a sequence for the customer. That is, the customer first bought the items in  $e_1$ , then later bought the items in  $e_2$ , and so on. An item can occur at most once in an event of a sequence,<sup>2</sup> but can occur multiple times in different events of a sequence. The number of instances of items in a sequence is called the **length** of the sequence. A sequence with length  $l$  is called an  **$l$ -sequence**. A sequence  $\alpha = \langle a_1 a_2 \dots a_n \rangle$  is called a **subsequence** of another sequence  $\beta = \langle b_1 b_2 \dots b_m \rangle$ , and  $\beta$  is a **supersequence** of  $\alpha$ , denoted as  $\alpha \sqsubseteq \beta$ , if there exist integers  $1 \leq j_1 < j_2 < \dots < j_n \leq m$  such that  $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$ . For example, if  $\alpha = \langle (ab), d \rangle$  and  $\beta = \langle (abc), (de) \rangle$  where  $a, b, c, d$ , and  $e$  are items, then  $\alpha$  is a subsequence of  $\beta$  and  $\beta$  is a supersequence of  $\alpha$ .

A **sequence database**,  $S$ , is a set of tuples,  $\langle SID, s \rangle$ , where  $SID$  is a *sequence\_ID* and  $s$  is a sequence. For our example,  $S$  contains sequences for all customers of the store. A tuple  $\langle SID, s \rangle$  is said to **contain** a sequence  $\alpha$ , if  $\alpha$  is a subsequence of  $s$ . The **support** of a sequence  $\alpha$  in a sequence database  $S$  is the number of tuples in the database containing  $\alpha$ , that is,  $support_S(\alpha) = |\{(SID, s) | (SID, s) \in S \wedge (\alpha \sqsubseteq s)\}|$ . It can be denoted as  $support(\alpha)$  if the sequence database is clear from the context. Given a positive integer  $min\_sup$  as the **minimum support threshold**, a sequence  $\alpha$  is **frequent** in sequence database  $S$  if  $support_S(\alpha) \geq min\_sup$ . That is, for sequence  $\alpha$  to be frequent, it must occur at least  $min\_sup$  times in  $S$ . A *frequent sequence* is called a **sequential pattern**. A sequential pattern with length  $l$  is called an  **$l$ -pattern**. The following example illustrates these concepts.

**Example 5.12. Sequential patterns.** Consider the sequence database,  $S$ , given in Table 5.4, which will be used in examples throughout this section. Let  $min\_sup = 2$ . The set of *items* in the database is  $\{a, b, c, d, e, f, g\}$ . The database contains four sequences.

Let's have a close look at *sequence 1*, which is  $\langle a(abc)(ac)d(cf) \rangle$ . It has five *events*, namely  $(a)$ ,  $(abc)$ ,  $(ac)$ ,  $(d)$  and  $(cf)$ , which occur in the order listed. Items  $a$  and  $c$  each appear more than once in different events of the sequence. There are nine instances of items in sequence 1. Therefore it has a *length* of nine and is called a *9-sequence*. Item  $a$  occurs three times in sequence 1 and so contributes three to the length of the sequence. However, the entire sequence contributes only one to the *support*

<sup>2</sup> We simplify our discussion here in the same spirit as frequent itemset mining, but the developed method can be extended to consider multiple identical items.

Sequence_ID	Sequence
1	$\langle a(abc)(ac)d(cf) \rangle$
2	$\langle (ad)c(bc)(ae) \rangle$
3	$\langle (ef)(ab)(df)cb \rangle$
4	$\langle eg(af)cbc \rangle$

of  $\langle a \rangle$ . Sequence  $\langle a(bc)df \rangle$  is a *subsequence* of sequence 1 since the events of the former are each subsets of events in sequence 1, and the order of events is preserved. Consider subsequence  $s = \langle (ab)c \rangle$ . Looking at the sequence database,  $S$ , we see that sequences 1 and 3 are the only ones that *contain* the subsequence  $s$ . The support of  $s$  is thus 2, which satisfies minimum support. Therefore  $s$  is frequent, and so we call it a *sequential pattern*. It is a *3-pattern* since it is a sequential pattern of length three.  $\square$

This model of sequential pattern mining is an abstraction of customer-shopping sequence analysis. Scalable methods for sequential pattern mining on such data are described in Section 5.4.2, which follows. Many other sequential pattern mining applications may not be covered by this model. For example, when analyzing Web clickstream sequences, gaps between clicks become important if one wants to predict what the next click might be. In DNA sequence analysis, *approximate* patterns become useful since DNA sequences may contain (symbol) insertions, deletions, and mutations. Such diverse requirements can be viewed as *constraint relaxation* or *enforcement*. In Section 5.4.3, we discuss how to extend the basic sequential mining model to *constrained* sequential pattern mining in order to handle these cases.

## 5.4.2 Scalable methods for mining sequential patterns

Sequential pattern mining is computationally challenging since such mining may generate and/or test a combinatorially explosive number of intermediate subsequences.

“How can we develop efficient and scalable methods for sequential pattern mining?” We may categorize the sequential pattern mining methods into two categories: (1) efficient methods for mining the *full set* of sequential patterns, and (2) efficient methods for mining only the *set of closed* sequential patterns, where a sequential pattern  $s$  is **closed** if there exists no sequential pattern  $s'$  where  $s'$  is a proper supersequence of  $s$ , and  $s'$  has the same (frequency) support as  $s$ .<sup>3</sup> Since all of the subsequences of a frequent sequence are also frequent, mining the set of closed sequential patterns may avoid the generation of unnecessary subsequences and thus lead to more compact results as well as more efficient methods than mining the full set. We will first examine methods for mining the full set and then study how they can be extended for mining the closed set. In addition, we discuss modifications for mining multilevel, multidimensional sequential patterns (that is, with multiple levels of granularity).

The major approaches for mining the full set of sequential patterns are similar to those introduced for frequent itemset mining in Chapter 5. Here, we discuss three such approaches for sequential pattern mining, represented by the algorithms GSP, SPADE, and PrefixSpan, respectively. GSP adopts a

<sup>3</sup> Closed frequent itemsets were introduced in Chapter 4. Here, the definition is applied to sequential patterns.

*candidate generate-and-test* approach using *horizontal data format* (where the data are represented as  $\langle \text{sequence\_ID} : \text{sequence\_of\_itemsets} \rangle$ , as usual, where each itemset is an event). SPADE adopts a candidate generate-and-test approach using *vertical data format* (where the data are represented as  $\langle \text{itemset} : (\text{sequence\_ID}, \text{event\_ID}) \rangle$ ). The vertical data format can be obtained by transforming from a horizontally formatted sequence database in just one scan. PrefixSpan is a *pattern growth* method, which does not require candidate generation.

All three approaches either directly or indirectly explore the **Apriori property**, stated as follows: *every nonempty subsequence of a sequential pattern is a sequential pattern*. (Recall that for a pattern to be called sequential, it must be frequent. That is, it must satisfy minimum support.) The Apriori property is antimonotonic (or downward-closed) in that, if a sequence cannot pass a test (e.g., regarding minimum support), all of its supersequences will also fail the test. Use of this property to prune the search space can help make the discovery of sequential patterns more efficient.

### **GSP: a sequential pattern mining algorithm based on candidate generate-and-test**

GSP (generalize sequential patterns) is a sequential pattern mining method that was developed by Srikant and Agrawal in 1996. It is an extension of their seminal algorithm for frequent itemset mining, known as Apriori (Section 5.2). GSP makes use of the downward-closure property of sequential patterns and adopts a multiple-pass, candidate generate-and-test approach. The algorithm is outlined as follows. In the first scan of the database, it finds all of the frequent items, that is, those with minimum support. Each such item yields a length-1 frequent sequence consisting of that item. Each subsequent pass starts with a *seed set* of sequential patterns—the set of sequential patterns found in the previous pass. This seed set is used to generate new potentially frequent patterns, called *candidate sequences*. Each candidate sequence contains one more item than the seed sequential pattern from which it was generated. Recall that the number of instances of items in a sequence is the *length* of the sequence. Therefore all of the candidate sequences in a given pass will have the same length. We refer to a sequence with length  $k$  as a  $k$ -sequence. Let  $C_k$  denote the set of candidate  $k$ -sequences. A pass over the database finds the support for each candidate  $k$ -sequence. The candidates in  $C_k$  with at least  $\text{min\_sup}$  form  $L_k$ , the set of all *frequent*  $k$ -sequences. This set then becomes the seed set for the next pass,  $k + 1$ . The algorithm terminates when no new sequential pattern is found in a pass, or no candidate sequence can be generated.

The method is illustrated in the following example.

**Example 5.13. GSP: candidate generate-and-test (using horizontal data format).** Suppose we are given the same sequence database,  $S$ , of Table 5.4 from Example 5.12, with  $\text{min\_sup} = 2$ . Note that the data are represented in horizontal data format. In the first scan ( $k = 1$ ), GSP collects the support for each item. The set of candidate 1-sequences is thus (shown here in the form of “*sequence : support*”):  $\langle a \rangle : 4$ ,  $\langle b \rangle : 4$ ,  $\langle c \rangle : 4$ ,  $\langle d \rangle : 3$ ,  $\langle e \rangle : 3$ ,  $\langle f \rangle : 3$ ,  $\langle g \rangle : 1$ .

The sequence  $\langle g \rangle$  has a support of only 1, and is the only sequence that does not satisfy minimum support. By filtering it out, we obtain the first seed set,  $L_1 = \{\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle, \langle f \rangle\}$ . Each member in the set represents a length-1 sequential pattern. Each subsequent pass starts with the seed set found in the previous pass and uses it to generate new candidate sequences, which are potentially frequent.

Using  $L_1$  as the seed set, this set of 6 length-1 sequential patterns generates a set of  $6 \times 6 + \frac{6 \times 5}{2} = 51$  candidate sequences of length 2,  $C_2 = \{\langle aa \rangle, \langle ab \rangle, \dots, \langle af \rangle, \langle ba \rangle, \langle bb \rangle, \dots, \langle ff \rangle, \langle (ab) \rangle, \langle (ac) \rangle, \dots, \langle (ef) \rangle\}$ . Note that  $\langle aa \rangle$  indicates that  $\langle a \rangle$  happens twice in sequel, and  $\langle ab \rangle$  indicates that  $\langle a \rangle$  happens followed by  $\langle b \rangle$ .



In general, the set of candidates is generated by a self-join of the sequential patterns found in the previous pass (see Section 5.2.1 for details). GSP applies the Apriori property to prune the set of candidates as follows. In the  $k$ th pass, a sequence is a candidate only if each of its length- $(k - 1)$  subsequences is a sequential pattern found at the  $(k - 1)$ th pass. A new scan of the database collects the support for each candidate sequence and finds a new set of sequential patterns,  $L_k$ . This set becomes the seed for the next pass. The algorithm terminates when no sequential pattern is found in a pass, or when there is no candidate sequence generated. Clearly, the number of scans is at least the maximum length of sequential patterns. GSP needs one more scan if the sequential patterns obtained in the last scan still generate new candidates.

Although GSP benefits from the Apriori pruning, it still generates a large number of candidates. In this example, 6 length-1 sequential patterns generate 51 length-2 candidates; 22 length-2 sequential patterns generate 64 length-3 candidates; and so on. Some candidates generated by GSP may not appear in the database at all. In this example, 13 out of 64 length-3 candidates do not appear in the database, resulting in wasted search effort.  $\square$

The example shows that although an Apriori-like sequential pattern mining method, such as GSP, reduces search space, it typically needs to scan the database multiple times. It will likely generate a huge set of candidate sequences, especially when mining long sequences. There is a need for more efficient mining method.

### ***SPADE: an Apriori-based vertical data format sequential pattern mining algorithm***

The Apriori-like sequential pattern mining approach (based on candidate generate-and-test) can also be explored by mapping a sequence database into vertical data format. In **vertical data format**, the database becomes a set of tuples of the form  $\langle \text{itemset} : (\text{sequence\_ID}, \text{event\_ID}) \rangle$ . That is, for a given itemset, we record the sequence identifier and corresponding event identifier for which the itemset occurs. The **event identifier** serves as a timestamp within a sequence. The *event\_ID* of the  $i$ th itemset (or event) in a sequence is  $i$ . Note that an itemset can occur in more than one sequence. The set of  $(\text{sequence\_ID}, \text{event\_ID})$  pairs for a given itemset forms the **ID\_list** of the itemset. The mapping from horizontal to vertical format requires one scan of the database. A major advantage of using this format is that we can determine the support of any  $k$ -sequence by simply joining the ID\_lists of any two of its  $(k - 1)$ -length subsequences. The length of the resulting ID\_list (i.e., unique *sequence\_ID* values) is equal to the support of the  $k$ -sequence, which tells us whether or not the sequence is frequent.

SPADE (sequential **p**attern **d**iscovery using equivalent classes) is an Apriori-based sequential pattern mining algorithm that uses vertical data format. As with GSP, SPADE requires one scan to find the frequent 1-sequences. To find candidate 2-sequences, we join all pairs of single items if they are frequent (therein, it applies the Apriori property), share the same sequence identifier, and their event identifiers follow a sequential ordering. That is, the first item in the pair must occur as an event before the second item, where both occur in the same sequence. Similarly, we can grow the length of itemsets from length 2 to length three, and so on. The procedure stops when no frequent sequences can be found or no such sequences can be formed by such joins. The following example helps illustrate the process.

**Example 5.14. SPADE: candidate generate-and-test using vertical data format.** Let  $\text{min\_sup} = 2$ . Our running example sequence database,  $S$ , of Table 5.4 is in horizontal data format. SPADE first scans  $S$  and transforms it into the vertical format, as shown in Fig. 5.8(a). Each itemset (or event) is associated with its ID\_list, which is the set of *SID* (*sequence\_ID*) and *EID* (*event\_ID*) pairs that contain the itemset.

<i>SID</i>	<i>EID</i>	<i>itemset</i>
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

(a) vertical format database

a		b		...
<i>SID</i>	<i>EID</i>	<i>SID</i>	<i>EID</i>	...
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	
2	4	4	5	
3	2			
4	3			

(b) ID\_lists for some 1-sequences

ab			ba			...
<i>SID</i>	<i>EID(a)</i>	<i>EID(b)</i>	<i>SID</i>	<i>EID(b)</i>	<i>EID(a)</i>	...
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

(c) ID\_lists for some 2-sequences

aba				...
<i>SID</i>	<i>EID(a)</i>	<i>EID(b)</i>	<i>EID(a)</i>	...
1	1	2	3	
2	1	3	4	

(d) ID\_lists for some 3-sequences

**FIGURE 5.8**

The SPADE mining process: (a) vertical format database; and (b) to (d) show fragments of the ID\_lists for 1-sequences, 2-sequences, and 3-sequences, respectively.

The ID\_list for individual items,  $a$ ,  $b$ , and so on, is shown in Fig. 5.8(b). For example, the ID\_list for item  $b$  consists of the following  $(SID, EID)$  pairs:  $\{(1, 2), (2, 3), (3, 2), (3, 5), (4, 5)\}$ , where the entry  $(1,2)$  means that  $b$  occurs in sequence 1, event 2, etc. Items  $a$  and  $b$  are frequent. They can be joined to form the length-2 sequence,  $\langle a, b \rangle$ . We find the support of this sequence as follows. We join the ID\_lists of  $a$  and  $b$  by joining on the same  $sequence\_ID$  wherever, according to the  $event\_IDs$ ,  $a$  occurs before  $b$ . That is, the join must preserve the temporal order of the events involved. The result of such a join for  $a$  and  $b$  is shown in the ID\_list for  $ab$  of Fig. 5.8(c). For example, the ID\_list for 2-sequence  $ab$  is a set of triples,  $(SID, EID(a), EID(b))$ , namely  $\{(1, 1, 2), (2, 1, 3), (3, 2, 5), (4, 3, 5)\}$ . The entry  $(2,1,3)$ , for example, shows that both  $a$  and  $b$  occur in sequence 2, and that  $a$  (event 1 of the sequence) occurs before  $b$  (event 3), as required. Furthermore, the frequent 2-sequences can be joined (while considering the Apriori pruning heuristic that the  $(k-1)$ -subsequences of a candidate  $k$ -sequence must be frequent) to form 3-sequences, as in Fig. 5.8(d), and so on. The process terminates when no frequent sequences can be found or no candidate sequences can be formed.  $\square$

The use of vertical data format, with the creation of ID\_lists, reduces scans of the sequence database. The ID\_lists carry the information necessary to find the support of candidates. As the length of a frequent sequence increases, the size of its ID\_list decreases, resulting in fast joins. However, the basic search methodology of SPADE and GSP is breadth-first search (e.g., exploring 1-sequences, then 2-sequences, and so on) and Apriori pruning. Despite the pruning, both algorithms have to generate large sets of candidates in breadth-first manner in order to grow longer sequences. Thus, most of the difficulties suffered in the GSP algorithm will reoccur in SPADE as well.

### **PrefixSpan: prefix-projected sequential pattern growth**

*Pattern growth* is a method of frequent-pattern mining that does not require candidate generation. The technique originated in the FP-growth algorithm for transaction databases, presented in Section 5.6. The general idea of this approach is as follows: it finds the frequent single items, then compresses this information into a *frequent-pattern tree*, or *FP-tree*. The FP-tree is used to generate a set of projected databases, each associated with one frequent item. Each of these databases is mined separately and recursively, avoiding candidate generation. Interestingly, the pattern-growth approach can be extended to mining sequential patterns, which leads to a new algorithm, PrefixSpan, illustrated below.

Without loss of generality, all the items within an event can be listed alphabetically. For example, instead of listing the items in an event as, say,  $\langle bac \rangle$ , we can list them as  $\langle abc \rangle$ . Given a sequence  $\alpha = \langle e_1 e_2 \dots e_n \rangle$  (where each  $e_i$  corresponds to a frequent event in a sequence database,  $S$ ), a sequence  $\beta = \langle e'_1 e'_2 \dots e'_m \rangle$  ( $m \leq n$ ) is called a **prefix** of  $\alpha$  if and only if (1)  $e'_i = e_i$  for  $(i \leq m - 1)$ ; (2)  $e'_m \subseteq e_m$ ; and (3) all the frequent items in  $(e_m - e'_m)$  are alphabetically after those in  $e'_m$ . Sequence  $\gamma = \langle e''_m e_{m+1} \dots e_n \rangle$  is called the **suffix** of  $\alpha$  with respect to prefix  $\beta$ , denoted as  $\gamma = \alpha/\beta$ , where  $e''_m = (e_m - e'_m)$ .<sup>4</sup> We also denote  $\alpha = \beta \cdot \gamma$ . Note if  $\beta$  is not a subsequence of  $\alpha$ , the suffix of  $\alpha$  with respect to  $\beta$  is empty.

We illustrate these concepts with the following example.

**Example 5.15. Prefix and suffix.** Let sequence  $s = \langle a(abc)(ac)d(cf) \rangle$ , which corresponds to sequence 1 of our running example sequence database.  $\langle a \rangle$ ,  $\langle aa \rangle$ ,  $\langle a(ab) \rangle$ , and  $\langle a(abc) \rangle$  are four prefixes

<sup>4</sup> If  $e''_m$  is not empty, the suffix is also denoted as  $\langle (\_ \text{ items in } e''_m) e_{m+1} \dots e_n \rangle$ .

of  $s$ .  $\langle(abc)(ac)d(cf)\rangle$  is the suffix of  $s$  with respect to the prefix  $\langle a \rangle$ ;  $\langle(\_bc)(ac)d(cf)\rangle$  is its suffix with respect to the prefix  $\langle aa \rangle$ ; and  $\langle(\_c)(ac)d(cf)\rangle$  is its suffix with respect to the prefix  $\langle a(ab) \rangle$ .  $\square$

Based on the concepts of prefix and suffix, the problem of mining sequential patterns can be decomposed into a set of subproblems as shown below.

1. Let  $\{\langle x_1 \rangle, \langle x_2 \rangle, \dots, \langle x_n \rangle\}$  be the complete set of length-1 sequential patterns in a sequence database,  $S$ . The complete set of sequential patterns in  $S$  can be partitioned into  $n$  disjoint subsets. The  $i$ th subset ( $1 \leq i \leq n$ ) is the set of sequential patterns with prefix  $\langle x_i \rangle$ .
2. Let  $\alpha$  be a length- $l$  sequential pattern and  $\{\beta_1, \beta_2, \dots, \beta_m\}$  be the set of all length- $(l+1)$  sequential patterns with prefix  $\alpha$ . The complete set of sequential patterns with prefix  $\alpha$ , except for  $\alpha$  itself, can be partitioned into  $m$  disjoint subsets. The  $j$ th subset ( $1 \leq j \leq m$ ) is the set of sequential patterns prefixed with  $\beta_j$ .

Based on this observation, the problem can be partitioned recursively. That is, each subset of sequential patterns can be further partitioned when necessary. This forms a *divide-and-conquer* framework. To mine the subsets of sequential patterns, we construct corresponding *projected databases* and mine each one recursively.

Let's use our running example to examine how to use the prefix-based projection approach for mining sequential patterns.

**Example 5.16. PrefixSpan: a pattern-growth approach.** Using the same sequence database,  $S$ , of Table 5.4 with  $min\_sup = 2$ , sequential patterns in  $S$  can be mined by a prefix-projection method in the following steps.

1. *Find length-1 sequential patterns.* Scan  $S$  once to find all of the frequent items in sequences. Each of these frequent items is a length-1 sequential pattern. They are  $\langle a \rangle : 4$ ,  $\langle b \rangle : 4$ ,  $\langle c \rangle : 4$ ,  $\langle d \rangle : 3$ ,  $\langle e \rangle : 3$ , and  $\langle f \rangle : 3$ , where the notation " $\langle pattern \rangle : count$ " represents the pattern and its associated support count.
2. *Partition the search space.* The complete set of sequential patterns can be partitioned into the following six subsets according to the six prefixes: (1) the ones with prefix  $\langle a \rangle$ , (2) the ones with prefix  $\langle b \rangle$ , ..., and (6) the ones with prefix  $\langle f \rangle$ .
3. *Find subsets of sequential patterns.* The subsets of sequential patterns mentioned in step 2 can be mined by constructing corresponding *projected databases* and mining each recursively. The projected databases, as well as the sequential patterns found in them, are listed in Table 5.5, while the mining process is explained as follows.
  - a. *Find sequential patterns with prefix  $\langle a \rangle$ .* Only the sequences containing  $\langle a \rangle$  should be collected. Moreover, in a sequence containing  $\langle a \rangle$ , only the subsequence prefixed with the first occurrence of  $\langle a \rangle$  should be considered. For example, in sequence  $\langle(ef)(ab)(df)cb\rangle$ , only the subsequence  $\langle(\_b)(df)cb\rangle$  should be considered for mining sequential patterns prefixed with  $\langle a \rangle$ . Notice that  $\langle\_b\rangle$  means that the last event in the prefix, which is  $a$ , together with  $b$ , form one event. The sequences in  $S$  containing  $\langle a \rangle$  are projected with respect to  $\langle a \rangle$  to form the  *$\langle a \rangle$ -projected database*, which consists of four suffix sequences:  $\langle(abc)(ac)d(cf)\rangle$ ,  $\langle(\_d)c(bc)(ae)\rangle$ ,  $\langle(\_b)(df)cb\rangle$  and  $\langle(\_f)cbc\rangle$ .  
By scanning the  $\langle a \rangle$ -projected database once, its locally frequent items are  $a : 2$ ,  $b : 4$ ,  $\_b : 2$ ,  $c : 4$ ,  $d : 2$ , and  $f : 2$ . Thus all the length-2 sequential patterns prefixed with  $\langle a \rangle$  are found, and they are  $\langle aa \rangle : 2$ ,  $\langle ab \rangle : 4$ ,  $\langle(\_a)b \rangle : 2$ ,  $\langle ac \rangle : 4$ ,  $\langle ad \rangle : 2$ , and  $\langle af \rangle : 2$ .

**Table 5.5 Projected databases and sequential patterns.**

Prefix	Projected Database	Sequential Patterns
$\langle a \rangle$	$\langle (abc)(ac)d(cf) \rangle, \langle (\_d)c(bc)(ae) \rangle, \langle (\_b)(df)cb \rangle, \langle (\_f)cbc \rangle$	$\langle a \rangle, \langle aa \rangle, \langle ab \rangle, \langle a(bc) \rangle, \langle a(bc)a \rangle, \langle aba \rangle, \langle abc \rangle, \langle (ab) \rangle, \langle (ab)c \rangle, \langle (ab)d \rangle, \langle (ab)f \rangle, \langle (ab)dc \rangle, \langle ac \rangle, \langle aca \rangle, \langle acb \rangle, \langle acc \rangle, \langle ad \rangle, \langle adc \rangle, \langle af \rangle$
$\langle b \rangle$	$\langle (\_c)(ac)d(cf) \rangle, \langle (\_c)(ae) \rangle, \langle (df)cb \rangle, \langle c \rangle$	$\langle b \rangle, \langle ba \rangle, \langle bc \rangle, \langle (bc) \rangle, \langle (bc)a \rangle, \langle bd \rangle, \langle bdc \rangle, \langle bf \rangle$
$\langle c \rangle$	$\langle (ac)d(cf) \rangle, \langle (bc)(ae) \rangle, \langle b \rangle, \langle bc \rangle$	$\langle c \rangle, \langle ca \rangle, \langle cb \rangle, \langle cc \rangle$
$\langle d \rangle$	$\langle (cf) \rangle, \langle c(bc)(ae) \rangle, \langle (\_f)cb \rangle$	$\langle d \rangle, \langle db \rangle, \langle dc \rangle, \langle dcb \rangle$
$\langle e \rangle$	$\langle (\_f)(ab)(df)cb \rangle, \langle (af)cbc \rangle$	$\langle e \rangle, \langle ea \rangle, \langle eab \rangle, \langle eac \rangle, \langle each \rangle, \langle eb \rangle, \langle ebc \rangle, \langle ec \rangle, \langle ecb \rangle, \langle ef \rangle, \langle efb \rangle, \langle efc \rangle, \langle efc b \rangle$
$\langle f \rangle$	$\langle (ab)(df)cb \rangle, \langle cbc \rangle$	$\langle f \rangle, \langle fb \rangle, \langle fbc \rangle, \langle fc \rangle, \langle fcb \rangle$

Recursively, all sequential patterns with prefix  $\langle a \rangle$  can be partitioned into six subsets: (1) those prefixed with  $\langle aa \rangle$ , (2) those with  $\langle ab \rangle$ , ..., and finally, (6) those with  $\langle af \rangle$ . These subsets can be mined by constructing respective projected databases and mining each recursively as follows.

- i. The  $\langle aa \rangle$ -projected database consists of two nonempty (suffix) subsequences prefixed with  $\langle aa \rangle$ :  $\{ \langle (\_bc)(ac)d(cf) \rangle, \{ \langle (\_e) \rangle \}$ . Since there is no hope of generating any frequent subsequence from this projected database, the processing of the  $\langle aa \rangle$ -projected database terminates.
  - ii. The  $\langle ab \rangle$ -projected database consists of three suffix sequences:  $\langle (\_c)(ac)d(cf) \rangle, \langle (\_c)a \rangle$ , and  $\langle c \rangle$ . Recursively mining the  $\langle ab \rangle$ -projected database returns four sequential patterns:  $\langle (\_c) \rangle, \langle (\_c)a \rangle, \langle a \rangle$ , and  $\langle c \rangle$  (i.e.,  $\langle a(bc) \rangle, \langle a(bc)a \rangle, \langle aba \rangle$ , and  $\langle abc \rangle$ .) They form the complete set of sequential patterns prefixed with  $\langle ab \rangle$ .
  - iii. The  $\langle (ab) \rangle$ -projected database contains only two sequences:  $\langle (\_c)(ac)d(cf) \rangle$  and  $\langle (df)cb \rangle$ , which leads to the finding of the following sequential patterns prefixed with  $\langle (ab) \rangle$ :  $\langle c \rangle, \langle d \rangle, \langle f \rangle$ , and  $\langle dc \rangle$ .
  - iv. The  $\langle ac \rangle$ -,  $\langle ad \rangle$ - and  $\langle af \rangle$ -projected databases can be constructed and recursively mined in a similar manner. The sequential patterns found are shown in Table 5.5.
- b. Find sequential patterns with prefix  $\langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle$  and  $\langle f \rangle$ , respectively. This can be done by constructing the  $\langle b \rangle$ -,  $\langle c \rangle$ -,  $\langle d \rangle$ -,  $\langle e \rangle$ -, and  $\langle f \rangle$ -projected databases and mining them. The projected databases and the sequential patterns found are also shown in Table 5.5.
4. The set of sequential patterns is the collection of patterns found in the above recursive mining process. □

The method described above generates no candidate sequences in the mining process. However, it may generate many projected databases, one for each frequent prefix-subsequence. Forming a large number of projected databases recursively may become the major cost of the method if such databases have to be generated physically. An important optimization technique is **pseudo-projection**, as shown in Fig. 5.9. For example, for a sequence  $\langle a(abc)(ac)d(cf) \rangle$ ,  $\langle a \rangle$ 's projection will generate a projected subsequence  $\langle (abc)(ac)d(cf) \rangle$  (i.e.,  $\langle a \rangle$ 's suffix), and a subsequent projection on  $\langle b \rangle$  generates an  $\langle ab \rangle$ 's projected sequence  $\langle (\_c)(ac)d(cf) \rangle$ . Such physical projection may take a lot of time and space to copy and store the projected subsequences, which contains a lot of redundancy. The pseudo-projection method registers the index (or identifier) of the corresponding sequence and the starting position of the

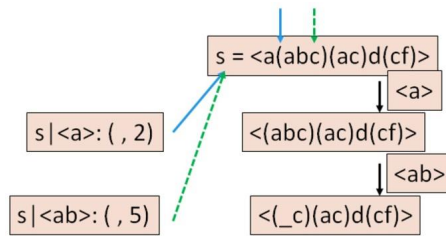


FIGURE 5.9

Pseudo projection vs. physical project in PrefixSpan.

projected suffix in the sequence instead of performing physical projection. That is, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point. For example, in the above two projections, instead of generating two physical projected suffixes, only two pointers are created (one pointing at position 2 shown by the solid arrow and the other at position 5 shown by the dashed arrow). This may save time to copy and paste suffixes and save spaces to store such suffixes.

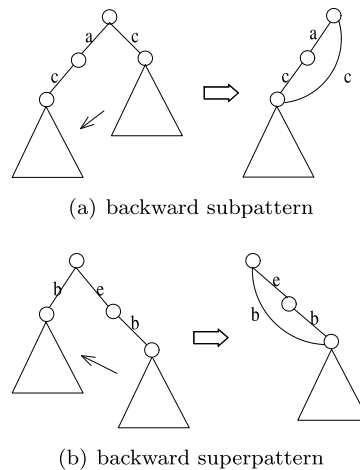
Pseudo-projection reduces the cost of projection substantially when such projection can be done in main memory. However, it may not be efficient if the pseudo-projection is used for disk-based accessing since random access to disk space is costly. The suggested approach is that if the original sequence database or the projected databases are too big to fit in memory, the physical projection should be applied, however, the execution should be swapped to pseudo-projection once the projected databases can fit in memory. This methodology is adopted in the PrefixSpan implementation.

A performance comparison of GSP, SPADE, and PrefixSpan shows that PrefixSpan has the best overall performance. SPADE, though weaker than PrefixSpan in most cases, outperforms GSP. Generating huge candidate sets may consume a tremendous amount of memory, thereby causing candidate generate-and-test algorithms to become rather slow. The comparison also found that when there is a large number of frequent subsequences, all three algorithms run slowly. This problem can be partially solved by closed sequential pattern mining.

### ***Mining closed sequential patterns***

Since mining the complete set of frequent subsequences can generate a huge number of sequential patterns, an interesting alternative is to mine frequent *closed subsequences* only, that is, those containing no supersequence with the same support. Mining closed sequential patterns can produce a significantly less number of sequences than the full set of sequential patterns. Note that the full set of frequent subsequences, together with their supports, can easily be derived from the closed subsequences. Thus closed subsequences have the same expressive power as the corresponding full set of subsequences. Because of their compactness, they may also be quicker to find.

CloSpan is an efficient closed sequential pattern mining method. Similar to mining closed frequent patterns, it can skip mining redundant closed sequential pattern if it finds the continuous mining will not generate any new results. For example, as shown in Fig. 5.10, if the projected database  $\Delta$  of prefix  $\langle ac \rangle$  is identical to the later projected database  $\Delta$  of prefix  $\langle c \rangle$  (which is called *backward subpattern* since  $\langle c \rangle \Delta$  arrives late and is a subpattern of  $\langle ac \rangle \Delta$ ), CloSpan will prune the later  $\Delta$  mining to avoid

**FIGURE 5.10**

The pruning of a backward subpattern or a backward superpattern.

redundancy. Similarly, CloSpan will search for *backward superpatterns* for pruning to avoid redundant mining. More concretely, it will stop growing a prefix-based projected databases  $S|_{\beta}$  if it is of the same size as that of the prefix-based projected database  $S|_{\alpha}$  and  $\alpha$  and  $\beta$  have substring/superstring relationships.

This is based on a property of sequence databases, called **equivalence of projected databases**, stated as follows: *Two projected sequence databases,  $S|_{\alpha} = S|_{\beta}$ ,<sup>5</sup>  $\alpha \sqsubseteq \beta$  (i.e.,  $\alpha$  is a subsequence of  $\beta$ ), are equivalent if and only if the total number of items in  $S|_{\alpha}$  is equal to the total number of items in  $S|_{\beta}$ .*

Let's examine one such example.

**Example 5.17. CloSpan: Pruning redundant projected database.** Given a small sequence database,  $S$ , shown in Fig. 5.11, with  $min\_sup = 2$ . The prefix project sequence database of the prefix  $\langle af \rangle$  is  $(\langle acg \rangle, \langle egb(ac) \rangle, \langle ea \rangle)$  with 12 symbols (including parentheses), and the projected sequence database of the prefix  $\langle f \rangle$  is of the same size. Clearly, the two projected databases should be identical and there is no need to mine the latter, the  $\langle f \rangle$ -projected sequence database. This is understandable since for any sequence  $s$ , if its projections on  $\langle af \rangle$  and  $\langle f \rangle$  respectively are not identical, the latter must contain more symbols than the former (e.g., it may contain only  $\langle f \rangle$  but not  $\langle a \dots f \rangle$  or has  $\langle f \rangle$  in front of  $\langle a \dots f \rangle$ ). However, now, the two sizes are equal. This implies that their projected databases must be identical. Such backward subpattern pruning and backward superpattern pruning can reduce the search space substantially.  $\square$

Empirical results show that CloSpan often derives a much smaller set of sequential patterns in a shorter time than PrefixSpan, which mines the complete set of sequential patterns.

<sup>5</sup> In  $S|_{\alpha}$ , a sequence database  $S$  is projected with respect to sequence (e.g., prefix)  $\alpha$ . The notation  $S|_{\beta}$  can be similarly defined.

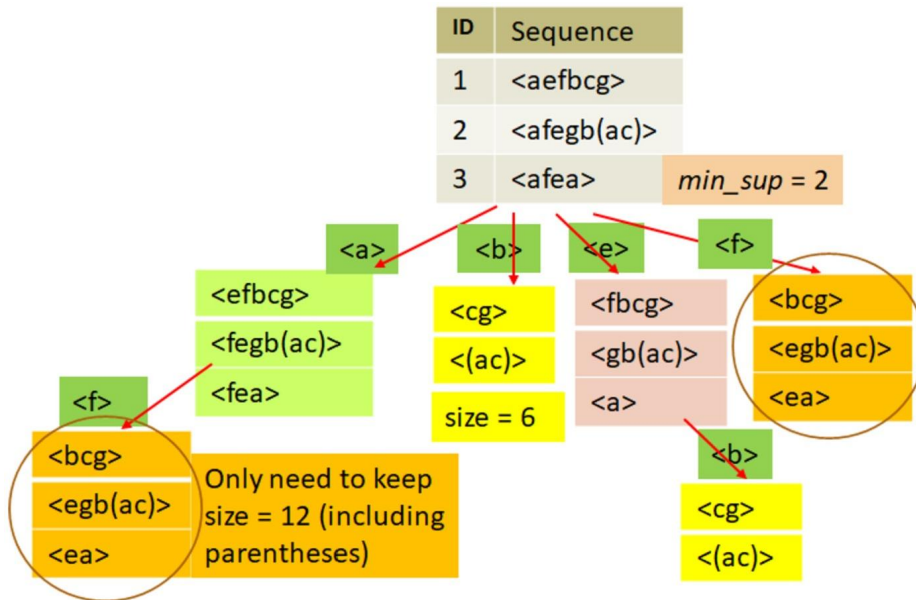


FIGURE 5.11

The pruning of a backward subpattern or a backward superpattern.

### Mining multidimensional, multilevel sequential patterns

Sequence identifiers (representing individual customers, for example) and sequence items (such as products bought) are often associated with additional pieces of information. Sequential pattern mining may take advantage of such additional information to discover interesting patterns in multidimensional, multilevel information space. Take customer shopping transactions, for instance. In a sequence database for such data, the additional information associated with sequence IDs could include customer residential area, group, and profession. Information associated with items could include item category, brand, model type, model number, place manufactured, and manufacture date. Mining *multidimensional, multilevel* sequential patterns is the discovery of interesting patterns in such a broad dimensional space, at different levels of detail.

**Example 5.18. Multidimensional, multilevel sequential patterns.** The discovery that “*Retired customers who purchase a smart home thermostat are likely to purchase a video doorbell within a month*” and that “*Young adults who purchase a laptop are likely to buy laser printer within 90 days*” are examples of multidimensional, multilevel sequential patterns. By grouping customers into “*retired customers*” and “*young adults*” according to the values in the age dimension, and by generalizing items to, say, “*smart thermostat*” rather than a specific model, the patterns mined here are associated with certain dimensions and are at a higher level of abstraction. □

“Can a typical sequential pattern algorithm such as *PrefixSpan* be extended to efficiently mine multidimensional, multilevel sequential patterns?” One suggested modification is to associate the mul-



tidimensional, multilevel information with the *sequence\_ID* and *item\_ID*, respectively, which the mining method can take into consideration when finding frequent subsequences. For example, (*Chicago, middle\_aged, business*) can be associated with *sequence\_ID\_1002* (for a given customer), whereas (*laserprinter, HP, LaserJetPro, G3Q47A, USA, 2020*) can be associated with *item\_ID\_543005* in the sequence. A sequential pattern mining algorithm will use such information in the mining process to find sequential patterns associated with multidimensional, multilevel information.

### 5.4.3 Constraint-based mining of sequential patterns

As shown in our study of frequent-pattern mining, mining that is performed without user-specified constraints may generate numerous patterns that are of no interest. Such unfocused mining can reduce both the efficiency and usability of frequent-pattern mining. Thus we promote **constraint-based mining**, which incorporates user-specified constraints to reduce the search space and derive only patterns that are of interest to the user.

Constraints can be expressed in many forms. They may specify desired relationships between attributes, attribute values, or aggregates within the resulting patterns mined. Regular expressions can also be used as constraints in the form of “pattern templates,” which specify the desired form of the patterns to be mined. The general concepts introduced for constraint-based frequent pattern mining apply to constraint-based sequential pattern mining as well. The key idea to note is that these kinds of constraints can be used *during* the mining process to confine the search space, thereby improving (1) the efficiency of the mining, and (2) the interestingness of the resulting patterns found. This idea is also referred to as “*pushing the constraints deep into the mining process.*”

We now examine some typical examples of constraints for sequential pattern mining.

First, constraints can be related to the **duration**,  $T$ , of a sequence. The duration can be user-specified, related to a particular time period, such as within the last 6 months. Sequential pattern mining can then be confined to the data within the specified duration,  $T$ . Constraints related to a specific duration, can be considered as *succinct* constraints. A constraint is **succinct** if we can enumerate all and only those sequences that are guaranteed to satisfy the constraint, even before support counting begins. In this case, we can push the data selection process deep into the mining process, and select sequences in the desired period before mining begins to reduce the search space.

Second, a user may confine the maximal or minimal length of the sequential patterns to be mined. The maximal or minimal length of sequential patterns can be treated as *antimonotonic* or *monotonic* constraints, respectively. For example, the constraint  $L \leq 10$  is *antimonotonic* since, if a sequential pattern violates this constraint, further mining following it will always violate the constraint. Similarly, data antimonotonicity and its search space pruning rules can be established correspondingly for sequential pattern mining as well.

Third, in sequential pattern mining, a constraint can be related to an **event folding window**,  $w$ . A set of events occurring within a specified period of time can be viewed as occurring together. If  $w$  is set to 0 (i.e., no event sequence folding), sequential patterns are found where each event occurs at a distinct time instant, such as “*a customer bought a laptop, then a digital camera, and then a laser printer*” will be considered as a length-3 sequence, even if all these happen within the same day. However, if  $w$  is set to be weekly based, then these transactions are considered as occurring within the same period, and such sequences are “folded” into a set in the analysis. On the extreme, if  $w$  is set to be as long as the whole duration,  $T$ , sequential pattern mining is degenerated into sequence-insensitive frequent pattern mining.

Fourth, a desired time **gap** between events in the discovered patterns may be specified as a constraint. For example,  $min\_gap \leq gap \leq max\_gap$  is to find patterns that are separated by at least  $min\_gap$  but at most  $max\_gap$ . A pattern like “If a person rents movie A, it is likely she will rent movie B not within 6 days but within 30 days” implies  $6 < gap \leq 30$  (days). It is straightforward to push gap constraints into the sequential pattern mining process. With minor modifications to the mining process, it can handle constraints with approximate gaps as well.

Finally, a user can specify constraints on the kinds of sequential patterns by providing “pattern templates” in the form of regular expressions. Here we discuss mining *serial episodes* and *parallel episodes* using *regular expressions*. A **serial episode** is a set of events that occurs in total order, whereas a **parallel episode** is a set of events whose occurrence ordering is trivial. Consider the following example.

**Example 5.19. Specifying serial episodes and parallel episodes with regular expressions.** Let the notation  $(E, t)$  represent *event type E at time t*. Consider the data  $(A, 1)$ ,  $(C, 2)$ , and  $(B, 5)$  with an event folding window width of  $w = 2$ , where the serial episode  $A \rightarrow B$  and the parallel episode  $A \& C$  both occur in the data. The user can specify constraints in the form of a regular expression, such as  $\{A|B\}C * \{D|E\}$ , which indicates that the user would like to find patterns where event  $A$  and  $B$  first occur (but they are parallel in that their relative ordering is unimportant), followed by one or a set of events  $C$ , followed by the events  $D$  and  $E$  (where  $D$  can occur either before or after  $E$ ). Other events can occur in between those specified in the regular expression.  $\square$

A regular expression constraint may be neither antimonotonic nor monotonic. In such cases, we cannot use it to prune the search space in the same ways as described above. However, by modifying the PrefixSpan-based pattern-growth approach, such constraints can be handled in an elegant manner. Let’s examine one such example.

**Example 5.20. Constraint-based sequential pattern mining with a regular expression constraint.** Suppose that our task is to mine sequential patterns, again using the sequence database,  $S$ , of Table 5.4. This time, however, we are particularly interested in patterns that match the regular expression constraint,  $C = \langle a * \{bb|(bc)d|dd\} \rangle$ , with minimum support.

This constraint cannot be pushed deep into the mining process. Nonetheless, it can easily be integrated with the pattern-growth mining process as follows. First, only the  $\langle a \rangle$ -projected database,  $S|_{\langle a \rangle}$ , needs to be mined since the regular expression constraint  $C$  starts with  $a$ . Retain only the sequences in  $S|_{\langle a \rangle}$  that contain items within the set  $\{b, c, d\}$ . Second, the remaining mining can proceed from the suffix. This is essentially the *Suffix-Span* algorithm, which is symmetric to PrefixSpan in that it grows suffixes from the end of the sequence forward. The growth should match the suffix as the constraint,  $\langle \{bb|(bc)d|dd\} \rangle$ . For the projected databases that match these suffixes, we can grow sequential patterns either in prefix- or suffix-expansion manner to find all of the remaining sequential patterns.  $\square$

Thus we have seen several ways in which constraints can be used to improve the efficiency and usability of sequential pattern mining.

## 5.5 Mining subgraph patterns

Graphs become increasingly important in modeling complicated structures, such as circuits, images, workflows, XML documents, webpages, chemical compounds, protein structures, biological networks,

social networks, information networks, knowledge graphs, and the Web. Many graph search algorithms have been developed in chemical informatics, computer vision, video indexing, Web search, and text retrieval. With the increasing demand on the analysis of large amounts of structured data, graph mining has become an active and important theme in data mining.

Among the various kinds of graph patterns, *frequent substructures* or *subgraphs* are the very basic patterns that can be discovered in a collection of graphs. They are useful for characterizing graph sets, discriminating different groups of graphs, classifying and clustering graphs, building graph indices, and facilitating similarity search in graph databases. Recent studies have developed several graph mining methods and applied them to the discovery of interesting patterns in various applications. For example, there have been reports on the discovery of active chemical structures in HIV-screening data sets by contrasting the support of frequent graphs between different classes. There have been studies on the use of frequent structures as features to classify chemical compounds, on the frequent graph mining technique to study protein structural families, on the detection of considerably large frequent subpathways in metabolic networks, and on the use of frequent graph patterns for graph indexing and similarity search in graph databases. Although graph mining may include mining frequent subgraph patterns, graph classification, clustering, and other analysis tasks, in this section we focus on mining frequent subgraphs. We look at various methods, their extensions, and applications.

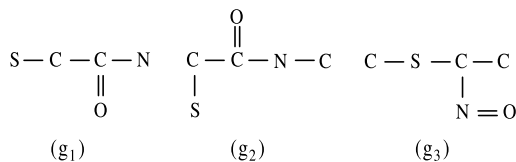
### 5.5.1 Methods for mining frequent subgraphs

Before presenting graph mining methods, it is necessary to first introduce some preliminary concepts relating to frequent graph mining.

We denote the **vertex set** of a graph  $g$  by  $V(g)$  and the **edge set** by  $E(g)$ . A label function,  $L$ , maps a vertex or an edge to a label. A graph  $g$  is a **subgraph** of another graph  $g'$  if there exists a subgraph isomorphism from  $g$  to  $g'$ . Given a labeled graph data set,  $D = \{G_1, G_2, \dots, G_n\}$ , we define *support*( $g$ ) (or *frequency*( $g$ )) as the percentage (or number) of graphs in a graph database (i.e., a collection of graphs)  $D$  where  $g$  is a subgraph. A **frequent graph** is a graph whose support is no less than a minimum support threshold,  $min\_sup$ .

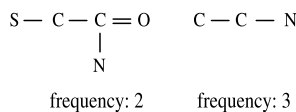
**Example 5.21. Frequent subgraph.** Fig. 5.12 shows a sample set of chemical structures. Fig. 5.13 depicts two of the frequent subgraphs in this data set, given a minimum support of 66.6%.  $\square$

“How can we discover frequent substructures?” The discovery of frequent substructures usually consists of two steps. In the first step, we generate frequent substructure candidates. The frequency of each candidate is checked in the second step. Most studies on frequent substructure discovery focus on



**FIGURE 5.12**

A sample graph data set.

**FIGURE 5.13**

Frequent graphs.

the optimization of the first step. This is because the second step involves a subgraph isomorphism test whose computational complexity is excessively high (that is, NP-complete).

In this section, we look at various methods for frequent substructure mining. In general, there are two basic approaches to this problem: an Apriori-based approach and a pattern-grown approach.

### ***Apriori-based approach***

Apriori-based frequent substructure mining algorithms share similar characteristics with Apriori-based frequent itemset mining algorithms (Chapter 4). The search for frequent graphs starts with graphs of small “size,” and proceeds in a bottom-up manner by generating candidates having an extra vertex, edge, or path. The definition of graph size depends on the algorithm used.

The general framework of Apriori-based methods for frequent substructure mining is outlined in Fig. 5.14. We refer to this algorithm as AprioriGraph.  $S_k$  is the frequent substructure set of size  $k$ . We will clarify the definition of graph size when we describe specific Apriori-based methods further below. AprioriGraph adopts a *level-wise* mining methodology. At each iteration, the size of newly discovered frequent substructures is increased by one. These new substructures are first generated by joining two similar but slightly different frequent subgraphs that were discovered in the previous call to AprioriGraph. This candidate generation procedure is outlined on line 4. The frequency of the newly formed structures is then checked. Those found to be frequent are used to generate larger candidates in the next round.

#### **Algorithm: AprioriGraph( $D$ , minsup, $S_k$ )**

Input: a graph data set  $D$ , and *min\_support*.

Output: The frequent substructure set  $S_k$ .

```

1:  $S_{k+1} \leftarrow \emptyset$ ;
2: for each frequent  $g_i \in S_k$  do
3:   for each frequent  $g_j \in S_k$  do
4:     for each size  $(k + 1)$  graph  $g$  formed by the merge of  $g_i$  and  $g_j$  do
5:       if  $g$  is frequent in  $D$  and  $g \notin S_{k+1}$  then
6:         insert  $g$  to  $S_{k+1}$ ;
7: if  $S_{k+1} \neq \emptyset$  then
8:   call AprioriGraph( $D$ , minsup,  $S_{k+1}$ );
9: return;

```

**FIGURE 5.14**

AprioriGraph.

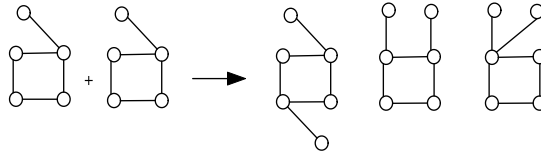


FIGURE 5.15

FSG: Two substructure patterns and their potential candidates.

The main design complexity of Apriori-based substructure mining algorithms is the candidate generation step. The candidate generation in frequent itemset mining is straightforward. For example, suppose we have two frequent itemsets of size-3:  $(abc)$  and  $(bcd)$ . The frequent itemset candidate of size-4 generated from them is simply  $(abcd)$ , derived from a join. However, the candidate generation problem in frequent substructure mining is harder than that in frequent itemset mining, since there are many ways to join two substructures, as shown below.

Apriori-based algorithms for frequent substructure mining include AGM, FSG, and a path-join method. AGM shares similar characteristics with Apriori-based itemset mining. FSG and the path-join method explore edges and connections in an Apriori-based fashion. Since edge is a bigger unit than vertex and it enforces more constraints than single vertex, the *edge-based candidate generation method* FSG leads to improved efficiency over the *vertex-based candidate generation method*, AGM. We examine the FSG method here.

The FSG algorithm adopts an *edge-based candidate generation* strategy that increases the substructure size by one edge in each call of AprioriGraph. Two size- $k$  patterns are merged if and only if they share the same subgraph having  $k - 1$  edges, which is called the **core**. Here, *graph size* is taken to be the number of edges in the graph. The newly formed candidate includes the core and the additional two edges from the size- $k$  patterns. Fig. 5.15 shows potential candidates formed by two structure patterns. Each candidate has one more edge than these two patterns, but this additional edge can be associated with different vertices. This example illustrates the complexity of joining two structures to form a large pattern candidate.

In a third Apriori-based approach, an *edge-disjoint path method* was proposed, where graphs are classified by the number of disjoint paths they have, and two paths are edge-disjoint if they do not share any common edge. A substructure pattern with  $k + 1$  disjoint paths is generated by joining substructures with  $k$  disjoint paths.

Apriori-based algorithms have considerable overhead when joining two size- $k$  frequent substructures to generate size- $(k + 1)$  graph candidates. The overhead occurs when (1) joining two size- $k$  frequent graphs (or other structures like paths) to generate size- $(k + 1)$  graph candidates, and (2) checking the frequency of these candidates separately. These two operations constitute the performance bottlenecks of the Apriori-like algorithms. In order to avoid such overhead, non-Apriori-based algorithms have been developed, most of which adopt the pattern-growth methodology. This methodology tries to extend patterns directly from a single pattern. In the following, we introduce the pattern-growth approach for frequent subgraph mining.

### Pattern-growth approach

The Apriori-based approach has to use the breadth-first search (BFS) strategy because of its level-wise candidate generation. In order to determine whether a size- $(k + 1)$  graph is frequent, it must check all of its corresponding size- $k$  subgraphs to obtain an upper bound of its frequency. Thus before mining any size- $(k + 1)$  subgraph, the Apriori-like approach usually has to complete the mining of size- $k$  subgraphs. Therefore, BFS is necessary in the Apriori-like approach. In contrast, the *pattern-growth approach* is more flexible regarding its search method. It can use breadth-first search and depth-first search (DFS), the latter of which consumes less memory.

A graph  $g$  can be *extended* by adding a new edge  $e$ . The newly formed graph is denoted by  $g \diamond_x e$ . Edge  $e$  may or may not introduce a new vertex to  $g$ . If  $e$  introduces a new vertex, we denote the new graph by  $g \diamond_{xf} e$ , otherwise,  $g \diamond_{xb} e$ , where  $f$  or  $b$  indicates that the extension is in a *forward* or *backward* direction.

Fig. 5.16 illustrates a general framework for pattern growth-based frequent substructure mining. We refer to the algorithm as PatternGrowthGraph. For each discovered graph  $g$ , it performs extensions recursively until all the frequent graphs with  $g$  embedded are discovered. The recursion stops once no frequent graph can be generated.

PatternGrowthGraph is simple, but not efficient. The bottleneck is at the inefficiency of extending a graph. The same graph can be discovered many times. For example, there may exist  $n$  different  $(n - 1)$ -edge graphs that can be extended to the same  $n$ -edge graph. The repeated discovery of the same graph is computationally inefficient. We call a graph that is discovered at the second time a **duplicate graph**. Although line 1 of PatternGrowthGraph gets rid of duplicate graphs, the generation and detection of duplicate graphs may increase the workload. In order to reduce the generation of duplicate graphs, each frequent graph should be extended as conservatively as possible. This principle leads to the design of several new algorithms. A typical example is the gSpan algorithm as described below.

The gSpan algorithm is designed to reduce the generation of duplicate graphs. It does not need to search previously discovered frequent graphs for duplicate detection. It does not extend any duplicate graph, yet still guarantees the discovery of the complete set of frequent graphs.

Let's see how the gSpan algorithm works. To traverse graphs, it adopts depth-first search. Initially, a starting vertex is randomly chosen and the vertices in a graph are marked so that we can tell which vertices have been visited. The visited vertex set is expanded repeatedly until a full depth-first search (DFS) tree is built. One graph may have various DFS trees depending on how the depth-first search is

**Algorithm:** PatternGrowthGraph( $g, D, \text{minsup}, S$ )

Input: A frequent graph  $g$ , a graph data set  $D$ , and the support threshold  $\text{minsup}$ .

Output: The frequent graph set  $S$ .

```

1: if  $g \in S$  then return;
2: else insert  $g$  to  $S$ ;
3: scan  $D$  once, find all the edges  $e$  such that  $g$  can be extended to  $g \diamond_x e$ ;
4: for each frequent  $g \diamond_x e$  do
5:   Call PatternGrowthGraph( $g \diamond_x e, D, \text{minsup}, S$ );
6: return;
```

**FIGURE 5.16**

PatternGrowthGraph.

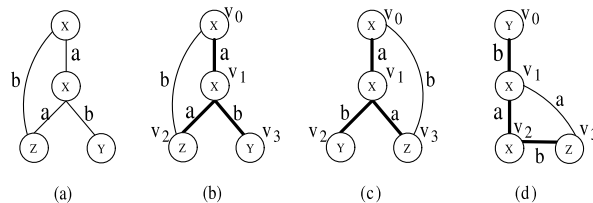


FIGURE 5.17

DFS subscripting.

performed, that is, the vertex visiting order. The darkened edges in Figs. 5.17(b) to 5.17(d) show three DFS trees for the same graph of Fig. 5.17(a). The vertex labels are  $x$ ,  $y$ , and  $z$ ; the edge labels are  $a$  and  $b$ . Alphabetic order is taken as the default order in the labels. When building a DFS tree, the visiting sequence of vertices forms a linear order. We use subscripts to record this order, where  $i < j$  means  $v_i$  is visited before  $v_j$  when the depth-first search is performed. A graph  $G$  subscripted with a DFS tree  $T$  is written as  $G_T$ .  $T$  is called a **DFS subscripting** of  $G$ . Given a DFS tree  $T$ , we call the starting vertex in  $T$ ,  $v_0$ , the *root*, and the last visited vertex,  $v_n$ , the *right-most vertex*. The straight path from  $v_0$  to  $v_n$  is called the *right-most path*. In Figs. 5.17(b) to 5.17(d), three different subscriptings are generated based on the corresponding DFS trees. The right-most path is  $(v_0, v_1, v_3)$  in Figs. 5.17(b) and 5.17(c), and  $(v_0, v_1, v_2, v_3)$  in Fig. 5.17(d).

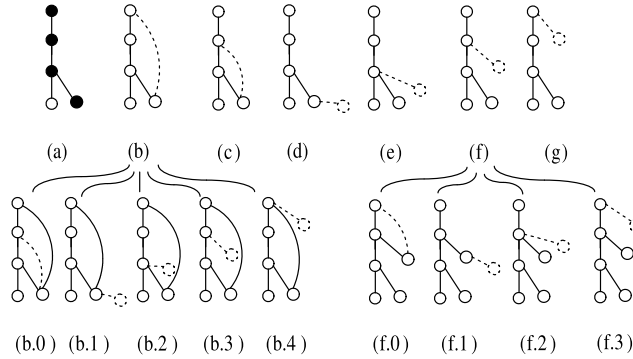
PatternGrowth extends a frequent graph in every possible position, which may generate a large number of duplicate graphs. The gSpan algorithm introduces a more sophisticated extension method. The new method restricts the extension as follows: Given a graph  $G$  and a DFS tree  $T$  in  $G$ , a new edge  $e$  can be added between the right-most vertex and other vertices on the right-most path (*backward extension*); or it can introduce a new vertex and connect to vertices on the right-most path (*forward extension*). Since both kinds of extensions take place on the right-most path, we call them *right-most extension*, denoted by  $G \diamond_r e$  (for brevity,  $T$  is omitted here).

**Example 5.22. Backward extension and forward extension.** If we want to extend the graph in Fig. 5.17(b), the backward extension candidates can be  $(v_3, v_0)$ . The forward extension candidates can be edges extending from  $v_3$ ,  $v_1$ , or  $v_0$  with a new vertex introduced.  $\square$

Figs. 5.18(b) to 5.18(g) show all the potential right-most extensions of Fig. 5.18(a). The darkened vertices show the rightmost path. Among these, Figs. 5.18(b) to 5.18(d) grow from the rightmost vertex while Figs. 5.18(e) to 5.18(g) grow from other vertices on the rightmost path. Figs. 5.18(b.0) to 5.18(b.4) are children of Fig. 5.18(b), and Figs. 5.18(f.0) to 5.18(f.3) are children of Fig. 5.18(f). In summary, backward extension only takes place on the rightmost vertex while forward extension introduces a new edge from vertices on the rightmost path.

Since many DFS trees/subscriptings may exist for the same graph, we choose one of them as the *base subscripting* and only conduct right-most extension on that DFS tree/subscripting. Otherwise, right-most extension cannot reduce the generation of duplicate graphs because we would have to extend the same graph for every DFS subscripting.

We transform each subscripted graph to an edge sequence, called a **DFS code**, so that we can build an order among these sequences. The goal is to select the subscripting that generates the minimum



**FIGURE 5.18**  
Right-most extension.

**Table 5.6 DFS code for Fig. 5.17(b), 5.17(c), and 5.17(d).**

edge	$\gamma_0$	$\gamma_1$	$\gamma_2$
$e_0$	(0, 1, X, a, X)	(0, 1, X, a, X)	(0, 1, Y, b, X)
$e_1$	(1, 2, X, a, Z)	(1, 2, X, b, Y)	(1, 2, X, a, X)
$e_2$	(2, 0, Z, b, X)	(1, 3, X, a, Z)	(2, 3, X, b, Z)
$e_3$	(1, 3, X, b, Y)	(3, 0, Z, b, X)	(3, 1, Z, a, X)

sequence as its base subscripting. There are two kinds of orders in this transformation process: (1) *edge order*, which maps edges in a subscripted graph into a sequence; and (2) *sequence order*, which builds an order among edge sequences, that is, graphs.

First, we introduce edge order. Intuitively, DFS tree defines the discovery order of forward edges. For the graph shown in Fig. 5.17(b), the forward edges are visited in the order of (0, 1), (1, 2), (1, 3). Now we put backward edges into the order as follows. Given a vertex  $v$ , all of its backward edges should appear just before its forward edges. If  $v$  does not have any forward edge, we put its backward edges after the forward edge where  $v$  is the second vertex. For vertex  $v_2$  in Fig. 5.17(b), its backward edge (2, 0) should appear after (1, 2) since  $v_2$  does not have any forward edge. Among the backward edges from the same vertex, we can enforce an order. Assume that a vertex  $v_i$  has two backward edges,  $(i, j_1)$  and  $(i, j_2)$ . If  $j_1 < j_2$ , then edge  $(i, j_1)$  will appear before edge  $(i, j_2)$ . So far, we have completed the ordering of the edges in a graph. Based on this order, a graph can be transformed into an edge sequence. A complete sequence for Fig. 5.17(b) is (0, 1), (1, 2), (2, 0), (1, 3).

Based on this ordering, three different DFS codes,  $\gamma_0$ ,  $\gamma_1$ , and  $\gamma_2$ , generated by DFS subscriptings in Figs. 5.17(b), 5.17(c), and 5.17(d), respectively, are shown in Table 5.6. An edge is represented by a 5-tuple,  $(i, j, l_i, l_{(i,j)}, l_j)$ ,  $l_i$  and  $l_j$  are the labels of  $v_i$  and  $v_j$ , respectively, and  $l_{(i,j)}$  is the label of the edge connecting them.

Through DFS coding, a one-to-one mapping is built between a subscripted graph and a DFS code (a one-to-many mapping between a graph and DFS codes). When the context is clear, we treat a sub-



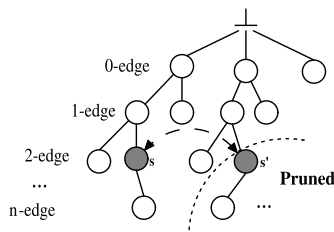


FIGURE 5.19

Lexicographic search tree.

scripted graph and its DFS code as the same. All the notations on subscripted graphs can also be applied to DFS codes. The graph represented by a DFS code  $\alpha$  is written  $G_\alpha$ .

Second, we define an order among edge sequences. Since one graph may have several DFS codes, we want to build an order among these codes and select one code to represent the graph. Since we are dealing with labeled graphs, the label information should be considered as one of the ordering factors. The labels of vertices and edges are used to break the tie when two edges have the exactly same subscript, but different labels. Let the edge order relation  $\prec_T$  take the first priority, the vertex label  $l_i$  take the second priority, the edge label  $l_{(i,j)}$  take the third, and the vertex label  $l_j$  take the fourth to determine the order of two edges. For example, the first edge of the three DFS codes in Table 5.6 is  $(0, 1, X, a, X)$ ,  $(0, 1, X, a, X)$ , and  $(0, 1, Y, b, X)$ , respectively. All of them share the same subscript  $(0, 1)$ . Therefore relation  $\prec_T$  cannot tell the difference among them. However, using label information, following the order of first vertex label, edge label, and second vertex label, we have  $(0, 1, X, a, X) < (0, 1, Y, b, X)$ . The ordering based on the above rules is called *DFS Lexicographic Order*. According to this ordering, we have  $\gamma_0 < \gamma_1 < \gamma_2$  for the DFS codes listed in Table 5.6.

Based on the DFS lexicographic ordering, the *minimum DFS code* of a given graph  $G$ , written as  $\text{dfs}(G)$ , is the minimal one among all the DFS codes. For example, code  $\gamma_0$  in Table 5.6 is the minimum DFS code of the graph in Fig. 5.17(a). The subscripting that generates the minimum DFS code is called the *base subscripting*.

We have the following important relationship between the minimum DFS code and the isomorphism of the two graphs: *Given two graphs  $G$  and  $G'$ ,  $G$  is isomorphic to  $G'$  if and only if  $\text{dfs}(G) = \text{dfs}(G')$ .* Based on this property, what we need to do for mining frequent subgraphs is to perform only the right-most extensions on the minimum DFS codes since such an extension will guarantee the completeness of mining results.

Fig. 5.19 shows how to arrange all DFS codes in a search tree through right-most extensions. The root is an empty code. Each node is a DFS code encoding a graph. Each edge represents a right-most extension from a  $(k - 1)$ -length DFS code to a  $k$ -length DFS code. The tree itself is ordered: left siblings are smaller than right siblings in the sense of DFS lexicographic order. Since any graph has at least one DFS code, the search tree can enumerate all possible subgraphs in a graph data set. However, one graph may have several DFS codes, minimum and nonminimum. The search of nonminimum DFS codes does not produce a useful result. “*Is it necessary to perform right-most extension on nonminimum DFS codes?*” The answer is “*no.*” If codes  $s$  and  $s'$  in Fig. 5.19 encode the same graph, the search space under  $s'$  can be safely pruned.

**Algorithm:** gSpan( $s, D, \text{minsup}, S$ )

Input: A DFS code  $s$ , a graph data set  $D$ , and  $\text{min\_support}$ .

Output: The frequent graph set  $S$ .

```

1: if  $s \neq dfs(s)$ , then
2:   return;
3: insert  $s$  into  $S$ ;
4: set  $C$  to  $\emptyset$ ;
5: scan  $D$  once, find all the edges  $e$  such that  $s$  can be right-most extended to  $s \diamond_r e$ ;
   insert  $s \diamond_r e$  into  $C$  and count its frequency;
6: sort  $C$  in DFS lexicographic order;
7: for each frequent  $s \diamond_r e$  in  $C$  do
8:   Call gSpan( $s \diamond_r e, D, \text{minsup}, S$ );
9: return;

```

**FIGURE 5.20**

gSpan: A pattern-growth algorithm for frequent substructure mining.

The details of gSpan are depicted in Fig. 5.20. gSpan is called recursively to extend graph patterns so that their frequent descendants are found until their support is lower than minsup or its code is not minimum any more. The difference between gSpan and PatternGrowth is at the right-most extension and extension termination of nonminimum DFS codes (lines 1–2). We replace the existence judgment in lines 1–2 of PatternGrowth with the inequation  $s \neq dfs(s)$ . Actually,  $s \neq dfs(s)$  is more efficient to calculate. Line 5 requires exhaustive enumeration of  $s$  in  $D$  in order to count the frequency of all the possible right-most extensions of  $s$ .

The algorithm of Fig. 5.20 implements a depth-first search version of gSpan. Actually, breadth-first search works too: for each newly discovered frequent subgraph in line 8, instead of directly calling gSpan, we insert it into a global first-in-first-out queue  $Q$ , which records all subgraphs that have not been extended. We then “gSpan” each subgraph in  $Q$  one by one. The performance of a breadth-first search version of gSpan is very close to that of the depth-first search although the latter usually consumes less memory.

### 5.5.2 Mining variant and constrained substructure patterns

The frequent subgraph mining discussed in the previous section handles only one special kind of graph: *labeled, undirected, connected simple graphs without any specific constraints*. That is, we assume that the database to be mined contains a set of graphs each consisting of a set of labeled vertices and labeled but undirected edges, with no other constraints. However, many applications or users may need to enforce various kinds of *constraints* on the patterns to be mined or seek *variant substructure patterns*. For example, we may like to mine patterns, each of which contains certain specific vertices/edges, or where the total number of vertices/edges is within a specified range. Or what if we seek patterns where the average density of the graph patterns is above a threshold? Although it is possible to develop customized algorithms for each such case, there are too many variant cases to consider. Instead, a general framework is needed—one that can organize variants and constraints and help develop efficient mining methods systematically. In this section, we study several variants and constrained substructure patterns and look at how they can be mined.

### ***Mining closed frequent substructures***

The first important variation of a frequent substructure is the **closed frequent substructure**. Take mining frequent subgraph as an example. Similar to mining frequent itemsets and mining sequential patterns, mining graph patterns may generate an explosive number of patterns. According to the Apriori property, all the subgraphs of a frequent graph are frequent. Thus a large graph pattern may generate an exponential number of frequent subgraphs. For example, among 423 confirmed active chemical compounds in an AIDS antiviral screen data set, there are nearly 1,000,000 frequent graph patterns whose support is at least 5%. This renders the further analysis on frequent graphs nearly impossible.

One way to alleviate this problem is to mine only frequent closed graphs, where a frequent graph  $G$  is **closed** if and only if there does not exist a proper supergraph  $G'$  that has the same support as  $G$ . Alternatively, we can mine maximal subgraph patterns where a frequent pattern  $G$  is **maximal** if and only if there does not exist a frequent superpattern of  $G$ . A set of closed subgraph patterns has the same expressive power as the full set of subgraph patterns under the same minimum support threshold because the latter can be generated by the derived set of closed graph patterns. On the other hand, the maximal pattern set is a subset of the closed pattern set. It is usually more compact than the closed pattern set. However, we cannot use it to reconstruct the entire set of frequent patterns—the support information of a pattern is lost if it is a proper subpattern of a maximal pattern, yet carries a different support.

**Example 5.23. Maximal frequent graph.** The two graphs in Fig. 5.13 are closed frequent graphs but only the first graph is a maximal frequent graph. The second graph is not maximal because it has a frequent supergraph. □

Mining closed graphs leads to a complete but more compact representation. For example, for the AIDS antiviral data set mentioned above, among the one million frequent graphs, only about 2000 are closed frequent graphs. If further analysis, such as classification or clustering, is performed on closed frequent graphs instead of frequent graphs, it will achieve similar accuracy with less redundancy and higher efficiency.

An efficient method, called CloseGraph, was developed for mining closed frequent graphs by extension of the gSpan algorithm. The key for efficient mining of closed frequent subgraphs is to figure out at what condition that the further growth of a frequent subgraph  $g$  should be pruned when its  $e$ -expanded subgraph  $g'$  has the same support as  $g$ . Experimental study has shown that CloseGraph often generates far fewer graph patterns and runs more efficiently than gSpan, which mines the full subgraph pattern set.

### ***Extension of pattern-growth approach: mining alternative substructure patterns***

A typical pattern-growth graph mining algorithm, such as gSpan or CloseGraph, mines *labeled, connected, undirected* frequent or closed subgraph patterns. Such a graph mining framework can be extended easily for mining *alternative substructure patterns*. Here we discuss a few such alternatives.

First, the method can be extended for **mining unlabeled or partially labeled graphs**. Each vertex and each edge in our previously discussed graphs contain labels. Alternatively, if none of the vertices and edges in a graph are labeled, the graph is **unlabeled**. A graph is **partially labeled** if only some of the edges and/or vertices are labeled. To handle such cases, we can build a label set that contains the original label set and a new empty label,  $\phi$ . Label  $\phi$  is assigned to vertices and edges that do not have labels. Notice that label  $\phi$  may match with any label or with  $\phi$  only, depending on the application

semantics. With this transformation, gSpan (and CloseGraph) can directly mine unlabeled or partially labeled graphs.

Second, we examine whether gSpan can be extended to **mining nonsimple graphs**. A **nonsimple graph** may have a *self-loop* (i.e., an edge joins a vertex to itself) and *multiple edges* (i.e., several edges connecting two of the same vertices). In gSpan, we always first grow backward edges and then forward edges. In order to accommodate self-loops, the growing order should be changed to *backward edges, self-loops, and forward edges*. If we allow sharing of the same vertices in two neighboring edges in a DFS code, the definition of DFS lexicographic order can handle multiple edges smoothly. Thus gSpan can mine nonsimple graphs efficiently too.

Third, we see how gSpan can be extended to handle **mining directed graphs**. In a directed graph, each edge of the graph has a defined direction. If we use a 5-tuple,  $(i, j, l_i, l_{(i,j)}, l_j)$ , to represent an undirected edge, then for directed edges, a new state is introduced to form a 6-tuple,  $(i, j, d, l_i, l_{(i,j)}, l_j)$ , where  $d$  represents the direction of an edge. Let  $d = +1$  be the direction from  $i$  ( $v_i$ ) to  $j$  ( $v_j$ ), whereas  $d = -1$  be that from  $j$  ( $v_j$ ) to  $i$  ( $v_i$ ). Notice that the sign of  $d$  is not related with the forwardness or backwardness of an edge. When extending a graph with one more edge, this edge may have two choices of  $d$ , which only introduces a new state in the growing procedure and need not change the framework of gSpan.

Fourth, the method can also be extended to **mining disconnected graphs**. There are two cases to be considered: (1) the graphs in the data set may be disconnected, and (2) the graph patterns may be disconnected. For the first case, we can transform the original data set by adding a virtual vertex to connect the disconnected graphs in each graph. We then apply gSpan on the new graph data set. For the second case, we redefine the DFS code. A disconnected graph pattern can be viewed as a set of connected graphs,  $r = \{g_0, g_1, \dots, g_m\}$ , where  $g_i$  is a connected graph,  $0 \leq i \leq m$ . Since each graph can be mapped to a minimum DFS code, a disconnected graph  $r$  can be translated into a code,  $\gamma = (s_0, s_1, \dots, s_m)$ , where  $s_i$  is the minimum DFS code of  $g_i$ . The order of  $g_i$  in  $r$  is irrelevant. Thus, we enforce an order in  $\{s_i\}$  such that  $s_0 \leq s_1 \leq \dots \leq s_m$ .  $\gamma$  can be extended by either adding one-edge  $s_{m+1}$  ( $s_m \leq s_{m+1}$ ) or by extending  $s_m, \dots$ , and  $s_0$ . When checking the frequency of  $\gamma$  in the graph data set, make sure that  $g_0, g_1, \dots$ , and  $g_m$  are disconnected with each other.

Finally, if we view a tree as a degenerated graph, it is straightforward to extend the method to **mining frequent subtrees**. In comparison with a general graph, a tree can be considered as a degenerated direct graph that does not contain any edges that can go back to its parent or ancestor nodes. Thus if we consider that our traversal always starts at the root (since the tree does not contain any backward edges), gSpan is ready to mine tree structures. Based on the mining efficiency of the pattern-growth-based approach, it is expected that gSpan can achieve good performance in tree-structure mining.

### ***Mining substructure patterns with user-specified constraints***

Various kinds of constraints or specific requirements can be associated with a user's mining request. Rather than developing many case-specific substructure mining algorithms, it is more appropriate to set up a general framework to facilitate such mining.

**Constraint-based mining of frequent substructures.** Constraint-based mining of frequent substructures can be developed systematically, similar to the constraint-based mining of frequent patterns and sequential patterns introduced previously. Take graph mining as an example. With the constraint-based frequent pattern mining framework, graph constraints can also be classified into a few categories, including *pattern antimonotonic*, *pattern monotonic*, *data antimonotonic*, and *succinct*. Efficient

constraint-based mining methods can be developed in a similar way by extending efficient graph-pattern mining algorithms, such as gSpan and CloseGraph.

**Example 5.24. Constraint-based substructure mining.** Let's examine a few commonly encountered classes of constraints to see how the constraint-pushing technique can be integrated into the pattern-growth mining framework.

1. **Element, set, or subgraph containment constraint.** Suppose a user requires that the mined pattern contains a particular set of subgraphs. This is a **succinct constraint** that can be pushed deep into the beginning of the mining process. That is, we can take the given set of subgraphs as a query, perform selection first using the constraint, and then mine on the selected data set by growing (i.e., extending) the patterns from such given set of subgraphs. A similar strategy can be developed if we require that the mined graph pattern must contain a particular set of edges or vertices.
2. **Geometric constraint.** A geometric constraint can be that the angle between each pair of connected edges must be within a range, written as " $C_G = \min\_angle \leq \text{angle}(e_1, e_2, v, v_1, v_2) \leq \max\_angle$ ," where two edges  $e_1$  and  $e_2$  are connected at vertex  $v$  with the two vertices at the other ends as  $v_1$  and  $v_2$ , respectively.  $C_G$  is a **pattern antimonotonic constraint** since if one angle in a graph formed by two edges does not satisfy  $C_G$ , further growth on the graph will never satisfy  $C_G$ . Thus  $C_G$  can be pushed deep into the edge growth process and reject any growth that does not satisfy  $C_G$ .  $C_G$  is also a **data antimonotonic constraint**: for any data graph  $g_i$ , with respect to a candidate subgraph  $g_c$ , if there is no component in the remaining  $g_i$  containing edges satisfying  $C_G$ ,  $g_i$  should not be further considered for  $g_c$  since it will not support  $g_c$ 's further expansion.
3. **Value-sum constraint.** One example of such a constraint can be that the sum of (positive) weights on the edges  $Sum_e$  be within a range from *low* to *high*. This constraint can be split into two constraints,  $Sum_e \geq low$  and  $Sum_e \leq high$ . The former is a **pattern monotonic constraint**, since once it is satisfied, further "growth" on the graph by adding more edges will always satisfy the constraint. The latter is a **pattern antimonotonic constraint**, because once the condition it is not satisfied, further growth of  $Sum_e$  will never satisfy it. Both constraints are **data antimonotonic** in the sense that any data graph that cannot satisfy these constraints during the pattern growth process should be pruned. The constraint pushing strategy can then be worked out easily.  $\square$

Notice that a graph-mining query may contain multiple constraints. For example, we may want to mine graph patterns satisfying constraints on both the geometry and the range of the sum of edge weights. In such cases, we should try to push multiple constraints simultaneously, exploring a method similar to that developed for frequent itemset mining. For the multiple constraints that are difficult to push in simultaneously, customized constraint-based mining algorithms can be developed accordingly.

### **Mining approximate frequent substructures**

An alternative way to reduce the number of patterns to be generated is to mine approximate frequent substructures, which allow slight structural variations. With this technique, we can represent several slightly different frequent substructures using one approximate substructure.

The principle of *minimum description length* (Chapter 6) is adopted in a substructure discovery system called SUBDUE, which mines approximate frequent substructures. It looks for a substructure pattern that can best compress a graph set based on the Minimum Description Length (MDL) principle, which essentially states that the simplest representation is preferred. SUBDUE adopts a constrained

beam search method. It grows a single vertex incrementally by expanding a node in it. At each expansion, it searches for the best total description length: the description length of the pattern and the description length of the graph set with all the instances of the pattern condensed into single nodes. SUBDUE performs approximate matching to allow slight variations of substructures, thus supporting the discovery of approximate substructures.

There should be many different ways to mine approximate substructure patterns. Some may lead to a better representation of the entire set of substructure patterns, whereas others may lead to more efficient mining techniques. More research is needed in this direction.

### ***Mining coherent substructures***

A frequent substructure  $G$  is a **coherent subgraph** if the mutual information between  $G$  and each of its own subgraphs is above some threshold. The number of coherent substructures is significantly smaller than that of frequent substructures. Thus mining coherent substructures can efficiently prune redundant patterns—that is, patterns that are similar to each other and have the similar support. A promising method was developed for mining such substructures. Its experiments demonstrate that in mining spatial motifs from protein structure graphs, the discovered coherent substructures are usually statistically significant. This indicates that coherent substructure mining selects a small subset of features that have high distinguishing power between protein classes.

---

## **5.6 Pattern mining: application examples**

Besides mining frequent patterns in shopping basket analysis, pattern mining captures intrinsic cooccurrence properties of multiple components in massive data sets and plays an important role in various applications. Here we introduce two such cases: phrase mining in massive text data and software bug analysis.

### **5.6.1 Phrase mining in massive text data**

Text data are ubiquitous and plays an essential role at conveying semantics in human communications. However, text data are unstructured and high dimensional. Thus transforming unstructured text into structured units will substantially reduce semantic ambiguity and enhance the power and efficiency at manipulating such data. One such structured unit is semantically meaningful phrases. Although word has been considered as a basic unit at conveying semantics in human languages, a single word (which is often called “unigram”) is often ambiguous at expressing semantic meanings. For example, a single word “*united*” could form *United Airline*, *United States*, *United Kingdom*, and so on when combining with other words, and the word itself may not be an independent semantic unit. However, a phrase like “*United States*” or “*United Airline*” will not lead to any ambiguity. Clearly, phrase represents a natural, meaningful, unambiguous semantic unit. Phrase mining, that is, extracting meaningful phrases from massive text, may transform text data from word granularity to phrase granularity and enhance the power and efficiency at manipulating unstructured data.

Phrase mining plays a key role in *named entity recognition*, a basic natural language processing task. Named entity recognition is often modeled as a sequence labeling problem. To solve such a problem, one can first label the words in a sentence by marking a word with “B,” as the *beginning* of a noun

phrase, the next word could be “I,” representing it is still *in* the same phrase, or “O,” representing it is *out* of the phrase, and some even uses “S” to represent the *singleton* of a phrase (i.e., unigram entity). Such kind of annotation may require human to annotate hundreds of documents as training data and then train a supervised model based on part-of-speech features. However, such kind of training can be costly since it requires human to do a lot of tedious labeling work. Further, this kind of human annotation process is not scalable to a new language, a new domain (e.g., science and engineering), or an emerging application, such as analyzing social media data. Obviously, an automated or semiautomated process could be more desirable for phrase mining.

A simple way to automate the phrase finding process is to take frequent recurring word sequences in text, such as frequent bigrams and tri-grams as phrases. Unfortunately, many frequent bigrams or tri-grams do not form meaningful phrases. For example, “study of” could be a frequent bigram but it is not a meaningful phrase, and the bigram “this paper,” though frequent, may not carry much useful information. Moreover, some highly frequent bigrams or tri-grams may not even occur “independently.” For example, “vector machine” could be a frequently occurring bigram in a machine learning literature corpus but may not exist independently since “vector machine” may only exist as a subsequence of a genuine frequent phrase “support vector machine.”

### ***How to judge the quality of a phrase?***

This leads to an important problem in phrase mining: how to judge a phrase is in high quality with respect to a given corpus. A phrase is a sequence of words that appear contiguously in the text, forming a complete semantic unit in certain context of the given documents. The raw frequency of a phrase is the total count of its occurrences. There is no universally accepted definition of phrase quality. However, it is useful to quantify phrase quality as the probability of a word sequence being a complete semantic unit, meeting the following criteria:

- **Popularity:** A quality phrase should occur with sufficient frequency in the given collection of documents.
- **Concordance:** The collocation of tokens in quality phrases should occur with significantly higher probability than what is expected due to chance. For example, “strong tea” (but not “powerful tea”) could likely be a phrase formed by two collocated words.
- **Informativeness:** A phrase is informative if it is indicative of a specific topic or concept. “This paper” is a popular and concordant phrase but not informative in a research paper corpus.
- **Completeness:** A phrase is complete if it can be interpreted as a whole semantic unit in certain context. For example, “vector machine” does not appear as a complete phrase in a machine learning corpus since almost every occurrence of “vector machine” is just a subcomponent of “support vector machine.” Note that a phrase and its subphrase can both be valid in appropriate context. For example, “relational database system,” “relational database,” and “database system” can all be valid in certain context.

### ***Phrasal segmentation and computing phrase quality***

Phrase quality can be defined to be the possibility of a multiword sequence being a coherent semantic unit, according to the above four criteria. Given a phrase  $v$ , its phrase quality can be defined as:  $Q(v) = p([v]|v) \in [0, 1]$  where  $[v]$  refers to the event that the words in  $v$  compose a phrase. For a single word  $w$ , we define  $Q(w) = 1$ . For phrases,  $Q$  is to be learned from data. For example, a good quality estimator is able to return  $Q(\text{relational database system}) \approx 1$  and  $Q(\text{vector machine}) \approx 0$ .

**Concordance computation.** Concordance contributes significantly to the evaluate the phrase quality since tokens in high quality phrases should cooccur (also called “colocation”) with significantly higher probability than what is expected due to chance. There are multiple measures that can be used to evaluate how a sequence of words that cooccur more frequently than expected in a corpus.

To make phrases with different lengths comparable, we partition each phrase candidate into two disjoint parts in all possible ways and derive effective features measuring their concordance.

Suppose for each word or phrase  $u \in \mathcal{U}$ , we have its raw frequency  $f[u]$ . Its probability  $p(u)$  is defined as

$$p(u) = \frac{f[u]}{\sum_{u' \in \mathcal{U}} f[u']}.$$

Given a phrase  $v \in \mathcal{P}$ , we split it into two most-likely subunits  $\langle u_l, u_r \rangle$  such that *pointwise mutual information* is minimized. Pointwise mutual information quantifies the discrepancy between the probability of their true collocation and the presumed collocation under independence assumption. Mathematically,

$$\langle u_l, u_r \rangle = \arg \min_{u_l \oplus u_r = v} \log \frac{p(v)}{p(u_l)p(u_r)}.$$

With  $\langle u_l, u_r \rangle$ , we can directly use the pointwise mutual information as one of the concordance features.

$$PMI(u_l, u_r) = \log \frac{p(v)}{p(u_l)p(u_r)}.$$

Another feature is also from information theory, called pointwise Kullback-Leibler divergence:

$$PKL(v \parallel \langle u_l, u_r \rangle) = p(v) \log \frac{p(v)}{p(u_l)p(u_r)}.$$

The additional  $p(v)$  is multiplied with pointwise mutual information, leading to less bias toward rare-occurred phrases.

Both features are supposed to be positively correlated with concordance. Concordance can also be evaluated using other statistical measures, such as *t*-test, *z*-test, chi-squared test, and likelihood ratio.

Many of these measures can be used to guide an agglomerative phrasal segmentation process.

**Phrasal segmentation.** A *phrasal segmentation* corresponds to a partition of a word sequence into multiple subsequences, such that every subsequence corresponds to either a single word or a phrase. Phrasal segmentation provides the necessary granularity we need to extract quality phrases. Consider the raw frequency of a phrase is the total count of its occurrences in the original corpus. The total count for a phrase to appear in the *segmented corpus* is called *rectified frequency*.

A sequence’s segmentation may not be unique. A sequence could be ambiguous and may have different interpretations based on different ways of segmentation. For example, “[support vector machine] learning” and “[support vector] [machine learning]” may both be valid partitions, with different meanings. Nevertheless, in most cases, it does not require perfect segmentation, no matter if such a segmentation exists, to extract quality phrases. In a large document collection, the popularly adopted phrases appear many times in a variety of context. Even with a few mistakes or debatable partitions, a reasonably high quality segmentation would retain sufficient support (i.e., rectified frequency) for



these quality phrases. On the other hand, a quality segmentation will unlikely generate partitions like “support [vector machine].” Thus, “vector machine,” even with high raw frequency, is a false phrase since it will have very low rectified frequency.

**Informativeness computation.** Some candidates are unlikely to be informative because they are functional or stopwords. The following stopword-based features can be used to compute informativeness:

- Whether stopwords are located at the beginning or the end of the phrase candidate, which requires a dictionary of stopwords. Phrases that begin or end with stopwords, such as “I am,” are often functional rather than informative.

A more generic feature is to measure the informativeness based on corpus statistics.

- Average inverse document frequency (IDF) computed over words, where IDF for a word  $w$  is computed as

$$\text{IDF}(w) = \log \frac{|\mathcal{C}|}{|\{d \in [D] : w \in C_d\}|},$$

where the IDF score of a word or phrase  $w$  is the logarithm of the total number of documents in the corpus (i.e.,  $|\mathcal{C}|$ ) divided by the number of documents where  $w$  appears (i.e.,  $|\{d \in [D] : w \in C_d\}|$ ). It is a traditional information retrieval measure of how much information a word provides in order to retrieve a small subset of documents from a corpus. In general, quality phrases are expected to have not too small average IDF.

In addition to word-based features, punctuation is frequently used in text to aid interpretations of specific concept or idea. This information is helpful for our task. Specifically, we adopt the following feature:

- Punctuation: probabilities of a phrase in quotes, brackets or capitalized.

Higher probability usually indicates that a phrase is more likely to be informative.

### ***Phrase mining methods***

With such quality measures as guidance, phrase mining can adopt an unsupervised, a weakly supervised, or a distantly supervised approach.

**Unsupervised phrase mining: ToPMine.** ToPMine finds quality phrases based on statistics in the corpus without using any human supervision or annotation. It first mines frequent phrases using a contiguous sequential pattern mining, and then use these phrases to segment each document through an agglomerative phrase construction method. For the frequent phrase mining process, it uses a typical contiguous sequential pattern mining algorithm such as PrefixSpan with the gap between the candidate words set to zero. Then it simply collects aggregate counts for all contiguous words in a corpus that satisfy a certain minimum support threshold.

For the agglomerative phrase construction process, it adopts a bottom-up phrase/word merging process. At each iteration, it makes locally optimal decisions in merging single- and multiword phrases as guided by a statistical significance score (i.e., merging two contiguous phrases such that their merging is of highest significance). The following iteration then considers the newly merged phrase as a single unit and assesses the significance of merging two phrases. The algorithm terminates when the next

	<i>Topic 1</i>	<i>Topic 2</i>	<i>Topic 3</i>	<i>Topic 4</i>	<i>Topic 5</i>
1-grams	problem algorithm optimal solution search solve constraints programming heuristic genetic	word language text speech system recognition character translation sentences grammar	data method algorithm learning clustering classification based features proposed classifier	programming language code type object implementation system compiler java data	data patterns mining rules set event time association stream large
n-grams	genetic algorithm optimization problem solve this problem optimal solution evolutionary algorithm local search search space optimization algorithm search algorithm objective function	natural language speech recognition language model natural language processing machine translation recognition system context free grammars sign language recognition rate character recognition	data sets support vector machine learning algorithm machine learning feature selection paper we propose clustering algorithm decision tree proposed method training data	programming language source code object oriented type system data structure program execution run time code generation object oriented programming java programs	data mining data sets data streams association rules data collection time series data analysis mining algorithms spatio temporal frequent itemsets

**FIGURE 5.21**

Five topics from a 50-topic run of ToPMine on a full DBLP abstracts data set. Overall we see coherent topics and high-quality topical phrases, which can be interpreted as “search/optimization,” “NLP,” “machine learning,” “programming languages,” and “data mining.”

merging with the highest significance does not meet a predetermined significance threshold or when all the terms have been merged into a single phrase. While the frequent phrase mining algorithm satisfies the frequency requirement, the phrase construction algorithm satisfies the collocation and completeness criterion.

By integrating such a phrase mining process with a refined LDA (Latent Dirichlet Allocation)-based topic modeling process, ToPMine generates topic clusters consisting of high quality phrases, without human supervision, as shown in Fig. 5.21.

**Weakly supervised phrase mining: SegPhrase.** It is possible to mine quality phrases without any human supervision; however, it is often more desirable to assist phrase mining with a small set of human-provided labeled data due to various ways to form phrases in diverse domains.

Here we introduce a weakly supervised phrase mining method, called SegPhrase, which takes a corpus with a small set  $L$  of labeled quality phrases and  $\bar{L}$  of inferior ones as the input and generates a ranked list of phrases with decreasing quality, together with a segmented corpus, as output. Taking a small set of labeled data, one can work with various classifiers that can be effectively trained with a small set of labeled data and output a probabilistic score between 0 and 1. For instance, we can adopt the *random forest* algorithm, which is a typical ensemble-based classification algorithm to be introduced in Chapter 7, and is effective to train a quality classifier with a small number of positive (i.e., quality phrases) and negative labels (i.e., inferior phrases). The ratio of positive predictions among all decision trees can be interpreted as a phrase’s quality estimation. The experiments show that 200–300 labels are enough to train a satisfactory classifier. Classification results will be fed into a phrasal segmentation process to compute rectified frequency of each phrase. Combined with phrase quality estimation, bad phrases with high raw frequency get removed as their rectified frequencies approach zero. Furthermore,

Conference	SIGMOD		SIGKDD	
Method	SegPhrase+	Chunking	SegPhrase+	Chunking
1	data base	data base	data mining	data mining
2	database system	database system	data set	association rule
3	relational database	query processing	association rule	knowledge discovery
4	query optimization	query optimization	knowledge discovery	frequent itemset
5	query processing	relational database	time series	decision tree
...	...	...	...	...
51	sql server	database technology	association rule mining	search space
52	relational data	database server	rule set	domain knowledge
53	data structure	large volume	concept drift	important problem
54	join query	performance study	knowledge acquisition	concurrency control
55	web service	web service	gene expression data	conceptual graph
...	...	...	...	...
201	high dimensional data	efficient implementation	web content	optimal solution
202	location based service	sensor network	frequent subgraph	semantic relationship
203	xml schema	large collection	intrusion detection	effective way
204	two phase locking	important issue	categorical attribute	space complexity
205	deep web	frequent itemset	user preference	small set
...	...	...	...	...

FIGURE 5.22

Interesting phrases mined from papers published in SIGMOD and SIGKDD conferences.

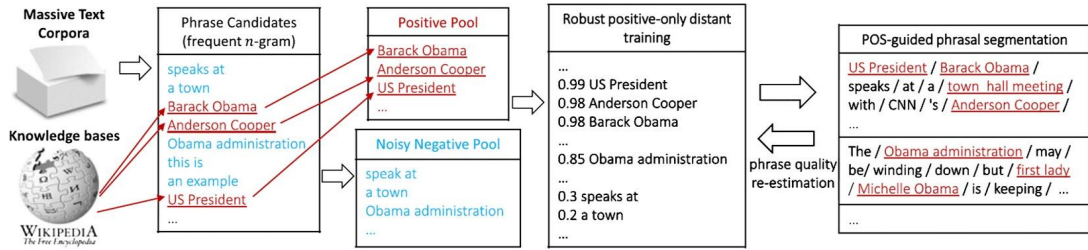
rectified phrase frequencies can be fed back to generate additional features and improve the phrase quality estimation.

Such a phrase quality estimation and phrasal segmentation form a mutually enhancement process. A better phrase quality estimator can guide a better segmentation, and a better segmentation will further improve phrase quality estimation. As a result, misclassified phrase candidates can get mostly corrected after retraining the classifier. Therefore, such an integrated, mutual enhancement framework is expected to leverage the quality on both quality estimation and phrasal segmentation and address all the four phrase quality requirements organically. The experiments show that the benefits brought by retraining the classifier with rectified frequency just need one round iteration, leaving performance curves over the next several iterations similar.

With only a small set of human crafted training data, SegPhrase has shown high performance on generating a large set of quality phrases from different kinds of corpora, in multiple natural languages. A performance study on the quality of phrases generated by different methods also show that SegPhrase outperforms many other phrase mining or chunking methods. Fig. 5.22 shows a set of interesting phrases mined from a large set of papers published in SIGMOD and SIGKDD conferences, which obviously outperforms the set of phrases generated by a phrase chunking methods adopted in JATE (<https://code.google.com/p/jatetoolkit>).

**Distantly supervised phrase mining: AutoPhrase.** AutoPhrase is an automated phrase mining framework, which further avoids additional manual labeling effort and enhances the performance with two techniques: (1) robust positive-only distant training and (2) POS-guided phrasal segmentation.

Many high-quality phrases are freely available in general knowledge bases, and they can be easily obtained to a scale that is much larger than that produced by human experts. Domain-specific corpora usually contain quality phrases encoded either in general knowledge bases or in domain-specific knowledge bases (e.g., biomedical knowledge bases). We can leverage the existing high-quality phrases from



**FIGURE 5.23**

Autophrase: automated phrase mining by distant supervision.

general knowledge bases (e.g., Wikipedia and Freebase) or domain-specific ones as available “positive” labels for distant training.

Knowledge bases, however, rarely, if ever, identify inferior phrases that fail to meet our criteria. An important observation is that the number of phrase candidates, based on n-grams, is huge, and the majority of them are actually of inferior quality (e.g., “Francisco opera and”). In practice, based on the experiments, among millions of phrase candidates, usually, only about 10% are in good quality. Therefore phrase candidates that are derived from the given corpus but that fail to match any high-quality phrase derived from the given knowledge base are used to populate a large but noisy negative pool. A framework for exploring knowledge-bases in distant supervision is outlined in Fig. 5.23.

Directly training a classifier based on the noisy label pools is not a wise choice: some phrases of high quality from the given corpus may have been missed (i.e., inaccurately binned into the negative pool) simply because they were not present in the knowledge base. Instead, a clever way is to utilize an ensemble classifier that averages the results of  $T$  independently trained base classifiers. For each base classifier,  $K$  phrase candidates are randomly drawn with replacement from the positive pool and the negative pool respectively. This size- $2K$  subset of the full set of all phrase candidates is called a perturbed training set, because the labels of some quality phrases are switched from positive to negative. In order for the ensemble classifier to alleviate the effect of such noise, we need to use base classifiers with the lowest possible training errors. An unpruned decision tree can be grown to the point of separating all phrases to meet this requirement. The phrase quality score of a particular phrase is computed as the proportion of all decision trees that predict that phrase is a quality phrase.

To further enhance the performance of phrase mining, a pretrained part-of-speech (POS) tagger can be incorporated to take advantage of linguistic knowledge. The POS-guided phrasal segmentation leverages the shallow syntactic information in POS tags to guide the phrasal segmentation model locating the boundaries of phrases more accurately. POS tags may provide shallow, language-specific knowledge, which may help boost phrase detection accuracy, especially at syntactic constituent boundaries for that language. For example, suppose the whole POS tag sequence is “ $NN NN NN VB DT NN$ .” A good POS sequence quality estimator might return  $Q(NN NN NN) \approx 1$  and  $Q(NN VB) \approx 0$  where  $NN$  refers to singular or mass noun (e.g., database),  $VB$  means verb in the base form (e.g., *is*), and  $DT$  is for determiner (e.g., *the*).

The extensive experiments show that AutoPhrase is domain-independent, outperforms other phrase mining methods, and supports multiple languages (e.g., English, Spanish, and Chinese) effectively, with minimal human effort.

## 5.6.2 Mining copy and paste bugs in software programs

Pattern mining has found its interesting applications in software program analysis since the source code of a program module consists of long sequences of programming statements and the execution of a software program forms a sequence of executed codes. A large software program may consist of many program modules and its executions may leave tremendous amount of execution traces. Manual examination of such programming code or execution traces could be tedious and costly. Frequent and sequential pattern mining could provide useful tools to uncover interesting regularities or irregularities. Typical examples may include mining software bugs from source programs or execution sequences, mining programming rules from program revision histories, mining software function precedence protocols by examination of frequent subsequences, and revealing neglected conditions by frequent itemset or subgraph mining.

Here we examine one example which explores pattern discovery to find *copy-pasting* software program bugs from source code. Because a lot of program fragments may share some similar functions, code copy-pasting has become popular in software programming. A programmer may highlight a few lines of program code at one location of a program, copy these lines, paste them to another location in a program, and then perform appropriate modifications of the pasted programming code.

Copy-pasting is a common programming practice. Some statistic shows that about 12% of program code in the Linux file system and about 19% in the X Window system are copy-pasted. However, copy-pasted code is error-prone. Due to programmer's carelessness, changes on pasted code may not always be done consistently throughout. Such "forget-to-change" bugs can be common and lead to buggy programs.

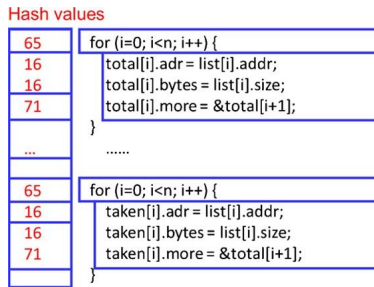
Interestingly, such copy-pasting bugs can be mined by transforming source code into a sequence data set, on which sequential pattern mining can be conducted to identify likely mismatched identifier names, and hence catch the "forget-to-change" bugs.

Let's examine such an example. Fig. 5.24 shows a program module that contains a copy-pasting bug: The first for-loop block is copied and pasted to form the second for-loop block, and every occurrence of the pasted identifier "total" should be consistently changed to "taken." Unfortunately, the last change of "total" was missing, leading to a bug.

```
void __init prom_meminit(void)
{
    .....
    for (i=0; i<n; i++) {
        total[i].adr = list[i].adr;
        total[i].bytes = list[i].size;
        total[i].more = &total[i+1];
    }
    .....
    for (i=0; i<n; i++) {
        taken[i].adr = list[i].adr;
        taken[i].bytes = list[i].size;
        taken[i].more = &total[i+1];
    }
}
```

**FIGURE 5.24**

A program fragment that contains a copy-pasting bug.

**FIGURE 5.25**

Transform a sequence of statements into a sequence of numbers.

The key to find such a programming bug is to identify the corresponding copy-pasting blocks and examine whether the modifications of the statements in the pasted block were conducted consistently. “*How to automatically identify such copy-pasting blocks?*” An interesting strategy is to map a long source program into a long sequence of numbers, where each statement is represented by a number. If a statement being copied and that being pasted can be mapped to the same number, the blocks of statements being copied and pasted will show a similar sequence, and a sequential pattern mining algorithm will be able to identify such copy-pasting blocks.

Let’s see how two statements, one copied and one pasted, can be mapped to the same number, by a clever design. To map the statements “total[i].adr = list[i].adr;” and “taken[i].adr = list[i].adr;” to the same number, we may design the following mapping rules: (1) the identifiers of the same type are mapped to the same token, (2) different operators, constants, and key words are mapped to different tokens, and (3) a statement consisting of the same sequence of tokens is mapped to the same number and that consisting of a different sequence of tokens is mapped to different numbers.

Following this set of rules, the name identifiers “total,” “list,” “taken,” “i,” and “adr” are mapped to the same token (e.g., 3). Similarly, we may have “[” mapped to 5, “]” to 6; “;” to 8, “=” to 9, “;” to 1, and “&” to 2. Then the statement “total[i].adr = list[i].adr;” is mapped to a sequence of tokens “3 5 3 6 8 3 9 3 5 3 6 8 3 1.” Such a sequence could be mapped (e.g., using a hash function) to a number (e.g., 16). By such mapping, each statement in a program is mapped to a number; and two statements with similar functions, such as “total[i].adr = list[i].adr;” and “taken[i].adr = list[i].adr;” will be mapped to the same number 16, despite their different identifier names, since they have the same sequence of tokens. Thus, a sequence of statements shown in Fig. 5.24 are transformed into a sequence of numbers (or hash values) as shown in Fig. 5.25.

The above-described transformation maps a program to a sequence of numbers. One can further cut a long sequence by blocks. Thus, our program code in Fig. 5.24 will be transformed into a sequence data set: (65), (16, 16, 71), . . . , (65), (16, 16, 71). By sequential pattern mining, one can find the sequential pattern “(65), (16, 16, 71).”

Note that even some other statements are inserted in the middle of such a sequence of statements, a typical sequential pattern mining algorithm such as PrefixSpan will still be able to find the correct sequential pattern. For example, mining the two sequences (16, 16, 71) and (16, 18, 16, 25, 71) will generate the same frequent subsequence (16, 16, 71). This will allow the method to detect copy-pasting

bugs even if a few other statements are inserted into the pasted program code as long as such an insertion are confined to a predefined maximal gap (for sequential pattern mining).

After identification of copy-pasting blocks, the next step is to find inconsistent modifications in the pasted statements. This can be done easily by comparing the two copy-pasting blocks. If the majority occurrences of one identifier (e.g., “total”) have been changed to another one (e.g., “taken”), the minority unchanged (e.g., the retained “total”) is likely the bug. An “unchanged ratio” can be easily set to identify such “forget to change” errors.

A software bug mining program, CP-Miner, adopting the mining methodology described here, has successfully uncovered many copy-pasting bugs in Linux, Apache, and other open source programs, out of millions of lines of code.

---

## 5.7 Summary

- The **scope** of frequent pattern mining research reaches far beyond the basic concepts and methods introduced in Chapter 4 for mining frequent itemsets and associations. This chapter presented a road map of the field, where topics are organized with respect to the kinds of patterns and rules that can be mined, mining methods, and applications.
- In addition to mining for basic frequent itemsets and associations, **advanced forms of patterns** can be mined such as multilevel associations and multidimensional associations, quantitative association rules, rare patterns, and negative patterns. We can also mine high-dimensional patterns and compressed or approximate patterns.
- **Multilevel associations** involve data at more than one abstraction level (e.g., “*buys computer*” and “*buys laptop*”). These may be mined using multiple minimum support thresholds. **Multidimensional associations** contain more than one dimension. Techniques for mining such associations differ in how they handle repetitive predicates. **Quantitative association rules** involve quantitative attributes. Discretization, clustering, and statistical analysis that discloses exceptional behavior can be integrated with the pattern mining process.
- **Rare patterns** occur rarely but are of special interest. **Negative patterns** are patterns with components that exhibit negatively correlated behavior. Care should be taken in the definition of negative patterns, with consideration of the null-invariance property. Rare and negative patterns may highlight exceptional behavior in the data, which is likely of interest.
- **Constraint-based mining** strategies can be used to help direct the mining process toward patterns that match users’ intuition or satisfy certain constraints. Many user-specified constraints can be pushed deep into the mining process. Constraints can be categorized into **pattern-pruning** and **data-pruning** constraints. Properties of such constraints include *monotonicity*, *antimonotonicity*, *data-antimonotonicity*, and *succinctness*. Constraints with such properties can be properly incorporated into efficient pattern mining processes.
- Methods have been developed for mining patterns in **high-dimensional space**. This includes a pattern growth approach based on *row enumeration* for mining data sets where the number of dimensions is large and the number of data tuples is small (e.g., for microarray data), as well as mining **colossal patterns** (i.e., patterns of very long length) by a *Pattern-Fusion* method.
- To reduce the number of patterns returned in mining, we can instead mine compressed patterns or approximate patterns. *Compressed patterns* can be mined with representative patterns defined based on

the concept of clustering, and *approximate patterns* can be mined by extracting **redundancy-aware top- $k$  patterns** (i.e., a small set of  $k$ -representative patterns that have not only high significance but also low redundancy with respect to one another).

- **Sequential pattern mining** is the mining of frequently occurring ordered events or subsequences as patterns. The Apriori pruning principles can be used for pruning in sequential pattern mining, which leads to efficient sequential mining algorithms, such as GSP, SPADE and PrefixSpan. CloSpan is an efficient method for mining closed sequential patterns. Efficient methods have also been developed for mining multidimensional and multilevel sequential patterns and constraint-based sequential pattern mining.
- **Subgraph pattern mining** is the mining of frequent subgraphs in a collection of graphs. The Apriori pruning principles can be used for pruning in subgraph pattern mining, which leads to efficient subgraph mining algorithms, such as AGM, FSG and gSpan (a pattern-growth approach). CloseGraph is an efficient method for mining closed subgraph patterns. Efficient methods have also been developed for mining other frequent substructure patterns, such as directed graphs, tree structures, and disconnected graphs, mining frequent substructures with user-specified constraints, mining approximate frequent substructures, and mining coherent substructures.
- Pattern mining has broad and interesting applications. Besides popular market analysis application, this chapter discusses phrase mining from massive text and software copy-pasting bug mining in software engineering. For phrase mining, an unsupervised method ToPMine, a weakly supervised method SegPhrase and a distantly supervised method AutoPhrase are introduced. For software copy-pasting bug mining, a methodology adopted in CP-Miner is introduced.

---

## 5.8 Exercises

- 5.1. Propose and outline a **level-shared mining** approach to mining multilevel association rules in which each item is encoded by its level position. Design it so that an initial scan of the database collects the count for each item *at each concept level*, identifying frequent and subfrequent items. Comment on the processing cost of mining multilevel associations with this method in comparison to mining single-level associations.
- 5.2. Suppose, as manager of a chain of stores, you would like to use sales transactional data to analyze the effectiveness of your store's advertisements. In particular, you would like to study how specific factors influence the effectiveness of advertisements that announce a particular category of items on sale. The factors to study are the *region* in which customers live and the *day-of-the-week* and *time-of-the-day* of the ads. Discuss how to design an efficient method to mine the transaction data sets and explain how **multidimensional** and **multilevel mining** methods can help you derive a good solution.
- 5.3. **Quantitative association rules** may disclose exceptional behaviors within a data set, where "exceptional" can be defined based on statistical theory. For example, Section 5.1.3 shows the association rule

$$gender = female \Rightarrow mean\_wage = \$7.90/hr \text{ (overall\_mean\_wage} = \$9.02/hr),$$

which suggests an exceptional pattern. The rule states that the average wage for females is only \$7.90 per hour, which is a significantly lower wage than the overall average of \$9.02 per hour.



Discuss how such quantitative rules can be discovered systematically and efficiently in large data sets with quantitative attributes.

- 5.4. In multidimensional data analysis, it is interesting to extract pairs of *similar* cell characteristics associated with substantial changes in measure in a data cube, where cells are considered *similar* if they are related by roll-up (i.e., *ancestors*), drill-down (i.e., *descendants*), or 1-D mutation (i.e., *siblings*) operations. Such an analysis is called **cube gradient analysis**. Suppose the measure of the cube is *average*. A user poses a set of *probe cells* and would like to find their corresponding sets of *gradient cells*, each of which satisfies a certain gradient threshold. For example, find the set of corresponding gradient cells that have an average sale price greater than 20% of that of the given probe cells. Develop an algorithm that mines the set of constrained gradient cells efficiently in a large data cube.
- 5.5. Section 5.1.5 presented various ways of defining negatively correlated patterns. Consider Definition 5.3: “Suppose that itemsets  $X$  and  $Y$  are both frequent, that is,  $sup(X) \geq min\_sup$  and  $sup(Y) \geq min\_sup$ , where  $min\_sup$  is the minimum support threshold. If  $(P(X|Y) + P(Y|X))/2 < \epsilon$ , where  $\epsilon$  is a negative pattern threshold, then pattern  $X \cup Y$  is a **negatively correlated pattern**.” Design an efficient pattern growth algorithm for mining the set of negatively correlated patterns.
- 5.6. Prove that each entry in the following table correctly characterizes its corresponding **rule constraint** for frequent itemset mining.

	Rule Constraint	Antimonotonic	Monotonic	Succinct
a.	$v \in S$	no	yes	yes
b.	$S \subseteq V$	yes	no	yes
c.	$min(S) \leq v$	no	yes	yes
d.	$range(S) \leq v$	yes	no	no

- 5.7. The price of each item in a store is nonnegative. The store manager is only interested in mining the rules, following the constraints given below. For each of the following cases, identify the kinds of **constraints** they represent and briefly discuss how to mine such association rules using **constraint-based pattern mining**.
- Containing at least one Blu-ray DVD movie.
  - Containing items with a sum of the prices that is less than \$150.
  - Containing one free item and other items with a sum of the prices that is at least \$200, whereas the average price of all the items is between \$100 and \$500.
- 5.8. Section 5.1.4 introduced a core Pattern-Fusion method for **mining high-dimensional data**. Explain why a long pattern, if existing in the data set, is likely to be discovered by this method.
- 5.9. Section 5.2.1 defined a **pattern distance measure** between closed patterns  $P_1$  and  $P_2$  as

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|},$$

where  $T(P_1)$  and  $T(P_2)$  are the supporting transaction sets of  $P_1$  and  $P_2$ , respectively. Is this a valid distance metric? Show the derivation to support your answer.

- 5.10. Association rule mining often generates a large number of rules, many of which may be similar, thus not containing much novel information. Design an efficient algorithm that **compresses** a large set of patterns into a small compact set. Discuss whether your mining method is robust under different pattern similarity definitions.

- 5.11. Frequent pattern mining may generate many superfluous patterns. Therefore, it is important to develop methods that mine compressed patterns. Suppose a user would like to obtain only  $k$  patterns (where  $k$  is a small integer). Outline an efficient method that generates the  $k$  **most representative patterns**, where more distinct patterns are preferred over very similar patterns. Illustrate the effectiveness of your method using a small data set.
- 5.12. Sequential pattern mining is to mine sequential patterns for a set of items occurring in sequence order. In practice, people may like to find sequential patterns for types of items instead of for concrete items, such as sequence patterns formed by high-level concepts. For example, instead of finding sequential patterns composed of concrete models of i-phones in shopping transactions, but finding patterns composed of Apple products, smart-phones, electronics, and so on. Outline an efficient sequential pattern mining algorithm that **simultaneously mines sequential patterns at multiple levels of abstraction**.
- 5.13. At studying customer shopping sequences, one may find if a customer buys a sequence of products from one company, the chance for him/her to buy the products of the similar kind from another company will be much reduced. Can you outline an efficient algorithm that will be able to capture such **negatively associated sequential patterns**?
- 5.14. Our study of subgraph pattern mining has been on how to mine frequent substructures from a collection of graph data sets. The current Web page structures (e.g., Wikipedia) or social networks may form one or a small number of gigantic network structures. One may need to find frequent common substructures from one gigantic network. Outline an efficient method that **finds top- $k$  large substructural patterns in a massive network**.
- 5.15. In this chapter, we introduce an effective method for mining copy-and-paste bugs in software programs. Typically, a software program may take different inputs which may lead to different program execution sequences. For some inputs, the program execution finishes successfully but for some other inputs, the program fails (e.g., getting a core dump). Can you work out an algorithm that may use sequential pattern mining to identify what execution sequences may be used to distinguish program failure from program success?

---

## 5.9 Bibliographic notes

This chapter described various ways in which the basic techniques of frequent itemset mining (presented in Chapter 4) have been extended. One line of extension is mining multilevel and multidimensional association rules. Multilevel association mining was studied in Srikant and Agrawal [SA95] and Han and Fu [HF95]. In Srikant and Agrawal [SA95], such mining was studied in the context of *generalized association rules*, and an R-interest measure was proposed for removing redundant rules. Mining multidimensional association rules using static discretization of quantitative attributes and data cubes was studied by Kamber, Han, and Chiang [KHC97].

Another line of extension is to mine patterns on numeric attributes. Srikant and Agrawal [SA96] proposed a nongrid-based technique for mining quantitative association rules, which uses a measure of partial completeness. Mining quantitative association rules based on rule clustering was proposed by Lent, Swami, and Widom [LSW97]. Techniques for mining quantitative rules based on  $x$ -monotone and rectilinear regions were presented by Fukuda, Morimoto, Morishita, and Tokuyama [FMMT96] and Yoda et al. [YFM<sup>+</sup>97]. Mining (distance-based) association rules over interval data was proposed by

Miller and Yang [MY97]. Aumann and Lindell [AL99] studied the mining of quantitative association rules based on a statistical theory to present only those rules that deviate substantially from normal data.

Mining rare patterns by pushing group-based constraints was proposed by Wang, He, and Han [WHH00]. Mining negative association rules was discussed by Savasere, Omiecinski, and Navathe [SON98] and by Tan, Steinbach, and Kumar [TSK05].

Constraint-based mining directs the mining process toward patterns that are likely of interest to the user. The use of metarules as syntactic or semantic filters defining the form of interesting single-dimensional association rules was proposed in Klemettinen et al. [KMR<sup>+</sup>94]. Metarule-guided mining, where the metarule consequent specifies an action (e.g., Bayesian clustering or plotting) to be applied to the data satisfying the metarule antecedent, was proposed in Shen, Ong, Mitbender, and Zaniolo [SOMZ96]. A relation-based approach to metarule-guided mining of association rules was studied in Fu and Han [FH95].

Methods for constraint-based mining using pattern pruning constraints were studied by Ng, Lakshmanan, Han, and Pang [NLHP98]; Lakshmanan, Ng, Han, and Pang [LNHP99]; and Pei, Han, and Lakshmanan [PHL01]. Constraint-based pattern mining by data reduction using data pruning constraints was studied by Bonchi, Giannotti, Mazzanti, and Pedreschi [BGMP03] and Zhu, Yan, Han, and Yu [ZYHY07]. An efficient method for mining constrained correlated sets was given in Grahne, Lakshmanan, and Wang [GLW00]. A dual mining approach was proposed by Bucila, Gehrke, Kifer, and White [BGKW03]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in Anand and Kahn [AK93]; Dhar and Tuzhilin [DT93]; Hoschka and Klösgen [HK91]; Liu, Hsu, and Chen [LHC97]; Silberschatz and Tuzhilin [ST96]; and Srikant, Vu, and Agrawal [SVA97].

Traditional pattern mining methods encounter challenges when mining high-dimensional patterns, with applications like bioinformatics. Pan et al. [PCT<sup>+</sup>03] proposed CARPENTER, a method for finding closed patterns in high-dimensional biological data sets, which integrates the advantages of vertical data formats and pattern growth methods. Pan, Tung, Cong, and Xu [PTCX04] proposed COBBLER, which finds frequent closed itemsets by integrating row enumeration with column enumeration. Liu, Han, Xin, and Shao [LHXS06] proposed TDClose to mine frequent closed patterns in high-dimensional data by starting from the maximal rowset, integrated with a row-enumeration tree. It uses the pruning power of the minimum support threshold to reduce the search space. For mining rather long patterns, called *colossal patterns*, Zhu et al. [ZYH<sup>+</sup>07] developed a core Pattern-Fusion method that leaps over an exponential number of intermediate patterns to reach colossal patterns.

To generate a reduced set of patterns, recent studies have focused on mining compressed sets of frequent patterns. Closed patterns can be viewed as a lossless compression of frequent patterns, whereas maximal patterns can be viewed as a simple lossy compression of frequent patterns. Top- $k$  patterns, such as by Wang, Han, Lu, and Tsvetkov [WHLT05], and error-tolerant patterns, such as by Yang, Fayyad, and Bradley [YFB01], are alternative forms of interesting patterns. Afrati, Gionis, and Mannila [AGM04] proposed to use  $k$ -itemsets to cover a collection of frequent itemsets. For frequent itemset compression, Yan, Cheng, Han, and Xin [YCHX05] proposed a profile-based approach, and Xin, Han, Yan, and Cheng [XHYC05] proposed a clustering-based approach. By taking into consideration both pattern significance and pattern redundancy, Xin, Cheng, Yan, and Han [XCYH06] proposed a method for extracting redundancy-aware top- $k$  patterns.

Automated semantic annotation of frequent patterns is useful for explaining the meaning of patterns. Mei et al. [MXC<sup>+</sup>07] studied methods for semantic annotation of frequent patterns.

An important extension to frequent itemset mining is mining sequence and structural data. This includes mining sequential patterns (Agrawal and Srikant [AS95]; Pei et al. [PHMA<sup>+</sup>01,PHMA<sup>+</sup>04]; and Zaki [Zak01]); mining frequent episodes (Mannila, Toivonen, and Verkamo [MTV97]); mining structural patterns (Inokuchi, Washio, and Motoda [IWM98]; Kuramochi and Karypis [KK01]; and Yan and Han [YH02]); mining cyclic association rules (Özden, Ramaswamy, and Silberschatz [ORS98]); intertransaction association rule mining (Lu, Han, and Feng [LHF98]); and calendric market basket analysis (Ramaswamy, Mahajan, and Silberschatz [RMS98]). Although the major graph pattern mining studies are on mining frequent graph patterns in a collection of graphs, there are also studies on mining large substructural patterns in a single large network, such as Zhu et al. [ZQL<sup>+</sup>11].

Pattern mining has been extended to help effective data classification and clustering. Pattern-based classification (Liu, Hsu, and Ma [LHM98] and Cheng, Yan, Han, and Hsu [CYHH07]) is discussed in Chapter 7. Pattern-based cluster analysis (Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98] and Wang, Wang, Yang, and Yu [WWYY02]) is discussed in Chapter 9.

Pattern mining also helps many other data analysis and processing tasks such as cube gradient mining and discriminative analysis (Imielinski, Khachiyan, and Abdulghani [IKA02]; Dong et al. [DHL<sup>+</sup>04]; Ji, Bailey, and Dong [JBD05]), discriminative pattern-based indexing (Yan, Yu, and Han [YYH05]), and discriminative pattern-based similarity search (Yan, Zhu, Yu, and Han [ZYH06]).

Pattern mining has been extended to mining spatial, temporal, time-series, multimedia data, and data streams. Mining spatial association rules or spatial collocation rules was studied by Koperski and Han [KH95]; Xiong et al. [XSH<sup>+</sup>04]; and Cao, Mamoulis, and Cheung [CMC05]. Pattern-based mining of time-series data is discussed in Shieh and Keogh [SK08] and Ye and Keogh [YK09]. There are many studies on pattern-based mining of multimedia data such as Zaïane, Han, and Zhu [ZH00] and Yuan, Wu, and Yang [YWY07]. Methods for mining frequent patterns on stream data have been proposed by many researchers, including Manku and Motwani [MM02]; Karp, Papadimitriou, and Shenker [KPS03]; and Metwally, Agrawal, and El Abbadi [MAA05].

Pattern mining has broad applications. Application areas include computer science such as software bug analysis, sensor network mining, and performance improvement of operating systems. For example, CP-Miner by Li, Lu, Myagmar, and Zhou [LLMZ04] uses pattern mining to identify copy-pasted code for bug isolation. PR-Miner by Li and Zhou [LZ05] uses pattern mining to extract application-specific programming rules from source code. Discriminative pattern mining is used for program failure detection to classify software behaviors (Lo et al. [LCH<sup>+</sup>09]) and for troubleshooting in sensor networks (Khan et al. [KLA<sup>+</sup>08]).

As another pattern mining application, phrase mining from massive text data has been studied in recent years. An unsupervised phrase mining method ToPMine, which explores frequent contiguous patterns is developed by El-Kishki et al. [EKSW<sup>+</sup>14]; a weakly supervised phrase mining method SegPhrase, which explores phrasal segmentation and pattern-guided classification is developed by Liu et al. [LSW<sup>+</sup>15]; AutoPhrase, which uses Wikipedia as a source for distant supervision for phrase mining is introduced by Shang et al. [SLJ<sup>+</sup>18]; and UCPhrase that explores the information derived from pretrained language models is developed by Gu et al. [GWB<sup>+</sup>21].

This page intentionally left blank

# Classification: basic concepts and methods

**Classification is a form** of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical (discrete, unordered) class labels. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or identify the early sign of cognitive impairment based on a patient’s functional magnetic resonance imaging (fMRI) scan, or help a self-driving car automatically recognize various road signs. Such analysis can help provide us with a better understanding of the data at large. Many classification methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Traditional classification algorithms typically assume a small or medium data size. Modern classification techniques have built on such work, developing scalable classification and prediction techniques capable of handling very large amounts of data. Classification belongs to supervised learning and is closely connected to many other data mining tasks. Classification has numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, medical diagnosis, and many more.

We start off by introducing the main ideas of classification in Section 6.1. In the rest of this chapter, you will learn the basic techniques for data classification such as how to build decision tree classifiers (Section 6.2), Bayes classifiers (Section 6.3), lazy learners (Section 6.4), and linear classifiers (Section 6.5). Section 6.6 discusses how to evaluate and compare different classifiers. Various measures of accuracy are given, as well as techniques for obtaining reliable accuracy estimates. Methods for improving classifier accuracy are presented in Section 6.7, including ensemble methods and class-imbalanced data (i.e., where the main class of interest is rare).

---

## 6.1 Basic concepts

We introduce the concept of classification in Section 6.1.1. Section 6.1.2 describes the general approach to classification as a two-step process. In the first step, we build a classification model based on previous data. In the second step, we determine if the model’s accuracy is acceptable, and if so, we use the model to classify new data.

### 6.1.1 What is classification?

A bank loans officer needs analysis of her data to learn which loan applicants are “safe” and which are “risky” for the bank, and her colleague from the risk management department wishes to detect fraudulent transactions. A marketing manager at an electronics store needs data analysis to help guess whether a customer with a given profile will buy a new computer, or understand the *sentiment* of social media posts regarding a newly released product, or detect *fake reviews* about a new product from an

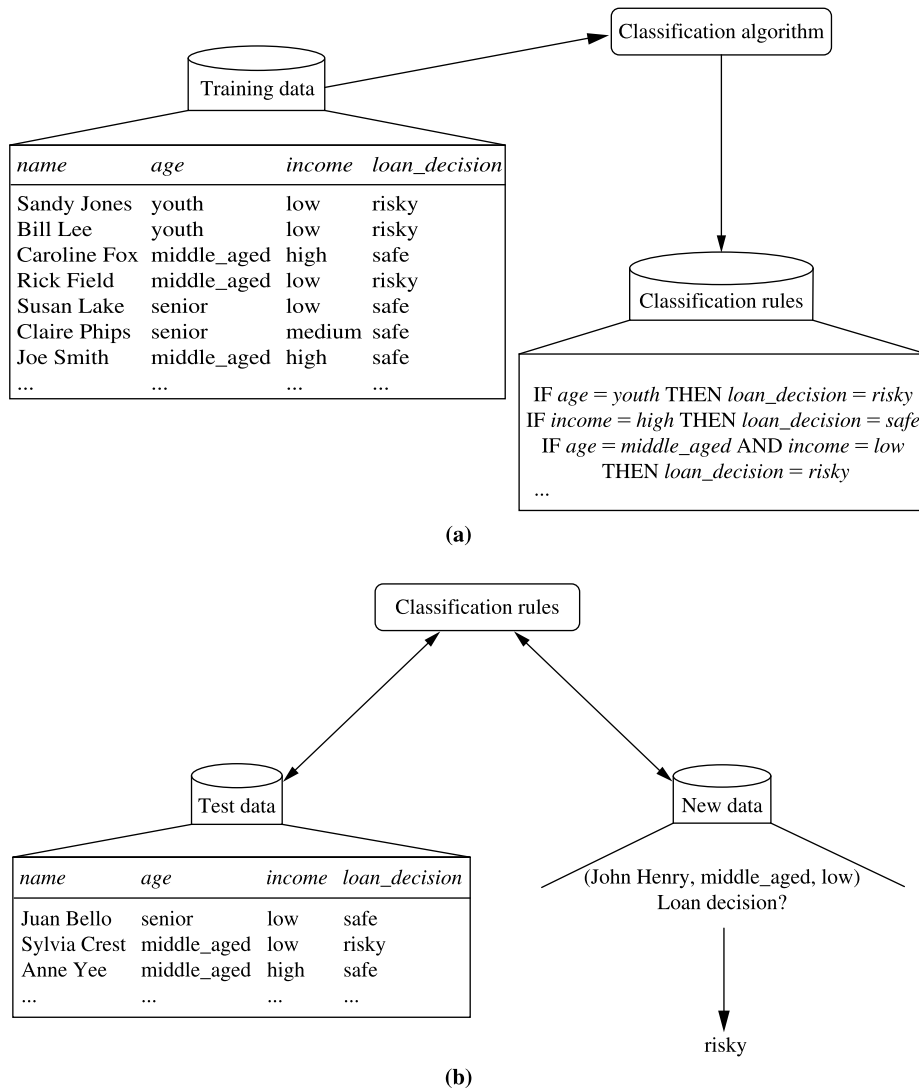
online review site, or identify a subscribed customer who is likely to switch to a competitive electronics store (i.e., churn prediction). An IT security analyst wants to know if the network system is under attack (intrusion detection) or if a given application is contaminated with malware (malware detection). A teacher wishes to know if a student enrolled in an online course will drop out before she completes the course. A talent recruiter wants to know if an individual is looking for the next career move. A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive, a cardiologist wants to identify the patient who is likely to have a congestive heart failure based on her chronic medical history, a neuroscientist wants to identify the early sign of cognitive impairment (which could lead to, say Alzheimer's disease) based on a patient's functional magnetic resonance imaging (fMRI) scan. An intelligent question-answering system needs to understand what type of question the user is asking (question classification), as the first step to automatically provide a high-quality answer. A self-driving car needs to automatically recognize various road signs (e.g., 'stop,' 'detour,' etc.). A physicist needs to identify *high energy event* from massive experiment data, which might lead to new discoveries. Law enforcement wishes to predict the crime hot spot so that the precaution measures can be taken proactively.

In each of these examples, the data analysis task is **classification**, where a model or **classifier** is constructed to predict *class (categorical) labels*, such as "safe" or "risky" for the loan application data; or "positive" or "negative" for sentiment classification; or "yes" or "no" for the marketing data; or "dropout" or "stay" for online course enrollment, or "treatment A," "treatment B," or "treatment C" for the medical data; or various question types for a question-answering system. These categories can be represented by discrete values, where the ordering among values has no meaning. For example, the values 1, 2, and 3 may be used to represent treatments A, B, and C, where there is no ordering implied among this group of treatment regimes.

Suppose that the marketing manager wants to predict how much a given customer will spend during a sale; or a realtor might be interested in knowing the average house pricing of the next year in different residential areas; or a career planner wants to forecast the average yearly income of students immediately after graduating from the college in different majors. This kind of data analysis task is an example of **numeric prediction**, where the model constructed predicts a *continuous-valued function, or ordered value*, as opposed to a class label. **Regression analysis** is a statistical methodology that is most often used for numeric prediction; hence the two terms tend to be used synonymously, although other methods for numeric prediction exist. **Ranking** is another type of numerical prediction where the model predicts the ordered values (i.e., ranks), for example, a web search engine (e.g., Google) ranks the relevant webpages with respect to a given query, with the higher-ranked webpages being more relevant to the query. Classification and numeric prediction are the two major types of **prediction problems**. This chapter primarily focuses on classification. It is worth pointing out that classification and numerical prediction (e.g., regression) are closely related to each other. Many classification techniques can be modified for the purpose of regression. We will see some examples, including regression trees (Section 6.2), lazy learners (Section 6.4.1), linear regression (Section 6.5), and gradient tree boosting (Section 6.7.1).

### 6.1.2 General approach to classification

"How does classification work?" **Data classification** is a two-step process, consisting of a *learning step* (where a classification model is constructed) and a *classification step* (where the model is used

**FIGURE 6.1**

The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan\_decision*, and the learned model or classifier is represented in the form of classification rules.

(b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is acceptable, the rules can be applied to the classification of new data tuples.

to predict class labels for given data). The process is shown for the loan application data in Fig. 6.1. The data are simplified for illustrative purposes. In reality, we may expect many more attributes to be considered.



In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the **learning step** (also known as the training phase), where a classification algorithm builds the classifier by analyzing or “learning from” a **training set** made up of database tuples and their associated class labels. A tuple,  $\mathbf{X}$ , is represented by an  $n$ -dimensional **attribute vector**,  $\mathbf{X} = (x_1, x_2, \dots, x_n)$ , depicting  $n$  measurements made on the tuple from  $n$  database attributes, respectively,  $A_1, A_2, \dots, A_n$ .<sup>1</sup> Each tuple,  $\mathbf{X}$ , is assumed to belong to a predefined class as determined by another database attribute called the **class label attribute**. The class label attribute is discrete-valued and unordered. It is *categorical* (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as **training tuples** and are randomly sampled from the database under analysis. In the context of classification, data tuples can be referred to as *samples, examples, instances, data points, or objects*.<sup>2</sup>

Because the class label of each training tuple *is provided*, this step belongs to **supervised learning** (i.e., the learning of the classifier is “supervised” in that it is told to which class each training tuple belongs). The scope of supervised learning is larger than classification, and it broadly encompasses learning methods for training a numerical prediction model (e.g., regression, ranking) if the true target values of training tuples are known during the learning step. Supervised learning contrasts with **unsupervised learning** (e.g., **clustering**), in which the true target value (e.g., class label) of each training tuple is not known, and the number or set of classes to be learned may not be known in advance. For example, if we did not have the *loan\_decision* data available for the training set, we could use clustering to try to determine “groups of like tuples” which may correspond to risk groups within the loan application data. Likewise, we could use clustering techniques to find social media posts sharing similar topics without knowing their actual class labels. Clustering is the topic of Chapters 8 and 9. The landscape of the prediction problem (e.g., classification, regression, ranking) has gone beyond supervised vs. unsupervised learning. To name a few, in **semisupervised classification**, it builds a classifier based on a limited number of labeled training tuples (whose true class labels are given during training) and a large number of unlabeled training tuples (whose class labels are unknown during training); in **zero-shot learning**, some class label might appear *after* the classification model has been built. In other words, during the training phase, there are no (i.e., zero) labeled training tuples for such a class label. Both semisupervised learning and zero-shot learning belong to **weakly supervised learning** in that the supervision information for training the model is weaker than the standard supervised learning. For the classification task, this means that the supervision (i.e., the true class labels of training tuples) is known only for a small fraction of the entire training set in semisupervised learning; or is absent for certain class label(s) in zero-shot learning. *Classification with weak supervision* will be introduced in Chapter 7.

The first step of the classification process can also be viewed as the learning of a mapping or function,  $y = f(\mathbf{X})$ , that can predict the associated class label  $y$  of a given tuple  $\mathbf{X}$ . In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae. In our example, the map-

<sup>1</sup> Each attribute represents a “feature” of  $\mathbf{X}$ . Hence, the pattern recognition literature uses the term *feature vector* rather than *attribute vector*. In our discussion, we use these two terms interchangeably. In our notation, any variable representing a vector is typically shown in bold italic font; measurements depicting the vector are shown in italic font (e.g.,  $\mathbf{X} = (x_1, x_2, x_3)$ ).

<sup>2</sup> In the machine learning literature, training tuples are commonly referred to as *training samples*. Throughout this text, we prefer to use the term *tuples* instead of *samples*.

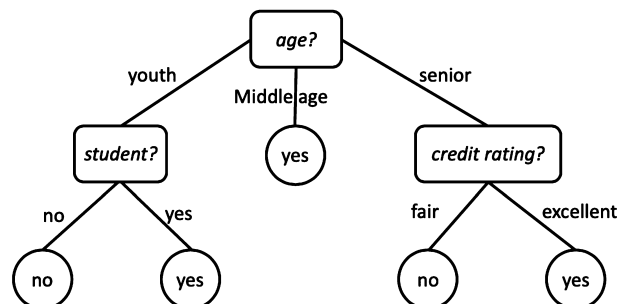
ping is represented as classification rules that identify loan applications as being either safe or risky (Fig. 6.1(a)). The rules can be used to categorize future data tuples, as well as provide deeper insight into the data contents. They also provide a compressed data representation.

“What about classification accuracy?” In the second step (Fig. 6.1(b)), the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier’s accuracy, this estimate would likely be too optimistic, because the classifier tends to **overfit** the data (i.e., during learning it may incorporate some particular anomalies of the training data that do not represent the general data set). Therefore a **test set** is used, made up of **test tuples** and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.

The **accuracy** of a classifier on a given test set is the percentage of test tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier’s class prediction for that tuple. Section 6.6 describes several methods for estimating classifier accuracy. If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known. Such data are also referred to in the machine learning literature as “unknown” or “previously unseen” data. For example, the classification rules learned in Fig. 6.1(a) from the analysis of data from previous loan applications can be used to approve or reject new or future loan applicants.

## 6.2 Decision tree induction

**Decision tree induction** is the learning of decision trees from class-labeled training tuples. A **decision tree** is a flowchart-like tree structure, where each **internal node** (nonleaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label. The topmost node in a tree is the **root** node. A typical decision tree is shown in Fig. 6.2. It represents the concept *buys\_computer*; that is, it predicts whether a customer at an electronics store is



**FIGURE 6.2**

A decision tree for the concept *buys\_computer*, indicating whether a customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys\_computer* = *yes* or *buys\_computer* = *no*).

likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals (or circles). Some decision tree algorithms produce only *binary* trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.

“*How are decision trees used for classification?*” Given a tuple,  $X$ , for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

“*Why are decision tree classifiers so popular?*” The construction of decision tree classifiers does not require any domain knowledge or parameter setting and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

In Section 6.2.1, we describe a basic algorithm for learning decision trees. During tree construction, *attribute selection measures* are used to select the attribute that best partitions the tuples into distinct classes. Popular measures of attribute selection are given in Section 6.2.2. When decision trees are built, many of the branches may reflect noise or outliers in the training data. *Tree pruning* attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data. Tree pruning is described in Section 6.2.3.

## 6.2.1 Decision tree induction

During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as **ID3** (Iterative Dichotomizer). This work expanded on earlier work on *concept learning systems*, described by E. B. Hunt, J. Marin, and P. T. Stone. Quinlan later presented **C4.5** (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared. In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book *Classification and Regression Trees (CART)*, which described the generation of binary decision trees. ID3 and CART were invented independent of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples. These two cornerstone algorithms spawned a flurry of work on decision tree induction.

ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built. A basic decision tree algorithm is summarized in Fig. 6.3. At first glance, the algorithm may appear long, but fear not! It is quite straightforward. The strategy is as follows.

- The algorithm is called with three parameters:  $D$ , *attribute\_list*, and *Attribute\_selection\_method*.  $D$  is a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter *attribute\_list* is a list of attributes describing the tuples. *Attribute\_selection\_method* specifies a heuristic procedure for selecting the attribute that “best” discriminates the given tuples

**Algorithm: Generate\_decision\_tree.** Generate a decision tree from the training tuples of data partition,  $D$ .

**Input:**

- Data partition,  $D$ , which is a set of training tuples and their associated class labels;
- *attribute\_list*, the set of candidate attributes;
- *Attribute\_selection\_method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion consists of a *splitting\_attribute* and, possibly, either a *split-point* or *splitting\_subset*.

**Output:** A decision tree.

**Method:**

- (1) create a node  $N$ ;
- (2) **if** tuples in  $D$  are all of the same class,  $C$ , **then**
- (3)     return  $N$  as a leaf node labeled with the class  $C$ ;
- (4) **if** *attribute\_list* is empty **then**
- (5)     return  $N$  as a leaf node labeled with the majority class in  $D$ ; // majority voting
- (6) apply **Attribute\_selection\_method**( $D$ , *attribute\_list*) to **find** the “best” *splitting\_criterion*;
- (7) label node  $N$  with *splitting\_criterion*;
- (8) **if** *splitting\_attribute* is discrete-valued **and**  
       multiway splits allowed **then** // not restricted to binary trees
- (9)     *attribute\_list*  $\leftarrow$  *attribute\_list* – *splitting\_attribute*; // remove *splitting\_attribute*
- (10) **for each** outcome  $j$  of *splitting\_criterion*  
       // partition the tuples and grow subtrees for each partition
- (11)     let  $D_j$  be the set of data tuples in  $D$  satisfying outcome  $j$ ; // a partition
- (12)     **if**  $D_j$  is empty **then**
- (13)         attach a leaf labeled with the majority class in  $D$  to node  $N$ ;
- (14)     **else** attach the node returned by **Generate\_decision\_tree**( $D_j$ , *attribute\_list*) to node  $N$ ;
- endfor**
- (15) return  $N$ .

**FIGURE 6.3**

Basic algorithm for inducing a decision tree from training tuples.

according to class. This procedure employs an attribute selection measure such as information gain or the Gini impurity. (We will introduce these measures in the next subsection.) Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini impurity, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

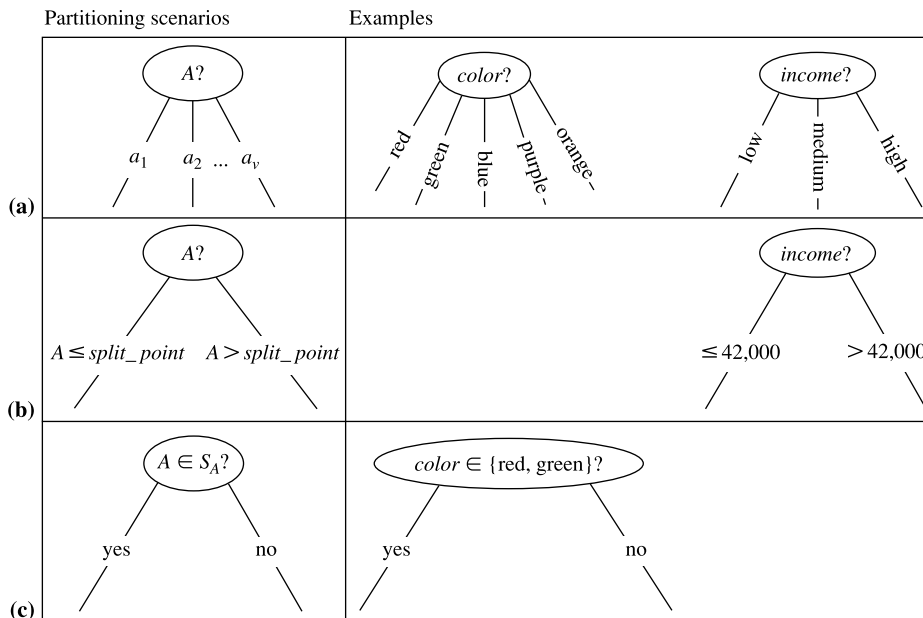
- The tree starts as a single node,  $N$ , representing the training tuples in  $D$  (step 1).<sup>3</sup>
- If the tuples in  $D$  are all of the same class, then node  $N$  becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All terminating conditions are explained at the end of the algorithm.
- Otherwise, the algorithm calls *Attribute\_selection\_method* to determine the **splitting criterion**. The splitting criterion tells us which attribute to test at node  $N$  by determining the “best” way to separate

<sup>3</sup> The partition of class-labeled training tuples at node  $N$  is the set of tuples that follow a path from the root of the tree to node  $N$  when being processed by the tree. This set is sometimes referred to in the literature as the *family* of tuples at node  $N$ . We have referred to this set as the “tuples represented at node  $N$ ,” “the tuples that reach node  $N$ ,” or simply “the tuples at node  $N$ .” Rather than storing the actual tuples at a node, most implementations store pointers to these tuples.

or partition the tuples in  $D$  into individual classes (step 6). The splitting criterion also tells us which branches to grow from node  $N$  with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the **splitting attribute** and may also indicate either a **split-point** or a **splitting subset**. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as “pure” as possible. A partition is **pure** if all the tuples in it belong to the same class. In other words, if we split up the tuples in  $D$  according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

- The node  $N$  is labeled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node  $N$  for each of the outcomes of the splitting criterion. The tuples in  $D$  are partitioned accordingly (steps 10–11). There are three possible scenarios, as illustrated in Fig. 6.4. Let  $A$  be the splitting attribute.  $A$  has  $v$  distinct values,  $\{a_1, a_2, \dots, a_v\}$ , based on the training data.

1. *A is discrete-valued:* In this case, the outcomes of the test at node  $N$  directly correspond to the known values of  $A$ . A branch is created for each known value,  $a_j$ , of  $A$  and labeled with that value (Fig. 6.4(a)). Partition  $D_j$  is the subset of class-labeled tuples in  $D$  having value  $a_j$  of  $A$ . Because all the tuples in a given partition have the same value for  $A$ ,  $A$  does not need to be



**FIGURE 6.4**

This figure shows three possibilities for partitioning tuples based on the splitting criterion, each with examples. Let  $A$  be the splitting attribute. (a) If  $A$  is discrete-valued, then one branch is grown for each known value of  $A$ . (b) If  $A$  is continuous-valued, then two branches are grown, corresponding to  $A \leq \textit{split\_point}$  and  $A > \textit{split\_point}$ . (c) If  $A$  is discrete-valued and a binary tree must be produced, then the test is of the form  $A \in S_A$ , where  $S_A$  is the splitting subset for  $A$ .

considered in any future partitioning of the tuples. Therefore it is removed from *attribute\_list* (steps 8 and 9).

2. *A is continuous-valued*: In this case, the test at node *N* has two possible outcomes, corresponding to the conditions  $A \leq \textit{split\_point}$  and  $A > \textit{split\_point}$ , respectively, where *split\_point* is the split-point returned by *Attribute\_selection\_method* as part of the splitting criterion. (In practice, the split-point, *a*, is often taken as the midpoint of two known adjacent values of *A* and therefore may not actually be a preexisting value of *A* from the training data.) Two branches are grown from *N* and labeled according to the previous outcomes (Fig. 6.4(b)). The tuples are partitioned such that  $D_1$  holds the subset of class-labeled tuples in *D* for which  $A \leq \textit{split\_point}$ , while  $D_2$  holds the rest.
  3. *A is discrete-valued* and a *binary tree* must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node *N* is of the form “ $A \in S_A?$ ,” where  $S_A$  is the splitting subset for *A*, returned by *Attribute\_selection\_method* as part of the splitting criterion. It is a subset of the known values of *A*. If a given tuple has value  $a_j$  of *A*, and if  $a_j \in S_A$ , then the test at node *N* is satisfied. Two branches are grown from *N* (Fig. 6.4(c)). By convention, the left branch out of *N* is labeled *yes* so that  $D_1$  corresponds to the subset of class-labeled tuples in *D* that satisfy the test. The right branch out of *N* is labeled *no* so that  $D_2$  corresponds to the subset of class-labeled tuples from *D* that do not satisfy the test.
- The algorithm uses the same process recursively to form a decision tree for the tuples at each result-*partition*,  $D_j$ , of *D* (step 14).
  - The recursive partitioning stops only when any one of the following terminating conditions is true:
    1. All the tuples in partition *D* (represented at node *N*) belong to the same class (steps 2 and 3).
    2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node *N* into a leaf and labeling it with the most common class in *D*. Alternatively, the class distribution of the node tuples may be stored.
    3. There are no tuples for a given branch, that is, a partition  $D_j$  is empty (step 12). In this case, a leaf is created with the majority class in *D* (step 13).
  - The resulting decision tree is returned (step 15).

The computational complexity of the algorithm given training set *D* is  $O(n \times |D| \times \log(|D|))$ , where *n* is the number of attributes describing the tuples in *D* and  $|D|$  is the number of training tuples in *D*. This means that the computational cost of growing a tree grows at most  $n \times |D| \times \log(|D|)$  with  $|D|$  tuples. The proof is left as an exercise for the reader.

**Incremental** versions of decision tree induction have also been proposed. When given new training data, it restructures the decision tree acquired from learning on previous training data rather than relearning a new tree from scratch.

Differences in decision tree algorithms include how the attributes are selected in creating the tree (Section 6.2.2) and the mechanisms used for pruning (Section 6.2.3).

Decision tree is closely related to another type of tree, called **regression tree**, which is used to predict the continuous output value. A regression tree is very similar to a decision tree in that it also partitions the entire attribute space into multiple subregions, each corresponding to a leaf node. The main difference is as follows. In a regression tree, a leaf node holds a *continuous value* instead of a

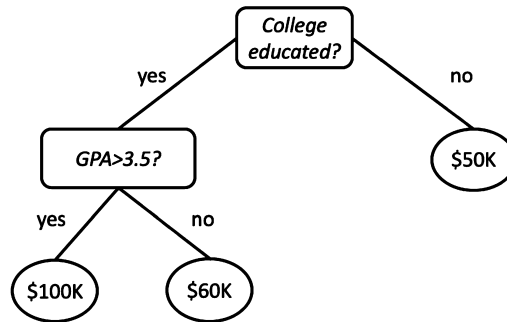


FIGURE 6.5

A regression tree for predicting the average yearly income based on an individual's education. The values of the three leaf nodes are calculated as follows. \$50K is the average yearly income of all training individuals who do not have a college degree; \$60K is the average yearly income of all training individuals who have a college degree with a GPA less than or equal to 3.5; and \$100K is the average yearly income of all training individuals who have a college degree with a GPA higher than 3.5. The leaf node values (\$50K, \$60K, and \$100K) are used to predict the yearly income of any test individual who falls into the corresponding leaf nodes.

categorical value (i.e., class label) in a decision tree. The continuous value of a leaf node is learned during the training phase, which is set as the average output value of all training tuples fallen in the corresponding subregions. CART uses **residual sum of squares** (RSS) as the objective function, which is the sum of the squared difference between the actual and predicted output values of training tuples

$$\text{RSS} = \sum_i (y_i - \hat{y}_i)^2, \quad (6.1)$$

where  $y_i$  is the actual output value of the  $i$ th training tuple, and  $\hat{y}_i$  is the predicted output by the regression tree. Choosing the average output of all training tuples in the corresponding subregion is optimal in that it minimizes the RSS in Eq. (6.1). Each leaf node value is then used to predict the output of a test tuple which falls into it. Fig. 6.5 presents an example of a regression tree for predicting the average yearly income based on an individual's education (e.g., whether or not the individual attended the college, the average GPA at college, etc.).

### 6.2.2 Attribute selection measures

An **attribute selection measure** is a heuristic for selecting the splitting criterion that “best” separates a given data partition,  $D$ , of class-labeled training tuples into individual classes. If we were to split  $D$  into smaller partitions according to the outcomes of the splitting criterion, ideally, each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the “best” splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as **splitting rules** because they determine how the tuples at a given node are to be split.

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure<sup>4</sup> is chosen as the *splitting attribute* for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees, then, respectively, either a *split point* or a *splitting subset* must also be determined as part of the splitting criterion. The tree node created for partition  $D$  is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. This section describes three popular attribute selection measures—*information gain*, *gain ratio*, and *Gini impurity*.

The notation used herein is as follows. Let  $D$ , the data partition, be a training set of class-labeled tuples. Suppose the class label attribute has  $m$  distinct values defining  $m$  distinct classes,  $C_i$  (for  $i = 1, \dots, m$ ). Let  $C_{i,D}$  be the set of tuples of class  $C_i$  in  $D$ . Let  $|D|$  and  $|C_{i,D}|$  denote the number of tuples in  $D$  and  $C_{i,D}$ , respectively.

### Information gain

ID3 uses **information gain** as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or “information content” of messages. Let node  $N$  represent or hold the tuples of partition  $D$ . The attribute with the highest information gain is chosen as the splitting attribute for node  $N$ . This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or “impurity” in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

The expected information needed to classify a tuple in  $D$  is given by

$$\text{Info}(D) = - \sum_{i=1}^m p_i \log_2(p_i), \quad (6.2)$$

where  $p_i$  is the nonzero probability that an arbitrary tuple in  $D$  belongs to class  $C_i$  and is estimated by  $|C_{i,D}|/|D|$ . A log function to the base 2 is used, because the information is encoded in bits.  $\text{Info}(D)$  is just the average amount of information needed to identify the class label of a tuple in  $D$ . Note that, at this point, the information we have is based solely on the proportions of tuples of each class.  $\text{Info}(D)$  is also known as the **entropy** of  $D$ .

Now, suppose we were to partition the tuples in  $D$  on some attribute  $A$  having  $v$  distinct values,  $\{a_1, a_2, \dots, a_v\}$ , as observed from the training data. If  $A$  is discrete-valued, these values correspond directly to the  $v$  outcomes of a test on  $A$ . Attribute  $A$  can be used to split  $D$  into  $v$  partitions or subsets,  $\{D_1, D_2, \dots, D_v\}$ , where  $D_j$  contains those tuples in  $D$  that have outcome  $a_j$  of  $A$ . These partitions would correspond to the branches grown from node  $N$ . Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class).

<sup>4</sup> Depending on the measure, either the highest or lowest score is chosen as the best (i.e., some measures strive to maximize, whereas others strive to minimize).



How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \times Info(D_j). \quad (6.3)$$

The term  $\frac{|D_j|}{|D|}$  acts as the weight of the  $j$ th partition.  $Info_A(D)$  is the expected information required to classify a tuple from  $D$  based on the partitioning by  $A$ . The smaller the expected information (still) required, the greater the purity of the partitions.  $Info_A(D)$  is also known as the conditional entropy of  $D$  (conditioned on the attribute  $A$ ).

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on  $A$ ). That is,

$$Gain(A) = Info(D) - Info_A(D). \quad (6.4)$$

In other words,  $Gain(A)$  tells us how much would be gained by branching on  $A$ . It is the expected reduction in the information requirement caused by knowing the value of  $A$ . The attribute  $A$  with the highest information gain,  $Gain(A)$ , is chosen as the splitting attribute at node  $N$ . This is equivalent to saying that we want to partition on the attribute  $A$  that would do the “best classification,” so that the amount of information still required to finish classifying the tuples is minimal (i.e., minimum  $Info_A(D)$ ).

**Example 6.1. Induction of a decision tree using information gain.** Table 6.1 presents a training set,  $D$ , of class-labeled tuples randomly selected from the customer database of an electronics store. (The data are adapted from Quinlan [Qui86]. In this example, each attribute is discrete-valued. Continuous-valued attributes have been generalized.) The class label attribute, *buys\_computer*, has two distinct

**Table 6.1 Class-labeled training tuples from the customer database of an electronics store.**

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

values (namely,  $\{yes, no\}$ ); therefore, there are two distinct classes (i.e.,  $m = 2$ ). Let class  $C_1$  correspond to *yes* and class  $C_2$  correspond to *no*. There are nine tuples of class *yes* and five tuples of class *no*. A (root) node  $N$  is created for the tuples in  $D$ . To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Eq. (6.2) to compute the expected information needed to classify a tuple in  $D$ :

$$Info(D) = -\frac{9}{14} \log_2 \left( \frac{9}{14} \right) - \frac{5}{14} \log_2 \left( \frac{5}{14} \right) = 0.940 \text{ bits.}$$

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* tuples for each category of *age*. For the *age* category "youth" there are two *yes* tuples and three *no* tuples. For the category "middle\_aged," there are four *yes* tuples and zero *no* tuples. For the category "senior," there are three *yes* tuples and two *no* tuples. Using Eq. (6.3), the expected information needed to classify a tuple in  $D$  if the tuples are partitioned according to *age* is

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times \left( -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} \right) \\ &\quad + \frac{4}{14} \times \left( -\frac{4}{4} \log_2 \frac{4}{4} \right) \\ &\quad + \frac{5}{14} \times \left( -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} \right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

Hence, the gain in information from such partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute  $Gain(income) = 0.029$  bits,  $Gain(student) = 0.151$  bits, and  $Gain(credit\_rating) = 0.048$  bits. Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node  $N$  is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Fig. 6.6. Notice that the tuples falling into the partition for  $age = middle\_aged$  all belong to the same class. Because they all belong to class "yes," a leaf should therefore be created at the end of this branch and labeled "yes." The final decision tree returned by the algorithm was shown earlier in Fig. 6.2.  $\square$

"But how can we compute the information gain of an attribute that is continuous-valued, unlike in the example?" Suppose, instead, that we have an attribute  $A$  that is continuous-valued rather than discrete-valued. (For example, suppose that instead of the discretized version of *age* from the example, we have the raw values for this attribute.) For such a scenario, we must determine the "best" **split-point** for  $A$ , where the split-point is a threshold on  $A$ .

We first sort the values of  $A$  in the increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point. Therefore, given  $v$  values of  $A$ ,  $(v - 1)$  possible splits are evaluated. For example, the midpoint between the values  $a_i$  and  $a_{i+1}$  of  $A$  is

$$\frac{a_i + a_{i+1}}{2}. \tag{6.5}$$

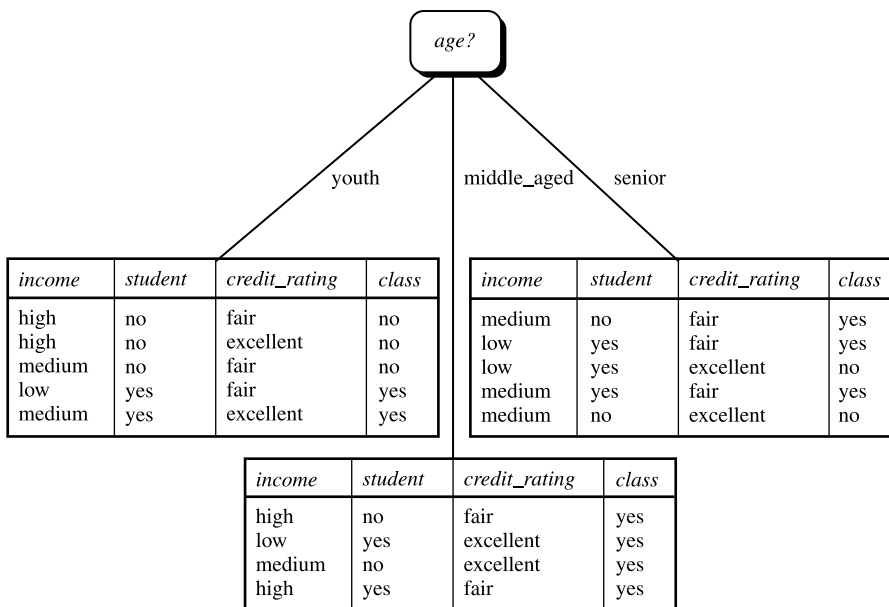


FIGURE 6.6

The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

If the values of  $A$  are sorted in advance, then determining the best split for  $A$  requires only one pass through the values. For each possible split-point for  $A$ , we evaluate  $Info_A(D)$ , where the number of partitions is two, that is,  $v = 2$  (or  $j = 1, 2$ ) in Eq. (6.3). The point with the minimum expected information requirement for  $A$  is selected as the *split\_point* for  $A$ .  $D_1$  is the set of tuples in  $D$  satisfying  $A \leq \textit{split\_point}$ , and  $D_2$  is the set of tuples in  $D$  satisfying  $A > \textit{split\_point}$ .

### Gain ratio

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier, such as *product\_ID*. A split on *product\_ID* would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set  $D$  based on this partitioning would be  $Info_{\textit{product\_ID}}(D) = 0$ . Therefore the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

C4.5, a successor of ID3, uses an extension to information gain known as *gain ratio*, which attempts to overcome this bias. It applies a kind of normalization to information gain using a “split information” value defined analogously with  $Info(D)$  as

$$\textit{SplitInfo}_A(D) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \times \log_2 \left( \frac{|D_j|}{|D|} \right). \quad (6.6)$$

This value represents the potential information generated by splitting the training data set,  $D$ , into  $v$  partitions, corresponding to the  $v$  outcomes of a test on attribute  $A$ . Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in  $D$ . It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}_A(D)}. \quad (6.7)$$

The attribute with the maximum gain ratio is selected as the splitting attribute. Note, however, that as the split information approaches 0, the ratio becomes unstable. A constraint is added to avoid this, whereby the information gain of the test selected must be large—at least as great as the average gain over all tests examined.

**Example 6.2. Computation of gain ratio for the attribute *income*.** A test on *income* splits the data of Table 6.1 into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*, we first use Eq. (6.6) to obtain

$$\begin{aligned} \text{SplitInfo}_{\text{income}}(D) &= -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) \\ &= 1.557. \end{aligned}$$

From Example 6.1, we have  $\text{Gain}(\text{income}) = 0.029$ . Therefore  $\text{GainRatio}(\text{income}) = 0.029/1.557 = 0.019$ .  $\square$

### Gini impurity

The Gini impurity (or Gini in short) is used in CART. Using the notation previously described, the Gini measures the impurity of  $D$ , a data partition or a set of training tuples, as

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2, \quad (6.8)$$

where  $p_i$  is the probability that a tuple in  $D$  belongs to class  $C_i$  and is estimated by  $|C_{i,D}|/|D|$ . The sum is computed over  $m$  classes.

The Gini impurity considers a binary split for each attribute. Let's first consider the case where  $A$  is a discrete-valued attribute having  $v$  distinct values,  $\{a_1, a_2, \dots, a_v\}$ , occurring in  $D$ . To determine the best binary split on  $A$ , we examine all the possible subsets that can be formed using known values of  $A$ . Each subset,  $S_A$ , can be considered as a binary test for attribute  $A$  of the form " $A \in S_A$ ?" Given a tuple, this test is satisfied if the value of  $A$  for the tuple is among the values listed in  $S_A$ . If  $A$  has  $v$  possible values, then there are  $2^v$  possible subsets. For example, if *income* has three possible values, namely  $\{\text{low}, \text{medium}, \text{high}\}$ , then the possible subsets are  $\{\text{low}, \text{medium}, \text{high}\}$ ,  $\{\text{low}, \text{medium}\}$ ,  $\{\text{low}, \text{high}\}$ ,  $\{\text{medium}, \text{high}\}$ ,  $\{\text{low}\}$ ,  $\{\text{medium}\}$ ,  $\{\text{high}\}$ , and  $\{\}$ . We exclude the power set,  $\{\text{low}, \text{medium}, \text{high}\}$ , and the empty set from consideration since, conceptually, they do not represent a split. Therefore there are  $(2^v - 2)/2$  possible ways to form two partitions of the data,  $D$ , based on a binary split on  $A$ .

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on  $A$  partitions  $D$  into  $D_1$  and  $D_2$ , the Gini impurity of  $D$  given that partitioning is

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2). \quad (6.9)$$

For each attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini impurity for that attribute is selected as its splitting subset.

For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point. The point giving the minimum Gini impurity for a given (continuous-valued) attribute is taken as the split-point of that attribute. Recall that for a possible split-point of  $A$ ,  $D_1$  is the set of tuples in  $D$  satisfying  $A \leq \text{split\_point}$ , and  $D_2$  is the set of tuples in  $D$  satisfying  $A > \text{split\_point}$ .

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute  $A$  is

$$\Delta Gini(A) = Gini(D) - Gini_A(D). \quad (6.10)$$

The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini impurity) is selected as the splitting attribute. This attribute and either its splitting subset (for a discrete-valued splitting attribute) or split-point (for a continuous-valued splitting attribute) together form the splitting criterion.

**Example 6.3. Induction of a decision tree using the Gini impurity.** Let  $D$  be the training data shown earlier in Table 6.1, where there are nine tuples belonging to the class  $\text{buys\_computer} = \text{yes}$  and the remaining five tuples belong to the class  $\text{buys\_computer} = \text{no}$ . A (root) node  $N$  is created for the tuples in  $D$ . We first use Eq. (6.8) for the Gini impurity to compute the impurity of  $D$ :

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

To find the splitting criterion for the tuples in  $D$ , we need to compute the Gini impurity for each attribute. Let's start with the attribute  $\text{income}$  and consider each of the possible splitting subsets. Consider the subset  $\{\text{low}, \text{medium}\}$ . This would result in 10 tuples in partition  $D_1$  satisfying the condition " $\text{income} \in \{\text{low}, \text{medium}\}$ ." The remaining four tuples of  $D$  would be assigned to partition  $D_2$ . The Gini impurity value computed based on this partitioning is

$$\begin{aligned} Gini_{\text{income} \in \{\text{low}, \text{medium}\}}(D) &= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\ &= \frac{10}{14} \left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14} \left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\ &= 0.443 \\ &= Gini_{\text{income} \in \{\text{high}\}}(D). \end{aligned}$$

Similarly, the Gini impurity values for splits on the remaining subsets are 0.458 (for the subsets  $\{low, high\}$  and  $\{medium\}$ ) and 0.450 (for the subsets  $\{medium, high\}$  and  $\{low\}$ ). Therefore the best binary split for attribute *income* is on  $\{low, medium\}$  (or  $\{high\}$ ) because it minimizes the Gini impurity. Evaluating *age*, we obtain  $\{youth, senior\}$  (or  $\{middle\_aged\}$ ) as the best split for *age* with a Gini impurity of 0.375; the attributes *student* and *credit\_rating* are both binary, with Gini impurity values of 0.367 and 0.429, respectively.

The attribute *age* and splitting subset  $\{youth, senior\}$  therefore give the minimum Gini impurity overall, with a reduction in impurity of  $0.459 - 0.357 = 0.102$ . The binary split “ $age \in \{youth, senior?\}$ ” results in the maximum reduction in impurity of the tuples in  $D$  and is returned as the splitting criterion. Node  $N$  is labeled with the criterion, two branches are grown from it, and the tuples are partitioned accordingly.  $\square$

“So, what is the relationship between Gini impurity and information gain?” Intuitively, both measures aim to quantify to what extent the impurity will be reduced if we split the current node based on the given attribute. Information gain, rooted in information theory, measures the impurity based on (the change of) the average amount of information needed to identify the class label of a tuple. Gini impurity is related to *mis-classification* in the following way. Based on the class label distribution in the current node, it tells how likely a randomly chosen tuple will be mis-classified if it is assigned to a random class label. Gini impurity is always used for binary split, whereas information gain allows multiway split. In terms of computation, Gini impurity is slightly more efficient than information gain, since the latter involves the logarithm computation. In practice, however, both measures often lead to very similar decision trees.

### **Other attribute selection measures**

This section on attribute selection measures was not intended to be exhaustive. We have shown three measures that are commonly used for building decision trees. These measures are not without their biases. Information gain, as we saw, is biased toward multivalued attributes. Although the gain ratio adjusts for this bias, it tends to prefer unbalanced splits in which one partition is much smaller than the others. The Gini impurity is biased toward multivalued attributes and has difficulty when the number of classes is large. It also tends to favor tests that result in equal-size partitions and purity in both partitions. Although biased, these measures give reasonably good results in practice.

Many other attribute selection measures have been proposed. CHAID, a decision tree algorithm that is popular in marketing, uses an attribute selection measure that is based on the statistical  $\chi^2$  test for independence. Other measures include C-SEP (which performs better than information gain and Gini impurity in certain cases) and G-statistic (an information theoretic measure that is a close approximation to  $\chi^2$  distribution).

For regression tree, it is natural to use RSS (Eq. (6.1)) as the splitting criteria. That is, the best split point for a given attribute is the one that leads the smallest RSS. We choose the attribute with the minimum RSS to split the tree node into two nodes, including left leaf node and right leaf node.

**Example 6.4.** Let us look at an example in Table 6.2 on how to use RSS to find the best split point. Suppose there are five training tuples at a regression tree node, and each training tuple has a true output value  $y_i$  and a continuous attribute  $x_i$  ( $i = 1, \dots, 5$ ). We want to find the best split point for attribute  $x_i$  to split the tree node into two leaf nodes. More specifically, all the tuples whose  $x_i$  is less than or equal to the split point will go to the left leaf node, and the remaining training tuples will go to the right leaf node.

**Table 6.2 Training data for regression.**

attribute $x_i$	1	2	3	4	5
output $y_i$	10	12	8	20	22

Given five training tuples at a regression tree node, each with a true output value  $y_i$  and a continuous attribute  $x_i$  ( $i = 1, \dots, 5$ ). We want to find the best split point for attribute  $x_i$  to split the tree node into two nodes (left node and right node).

**Table 6.3 Using RSS to choose the best split point for data tuples in Table 6.2.**

candidate split point $x_i$	1.5	2.5	3.5	4.5
predicted value of left leaf node $y_l$	10	11	10	12.5
predicted value of right leaf node $y_r$	15.5	16.7	21	22
RSS	131	116.67	10	83

Since  $x_i$  is a continuous attribute with five possible values, there are four candidate split points, including  $x_i = 1.5$ ,  $x_i = 2.5$ ,  $x_i = 3.5$  and  $x_i = 4.5$ . For each candidate split point, we partition the current tree node into two leaf nodes. The average output value  $y_l$  of the training tuples in the left leaf node is used to predict the output of all tuples residing in the left leaf node. Likewise, the average output value  $y_r$  of the training tuples in the right leaf node is used to predict the output of all tuples residing in the right leaf node. For example, if the split point  $x_i = 1.5$ , only the first training tuple goes to the left leaf node, and we have that  $y_l = y_1 = 10$ ; and  $y_r = (y_2 + y_3 + y_4 + y_5)/4 = (12 + 8 + 20 + 22)/4 = 15.5$ . Using the predicted output values for all five training tuples ( $y_l$  or  $y_r$ ), we can use Eq. (6.1) to calculate RSS. Again, if the split point  $x_i = 1.5$ , we have that  $\text{RSS} = \sum_{i=1}^5 (y_i - \hat{y}_i)^2 = (y_1 - y_l)^2 + (y_2 - y_r)^2 + (y_3 - y_r)^2 + (y_4 - y_r)^2 + (y_5 - y_r)^2 = 122.25$ . The computation results for all four possible split points are summarized in Table 6.3. Since  $x_i = 3.5$  has the smallest RSS, it is chosen as the split point.  $\square$

Attribute selection measures based on the **Minimum Description Length (MDL)** principle have the least bias toward multivalued attributes. MDL-based measures use encoding techniques to define the “best” decision tree as the one that requires the fewest number of bits to both (1) encode the tree and (2) encode the exceptions to the tree (i.e., cases that are not correctly classified by the tree). Its main idea is that the simplest solution is preferred. The philosophy underlying the MDL principle is **Occam’s razor**, also known as *law of parsimony*. In data mining and machine learning, Occam’s razor is often translated into a design principle that one should favor a model with a shorter description (hence minimum description length) for the data over a lengthier model, provided that everything else is equal (e.g., both shorter and lengthier models share the same training set errors).

Other attribute selection measures consider **multivariate splits** (i.e., where the partitioning of tuples is based on a *combination* of attributes, rather than on a single attribute). The CART system, for example, can find multivariate splits based on a linear combination of attributes. Multivariate splits are a form of **attribute** (or feature) **construction**, where new attributes are created based on the existing ones. (Attribute construction was also discussed in Chapter 2 as a form of data transformation.) These

other measures mentioned here are beyond the scope of this book. Additional references are given in the bibliographic notes at the end of this chapter (Section 6.10).

“Which attribute selection measure is the best?” All measures have some bias. It has been shown that the time complexity of decision tree induction generally increases exponentially with tree height. Hence, measures that tend to produce shallower trees (e.g., with multiway rather than binary splits, and that favor more balanced splits) may be preferred. However, some studies have found that shallow trees tend to have a large number of leaves and higher error rates. Despite several comparative studies, no single attribute selection measure has been found to be significantly superior to others. Most measures give quite good results.

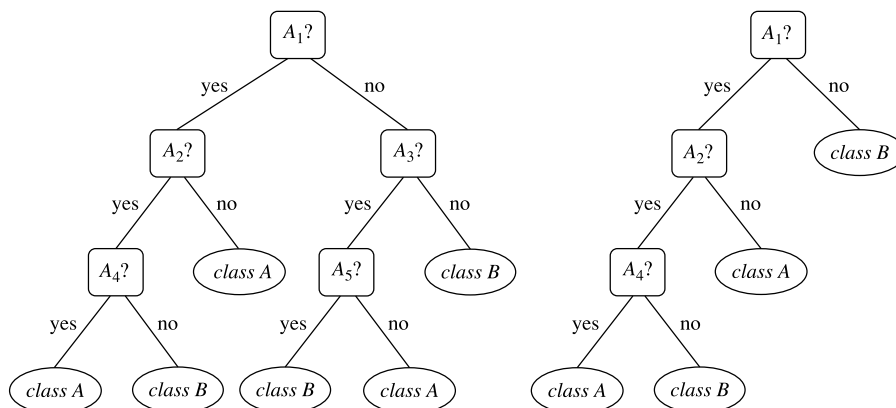
### 6.2.3 Tree pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of *overfitting* the data. Such methods typically use statistical measures to remove the least-reliable branches. An unpruned tree and a pruned version of it are shown in Fig. 6.7. Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

“How does tree pruning work?” There are two common approaches to tree pruning: *prepruning* and *postpruning*.

In the **prepruning** approach, a tree is “pruned” by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class label among the subset tuples or the probability distribution of the class labels of those tuples.

When constructing a tree, measures such as statistical significance, information gain, Gini impurity, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted.



**FIGURE 6.7**

An unpruned decision tree (left) and a pruned version of it (right).



There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

The second and more common approach is **postpruning**, which removes subtrees from a “fully grown” tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class label among the subtree being replaced. For example, notice the subtree at node “ $A_3$ ?” in the unpruned tree of Fig. 6.7. Suppose that the most common class within this subtree is “class B.” In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf “class B.”

The **cost complexity** pruning algorithm used in CART is an example of the postpruning approach. This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the **error rate** is the percentage of tuples misclassified by the tree). It starts from the bottom of the tree. For each internal node,  $N$ , it computes the cost complexity of the subtree at  $N$ , and the cost complexity of the subtree at  $N$  if it were to be pruned (i.e., replaced by a leaf node). The two values are compared. If pruning the subtree at node  $N$  would result in a smaller cost complexity, then the subtree is pruned; otherwise, it is kept.

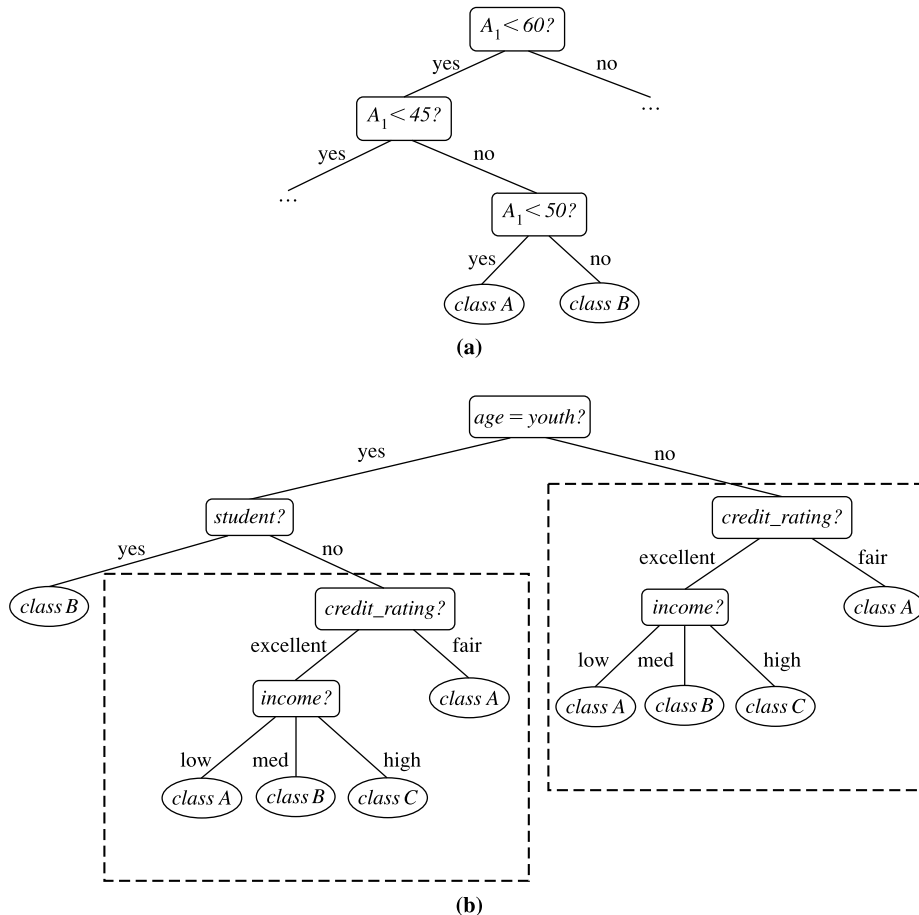
A **pruning set** of class-labeled tuples is used to estimate the cost complexity. This set is independent (1) of the training set used to build the unpruned tree and (2) of any test set used for accuracy estimation. The algorithm generates a set of progressively pruned trees. In general, the smallest decision tree that minimizes the cost complexity is preferred.

C4.5 uses a method called **pessimistic pruning**, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning. Pessimistic pruning, however, does not require the use of a pruning set. Instead, it uses the training set to estimate error rates. Recall that an estimate of accuracy or error based on the training set is overly optimistic and therefore strongly biased. The pessimistic pruning method, therefore, adjusts the error rates obtained from the training set by adding a penalty, so as to counter the bias incurred.

Rather than pruning trees based on estimated error rates, we can prune trees based on the number of bits required to encode them. The “best” pruned tree is the one that minimizes the number of encoding bits. This method adopts the MDL principle, which was briefly introduced in Section 6.2.2. The basic idea is that the simplest solution is preferred. Unlike cost complexity pruning, it does not require an independent set of tuples (i.e., the pruning set).

Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree. No single pruning method has been found to be superior over all others. Although some pruning methods do depend on the availability of additional data for pruning, this is usually not a concern when dealing with large databases.

Although pruned trees tend to be more compact than their unpruned counterparts, they may still be rather large and complex. Decision trees can suffer from *repetition* and *replication* (Fig. 6.8), making them overwhelming to interpret. **Repetition** occurs when an attribute is repeatedly tested along a given branch of the tree (e.g., “age < 60?” followed by “age < 45?” and so on). In **replication**, duplicate subtrees exist within the tree. These situations can impede the accuracy and comprehensibility of a decision tree. The use of multivariate splits (splits based on a combination of attributes) can prevent these problems. Another approach is to use a different form of knowledge representation, such as rules, instead of decision trees. This is described in Chapter 7, which shows how a *rule-based classifier* can be constructed by extracting IF-THEN rules from a decision tree.

**FIGURE 6.8**

An example of (a) subtree **repetition**, where an attribute is repeatedly tested along a given branch of the tree (e.g.,  $age$ ), and (b) subtree **replication**, where duplicate subtrees exist within a tree (e.g., the subtree headed by the node “ $credit\_rating?$ ”).

## 6.3 Bayes classification methods

“What are Bayesian classifiers?” Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.

Bayesian classification is based on Bayes’ theorem, described next. Studies comparing classification algorithms have found a simple Bayesian classifier known as the *naïve Bayesian classifier* to be comparable in performance with decision trees and selected neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called *class-conditional independence*. It is made to simplify the computations involved and, in this sense, is considered “naïve.”

Section 6.3.1 reviews basic probability notation and Bayes’ theorem. In Section 6.3.2, you will learn how to do naïve Bayesian classification.

### 6.3.1 Bayes’ theorem

Bayes’ theorem is named after Thomas Bayes, a nonconformist English clergyman who did early work in probability and decision theory during the 18th century. Let  $X$  be a data tuple. In Bayesian terms,  $X$  is considered “evidence.” As usual, it is described by measurements made on a set of  $n$  attributes. Let  $H$  be some hypothesis such as that the data tuple  $X$  belongs to a specified class  $C$ . For classification problems, we want to determine  $P(H|X)$ , the probability that the hypothesis  $H$  holds given the “evidence” or observed data tuple  $X$ . In other words, we are looking for the probability that tuple  $X$  belongs to class  $C$ , given that we know the attribute description of  $X$ .

$P(H|X)$  is the **posterior probability**, or a *posteriori probability*, of  $H$  conditioned on  $X$ . For example, suppose our world of data tuples is confined to customers described by the attributes *age* and *income*, respectively, and that  $X$  is a 35-year-old customer with an income of \$40,000. Suppose that  $H$  is the hypothesis that our customer will buy a computer. Then  $P(H|X)$  reflects the probability that customer  $X$  will buy a computer given that we know the customer’s age and income.

In contrast,  $P(H)$  is the **prior probability**, or a *priori probability*, of  $H$ . For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. The posterior probability,  $P(H|X)$ , is based on more information (e.g., customer information) than the prior probability,  $P(H)$ , which is independent of  $X$ .

Similarly,  $P(X|H)$  is the conditional probability of  $X$  conditioned on  $H$ . That is, it is the probability that a customer,  $X$ , is 35 years old and earns \$40,000, given that we know the customer will buy a computer. In classification,  $P(X|H)$  is also often referred to as *likelihood*.

$P(X)$  is the prior probability of  $X$ . Using our example, it is the probability that a person from our set of customers is 35 years old and earns \$40,000. In classification,  $P(X)$  is also often referred to as *marginal probability*.

“How are these probabilities estimated?”  $P(H)$ ,  $P(X|H)$ , and  $P(X)$  may be estimated from the given data, as we shall see next. **Bayes’ theorem** is useful in that it provides a way of calculating the posterior probability,  $P(H|X)$ , from  $P(H)$ ,  $P(X|H)$ , and  $P(X)$ . Bayes’ theorem is

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}. \quad (6.11)$$

“What does Bayes classifier look like?” Suppose that there are  $m$  classes,  $C_1, C_2, \dots, C_m$ . Given a tuple,  $X$ , we want to predict which class it belongs to. In Bayes classifier, it first calculates the posterior probabilities for each of the  $m$  classes,  $P(C_i|X)$  ( $i = 1, \dots, m$ ), and then predicts that tuple  $X$  belongs to the class with the highest posterior probability. In the above example, given a customer,  $X$ , of 35 years old and earning \$40,000, we want to predict if the customer will buy a computer. So, in this task, there are two possible classes (buy computer vs. not buy computer). Suppose  $P(\text{buy computer}|X) = 0.8$  and  $P(\text{not buy computer}|X) = 0.2$ . Bayes classifier will predict that the customer  $X$  will buy a computer.

“So, how good is Bayes classifier?” In theory, Bayes classifier is *optimal* in the sense that it has the smallest classification error rate compared to *all* other classifiers. Since Bayes classifier is a probabilistic method, it could make a wrong prediction for any given tuple. In the above example, Bayes classifier predicts the customer will buy a computer. Since  $P(\text{not buy computer}|\mathbf{X}) = 0.2$ , there is 20% chance that the prediction the Bayes classifier makes is incorrect. However, since Bayes classifier always predicts the class with the maximum posterior probability, the probability that its prediction is wrong for a given tuple  $\mathbf{X}$  (which is often called *risk*) is the lowest in comparison to all other classifiers. In our example, the risk for the given customer is 0.2. In other words, there is 20% probability that the prediction by Bayes classifier is wrong. Therefore, the overall classification error of Bayes classifier, which is the expectation (i.e., the weighted average) of the risk of all possible tuples, is the lowest in all possible classifiers. Given its theoretic optimality, Bayes classifier plays a foundational role in the statistical machine learning community. For example, many classifiers (e.g., naïve Bayesian classifier,  $k$ -Nearest-Neighbor classifier, logistic regression, Bayesian network, etc.) can be viewed as approximated Bayes classifiers. Bayes classifier is also useful in that it provides a theoretical justification for other classifiers that do not explicitly use Bayes’ theorem. For example, under certain assumptions, it can be shown that many neural network and curve-fitting algorithms output the *maximum posteriori* hypothesis, as does the Bayes classifier.

“Then, why do not we just use Bayes classifier?” According to Bayes’ theorem (Eq. (6.11)), in order to calculate the posterior probabilities  $P(C_i|\mathbf{X})$  ( $i = 1, \dots, m$ ), we need to know the conditional probabilities  $P(\mathbf{X}|C_i)$  ( $i = 1, \dots, m$ ), the priors  $P(C_i)$  ( $i = 1, \dots, m$ ) and the marginal probability  $P(\mathbf{X})$ . In Bayes classifier, we only need to know which class has the highest posterior probability and for a given tuple  $\mathbf{X}$ , its marginal probability is independent of different classes. In other words, different posterior probabilities  $P(C_i|\mathbf{X})$  ( $i = 1, \dots, m$ ) share the same marginal probability  $P(\mathbf{X})$ . Therefore for the purpose of predicting which class a given tuple belongs to, we only need to estimate the conditional probabilities  $P(\mathbf{X}|C_i)$  ( $i = 1, \dots, m$ ), and the priors  $P(C_i)$  ( $i = 1, \dots, m$ ).<sup>5</sup>

It is relatively easy to estimate the priors  $P(C_i)$  ( $i = 1, \dots, m$ ) from the training data set (the details will be introduced in the next section). On the other hand, it is usually very challenging to directly estimate the conditional probabilities  $P(\mathbf{X}|C_i)$  ( $i = 1, \dots, m$ ). To see this, let us assume there are  $n$  binary attributes  $A_1, A_2, \dots, A_n$ . Then, the  $n$ -dimensional attribute vector  $\mathbf{X}$  has  $2^n$  possible values and we need to estimate the conditional probability of each possible value of the attribute vector with respect to each class label.<sup>6</sup> In other words, the attribute value space is exponential! It is very difficult to estimate such a large number of parameters for the conditional probabilities.<sup>7</sup>

Therefore the main difficulty for Bayes classifier lies in how to efficiently estimate the conditional probabilities, often with some approximation. Many solutions have been developed. One of such efforts, probably the simplest yet quite effective solution, is the naïve Bayesian classifier, which we introduce next.

<sup>5</sup> The marginal probability  $P(\mathbf{X})$  itself can be calculated based on the conditional probabilities  $P(\mathbf{X}|C_i)$  ( $i = 1, \dots, m$ ), and the priors  $P(C_i)$  ( $i = 1, \dots, m$ ) based on the law of total probability, that is,  $P(\mathbf{X}) = \sum_{i=1}^m P(\mathbf{X}|C_i)P(C_i)$ . It is necessary to calculate the marginal probability in some scenarios (e.g., to estimate the risk of Bayes classifier).

<sup>6</sup> The total number of the parameters we need to estimate for conditional probabilities in this case is  $m(2^n - 1)$ . The details are left as an exercise.

<sup>7</sup> In statistics, it means that the estimation results bear high variance, which are not reliable.

### 6.3.2 Naïve Bayesian classification

The **naïve Bayesian** classifier, or **simple Bayesian** classifier, follows the same procedure as Bayes classifier, except the way it estimates the conditional probabilities. In detail, it works as follows:

1. Let  $D$  be a training set of tuples and their associated class labels. As usual, each tuple is represented by an  $n$ -dimensional attribute vector,  $\mathbf{X} = (x_1, x_2, \dots, x_n)$ , depicting  $n$  measurements made on the tuple from  $n$  attributes, respectively,  $A_1, A_2, \dots, A_n$ .
2. Suppose that there are  $m$  classes,  $C_1, C_2, \dots, C_m$ . Given a tuple,  $\mathbf{X}$ , the classifier will predict that  $\mathbf{X}$  belongs to the class having the highest posterior probability, conditioned on  $\mathbf{X}$ . That is, the naïve Bayesian classifier predicts that tuple  $\mathbf{X}$  belongs to the class  $C_i$  if and only if

$$P(C_i|\mathbf{X}) > P(C_j|\mathbf{X}) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize  $P(C_i|\mathbf{X})$ . The class  $C_i$  for which  $P(C_i|\mathbf{X})$  is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Eq. (6.11)),

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}. \quad (6.12)$$

3. As  $P(\mathbf{X})$  is constant for all classes, we only need to find out which class maximizes  $P(\mathbf{X}|C_i)P(C_i)$ . If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is,  $P(C_1) = P(C_2) = \dots = P(C_m)$ , and we would therefore maximize  $P(\mathbf{X}|C_i)$ . Otherwise, we maximize  $P(\mathbf{X}|C_i)P(C_i)$ . Note that the class prior probabilities may be estimated by  $P(C_i) = |C_{i,D}|/|D|$ , where  $|C_{i,D}|$  is the number of training tuples of class  $C_i$  in  $D$ .
4. Given a data set with many attributes, it would be extremely computationally expensive to compute  $P(\mathbf{X}|C_i)$  for the aforementioned reasons. To reduce computation in evaluating  $P(\mathbf{X}|C_i)$ , the naïve assumption of **class-conditional independence** is made. This presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple (i.e., there are no dependence relationships among the attributes, if we know which class the tuple belongs to.). Thus

$$\begin{aligned} P(\mathbf{X}|C_i) &= \prod_{k=1}^n P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \times \dots \times P(x_n|C_i). \end{aligned} \quad (6.13)$$

We can easily estimate the probabilities  $P(x_1|C_i), P(x_2|C_i), \dots, P(x_n|C_i)$  from the training tuples. Recall that here  $x_k$  refers to the value of attribute  $A_k$  for tuple  $\mathbf{X}$ . For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute  $P(\mathbf{X}|C_i)$ , we consider the following:

- a. If  $A_k$  is categorical, then  $P(x_k|C_i)$  is the number of tuples of class  $C_i$  in  $D$  having the value  $x_k$  for  $A_k$ , divided by  $|C_{i,D}|$ , the number of tuples of class  $C_i$  in  $D$ .<sup>8</sup>

<sup>8</sup> In statistics, this is the classic maximum likelihood estimation (MLE) method.

- b. If  $A_k$  is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean  $\mu$  and standard deviation  $\sigma$ , defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (6.14)$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \quad (6.15)$$

These equations may appear daunting, but hold on! We need to compute  $\mu_{C_i}$  and  $\sigma_{C_i}$ , which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute  $A_k$  for training tuples of class  $C_i$ . We then plug these two quantities into Eq. (6.14), together with  $x_k$ , to estimate  $P(x_k|C_i)$ .

For example, let  $X = (35, \$40,000)$ , where  $A_1$  and  $A_2$  are the attributes *age* and *income*, respectively. Let the class label attribute be *buys\_computer*. The associated class label for  $X$  is *yes* (i.e., *buys\_computer* = *yes*). Let's suppose that *age* has not been discretized and therefore exists as a continuous-valued attribute. Suppose that from the training set, we find that customers in  $D$  who buy a computer are  $38 \pm 12$  years of age. In other words, for attribute *age* and this class (i.e., *buys\_computer* = *yes*), we have  $\mu = 38$  years and  $\sigma = 12$ . We can plug these quantities, along with  $x_1 = 35$  for our tuple  $X$ , into Eq. (6.14) to estimate  $P(\text{age} = 35 | \text{buys\_computer} = \text{yes})$ . For a quick review of mean and standard deviation calculations, please see Section 2.2.

5. To predict the class label of  $X$ ,  $P(X|C_i)P(C_i)$  is evaluated for each class  $C_i$ . The classifier predicts that the class label of tuple  $X$  is the class  $C_i$  if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \quad (6.16)$$

In other words, the predicted class label is the class  $C_i$  for which  $P(X|C_i)P(C_i)$  is the maximum.

“How effective is naïve Bayesian classifier?” Notice that the only difference between naïve Bayesian classifier and Bayes classifier is the class-conditional independence assumption. Therefore if such an assumption indeed holds, naïve Bayesian classifier would be optimal with the smallest possible classification error. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class-conditional independence, and the lack of available probability data. Nonetheless, various empirical studies of this classifier in comparison to decision trees and selected neural network classifiers have found it to be comparable in some domains. Another advantage of naïve Bayesian classifier is that it can naturally handle the missing attribute(s).

**Example 6.5. Predicting a class label using naïve Bayesian classification.** We wish to predict the class label of a tuple using naïve Bayesian classification, given the same training data as in Example 6.3 for decision tree induction. The training data were shown earlier in Table 6.1. The data tuples are described by the attributes *age*, *income*, *student*, and *credit\_rating*. The class label attribute, *buys\_computer*, has two distinct values (namely, {*yes*, *no*}). Let  $C_1$  correspond to the class *buys\_computer* = *yes* and  $C_2$  correspond to *buys\_computer* = *no*. The tuple we wish to classify is

$$X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit\_rating} = \text{fair}).$$

We need to find out which class maximizes  $P(\mathbf{X}|C_i)P(C_i)$ , for  $i = 1, 2$ .  $P(C_i)$ , the prior probability of each class, can be computed based on the training tuples:

$$P(\text{buys\_computer} = \text{yes}) = 9/14 = 0.643$$

$$P(\text{buys\_computer} = \text{no}) = 5/14 = 0.357.$$

To compute  $P(\mathbf{X}|C_i)$ , for  $i = 1, 2$ , we compute the following conditional probabilities:

$$P(\text{age} = \text{youth} \mid \text{buys\_computer} = \text{yes}) = 2/9 = 0.222$$

$$P(\text{age} = \text{youth} \mid \text{buys\_computer} = \text{no}) = 3/5 = 0.600$$

$$P(\text{income} = \text{medium} \mid \text{buys\_computer} = \text{yes}) = 4/9 = 0.444$$

$$P(\text{income} = \text{medium} \mid \text{buys\_computer} = \text{no}) = 2/5 = 0.400$$

$$P(\text{student} = \text{yes} \mid \text{buys\_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{student} = \text{yes} \mid \text{buys\_computer} = \text{no}) = 1/5 = 0.200$$

$$P(\text{credit\_rating} = \text{fair} \mid \text{buys\_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{credit\_rating} = \text{fair} \mid \text{buys\_computer} = \text{no}) = 2/5 = 0.400.$$

Using these probabilities, we obtain

$$\begin{aligned} P(\mathbf{X} \mid \text{buys\_computer} = \text{yes}) &= P(\text{age} = \text{youth} \mid \text{buys\_computer} = \text{yes}) \\ &\quad \times P(\text{income} = \text{medium} \mid \text{buys\_computer} = \text{yes}) \\ &\quad \times P(\text{student} = \text{yes} \mid \text{buys\_computer} = \text{yes}) \\ &\quad \times P(\text{credit\_rating} = \text{fair} \mid \text{buys\_computer} = \text{yes}) \\ &= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044. \end{aligned}$$

Similarly,

$$P(\mathbf{X} \mid \text{buys\_computer} = \text{no}) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019.$$

To find the class,  $C_i$ , that maximizes  $P(\mathbf{X}|C_i)P(C_i)$ , we compute

$$P(\mathbf{X} \mid \text{buys\_computer} = \text{yes})P(\text{buys\_computer} = \text{yes}) = 0.044 \times 0.643 = 0.028$$

$$P(\mathbf{X} \mid \text{buys\_computer} = \text{no})P(\text{buys\_computer} = \text{no}) = 0.019 \times 0.357 = 0.007.$$

Therefore the naïve Bayesian classifier predicts  $\text{buys\_computer} = \text{yes}$  for tuple  $\mathbf{X}$ . □

“What if I encounter probability values of zero?” Recall that in Eq. (6.13), we estimate  $P(\mathbf{X}|C_i)$  as the product of the probabilities  $P(x_1|C_i)$ ,  $P(x_2|C_i)$ ,  $\dots$ ,  $P(x_n|C_i)$ , based on the assumption of class-conditional independence. These probabilities can be estimated from the training tuples (step 4). We need to compute  $P(\mathbf{X}|C_i)$  for each class ( $i = 1, 2, \dots, m$ ) to find the class  $C_i$  for which  $P(\mathbf{X}|C_i)P(C_i)$  is the maximum (step 5). Let’s consider this calculation. For each attribute–value pair (i.e.,  $A_k = x_k$ , for  $k = 1, 2, \dots, n$ ) in tuple  $\mathbf{X}$ , we need to count the number of tuples having that attribute–value pair, per class (i.e., per  $C_i$ , for  $i = 1, \dots, m$ ). In Example 6.5, we have two classes ( $m = 2$ ), namely  $\text{buys\_computer} = \text{yes}$  and  $\text{buys\_computer} = \text{no}$ . Therefore, for the attribute–value pair  $\text{student} = \text{yes}$  of  $\mathbf{X}$ , say,

we need two counts—the number of customers who are students and for which *buys\_computer* = *yes* (which contributes to  $P(X|buys\_computer = yes)$ ) and the number of customers who are students and for which *buys\_computer* = *no* (which contributes to  $P(X|buys\_computer = no)$ ).

However, what if, say, there are no training tuples representing students for the class *buys\_computer* = *no*, resulting in  $P(student = yes|buys\_computer = no) = 0$ ? In other words, what happens if we should end up with a probability value of zero for some  $P(x_k|C_i)$ ? Plugging this zero value into Eq. (6.13) would return a zero probability for  $P(X|C_i)$ , even though, without the zero probability, we may have ended up with a high probability, suggesting that  $X$  belonged to class  $C_i$ ! A zero probability cancels the effects of all the other (posteriori) probabilities (on  $C_i$ ) involved in the product.

There is a simple trick to avoid this problem. We can assume that our training database,  $D$ , is so large that adding one to each count that we need would only make a negligible difference in the estimated probability value yet would conveniently avoid the case of probability values of zero. This technique for probability estimation is known as the **Laplacian correction** or **Laplace estimator**, named after Pierre Laplace, a French mathematician who lived from 1749 to 1827.<sup>9</sup> If we have, say,  $q$  counts to which we each add one, then we must remember to add  $q$  to the corresponding denominator used in the probability calculation. We illustrate this technique in Example 6.6.

**Example 6.6. Using the Laplacian correction to avoid computing probability values of zero.** Suppose that for the class *buys\_computer* = *yes* in some training database,  $D$ , containing 1000 tuples, we have 0 tuples with *income* = *low*, 990 tuples with *income* = *medium*, and 10 tuples with *income* = *high*. The probabilities of these events, without the Laplacian correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively. Using the Laplacian correction for the three quantities, we pretend that we have one more tuple for each income-value pair. In this way, we instead obtain the following probabilities (rounded up to three decimal places):

$$\frac{1}{1003} = 0.001, \quad \frac{991}{1003} = 0.988, \quad \text{and} \quad \frac{11}{1003} = 0.011,$$

respectively. The “corrected” probability estimates are close to their “uncorrected” counterparts, yet the zero probability value is avoided.  $\square$

The main idea of naïve Bayesian classifier lies in the class-conditional independence assumption, which significantly simplifies the estimation of the conditional probabilities  $P(X|C_i)$  ( $i = 1, \dots, m$ ). However, this (class-conditional independence assumption) is also one major limitation of naïve Bayesian classifier, since it might not be true for some applications. To address this issue, we need more sophisticated ways to approximate the conditional probabilities, such as Bayesian networks, which will be introduced in the next chapter.

<sup>9</sup> In statistics, this belongs to the Maximum a Posteriori (MAP) method. This can also be viewed as a smoothing technique (i.e., to “smooth” the zero probabilities). In practice, we can also replace 1 by a small integer  $k$ . The intuition is that we have  $k$  (instead of 1) more tuples for each attribute-value pair.



## 6.4 Lazy learners (or learning from your neighbors)

The classification methods discussed so far in this book—decision tree induction and Bayesian classification—are both examples of *eager learners*. **Eager learners**, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples.

Imagine a contrasting lazy approach, in which the learner instead waits until the last minute before doing any model construction to classify a given test tuple. That is, when given a training tuple, a **lazy learner** simply stores it (or does only a little minor processing) and waits until it is given a test tuple. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work when a training tuple is presented and more work when making a classification or numeric prediction. Because lazy learners store the training tuples or “instances,” they are also referred to as **instance-based learners**, even though all learning is essentially based on instances.

When making a classification or numeric prediction, lazy learners can be computationally expensive. They require efficient storage techniques and are well suited to implementation on parallel hardware. They offer little explanation or insight into the data’s structure. Lazy learners, however, naturally support incremental learning. They are able to model complex decision spaces having hyperpolygonal shapes that may not be as easily describable by other learning algorithms (such as hyperrectangular shapes modeled by decision trees). In this section, we look at two examples of lazy learners: *k-nearest-neighbor classifiers* (Section 6.4.1) and *case-based reasoning classifiers* (Section 6.4.2).

### 6.4.1 *k*-nearest-neighbor classifiers

The *k*-nearest-neighbor method was first described in the early 1950s. The method is labor-intensive when given a large training set, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition.

Suppose you want to make a decision on whether or not you should buy a computer. What would you do? One possible way to make such a decision is to find out your friends’ decision on this (whether or not to buy a computer). If most of your close friends buy a computer, maybe you will decide to buy a computer as well. Nearest-neighbor classifiers follow a very similar idea of learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by  $n$  attributes. Each tuple represents a point in an  $n$ -dimensional space. In this way, all the training tuples are stored in an  $n$ -dimensional attribute space. When given an unknown tuple, a ***k*-nearest-neighbor classifier** searches the attribute space for the  $k$  training tuples that are closest to the unknown tuple (i.e., to find your close friends in the above example). These  $k$  training tuples are the  $k$  “nearest neighbors” of the unknown tuple. Then *k-nearest-neighbor classifier* chooses the most common class label among the  $k$  nearest neighbors as the predicted class label of the unknown tuple (i.e., to follow the majority decision of your friends in the above example).

“Closeness” is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, say,  $X_1 = (x_{11}, x_{12}, \dots, x_{1n})$  and  $X_2 = (x_{21}, x_{22}, \dots, x_{2n})$ , is

$$\text{dist}(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}. \quad (6.17)$$

In other words, for each numeric attribute, we take the difference between the corresponding values of that attribute in tuple  $X_1$  and in tuple  $X_2$ , square this difference, and accumulate it. The square root is taken of the total accumulated distance count. Typically, we normalize the values of each attribute before using Eq. (6.17). This helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). Min-max normalization, for example, can be used to transform a value  $v$  of a numeric attribute  $A$  to  $v'$  in the range  $[0, 1]$  by computing

$$v' = \frac{v - \min_A}{\max_A - \min_A}, \quad (6.18)$$

where  $\min_A$  and  $\max_A$  are the minimum and maximum values of attribute  $A$ . Chapter 2 describes other methods for data normalization as a form of data transformation.

For  $k$ -nearest-neighbor classification, the unknown tuple is assigned the most common class label among its  $k$ -nearest neighbors. When  $k = 1$ , the unknown tuple is assigned the class of the training tuple that is closest to it in the attribute space. When  $k > 1$ , we can take a (weighted) majority voting on the class labels among its  $k$ -nearest neighbors. Nearest-neighbor classifiers can also be used for numeric prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the (weighted) average value of the real-valued labels associated with the  $k$ -nearest neighbors of the unknown tuple.

*“But how can distance be computed for attributes that are not numeric, but nominal (or categorical) such as color?”* The previous discussion assumes that the attributes used to describe the tuples are all numeric. For nominal attributes, a simple method is to compare the corresponding value of the attribute in tuple  $X_1$  with that in tuple  $X_2$ . If the two are identical (e.g., tuples  $X_1$  and  $X_2$  both have the color blue), then the difference between the two is taken as 0. If the two are different (e.g., tuple  $X_1$  is blue but tuple  $X_2$  is red), then the difference is considered to be 1. Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a larger difference score is assigned, say, for blue and white than for blue and black).

*“What about missing values?”* In general, if the value of a given attribute  $A$  is missing in tuple  $X_1$  or in tuple  $X_2$ , we assume the maximum possible difference. Suppose that each of the attributes has been mapped to the range  $[0, 1]$ . For nominal attributes, we take the difference value to be 1 if either one or both of the corresponding values of  $A$  are missing. If  $A$  is numeric and missing from both tuples  $X_1$  and  $X_2$ , then the difference is also taken to be 1. If only one value is missing and the other (which we will call  $v'$ ) is present and normalized, then we can take the difference to be either  $|1 - v'|$  or  $|0 - v'|$  (i.e.,  $1 - v'$  or  $v'$ ), whichever is greater.

*“How can I determine a good value for  $k$ , the number of neighbors?”* This can be determined experimentally. Starting with  $k = 1$ , we use a test set to estimate the error rate of the classifier. This process can be repeated each time by incrementing  $k$  to allow for one more neighbor. The  $k$  value that gives the minimum error rate may be selected. In general, the larger the number of training tuples, the larger the value of  $k$  will be (so that classification and numeric prediction decisions can be based on a larger portion of the stored tuples). As the number of training tuples approaches infinity and  $k = 1$ , the error rate can be no worse than twice the Bayes error rate (the latter being the theoretical minimum). In

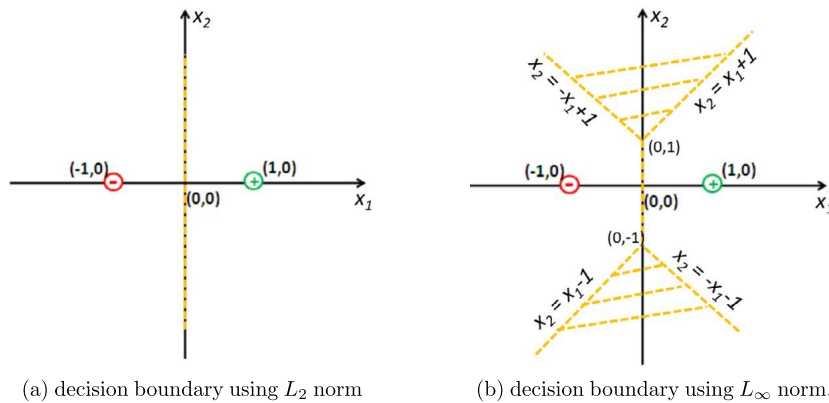


FIGURE 6.9

The impact of distance metrics on 1-nearest-neighbor classifier. Given two training examples, including a positive example at  $(1, 0)$  and a negative example at  $(-1, 0)$ . The decision boundaries of 1-nearest-neighbor classifier using different distance metrics are quite different from each other. Using  $L_2$  norm (on the left), the decision boundary is a vertical line at  $x_2 = 0$ . Using  $L_\infty$  norm (on the right), the decision boundary includes a line segment between  $(0, -1)$  and  $(0, 1)$  and two shaded areas.

other words, 1-nearest-neighbor classifier is asymptotically near-optimal. If  $k$  approaches infinity, the error rate approaches the Bayes error rate.

Nearest-neighbor classifiers use distance-based comparisons that intrinsically assign equal weight to each attribute. They, therefore, can suffer from poor accuracy when given noisy or irrelevant attributes. The method, however, has been modified to incorporate attribute weighting and the pruning of noisy data tuples. The choice of a distance metric can be critical. The Manhattan (city block) distance (Section 2.3), or other distance measurements, may also be used. Fig. 6.9 presents an illustrative example in terms of the impact of distance metrics on the decision boundary of  $k$ -nearest-neighbor classifier.

Nearest-neighbor classifiers can be extremely slow when classifying test tuples. If  $D$  is a training database of  $|D|$  tuples and  $k = 1$ , then  $O(|D|)$  comparisons are required to classify a given test tuple. By presorting and arranging the stored tuples into search trees, the number of comparisons can be reduced to  $O(\log(|D|))$ . Parallel implementation can reduce the running time to a constant, that is,  $O(1)$ , which is independent of  $|D|$ .

Other techniques to speed up classification time include the use of *partial distance* calculations and *editing* the stored tuples. In the **partial distance** method, we compute the distance based on a subset of the  $n$  attributes. If this distance exceeds a threshold, then further computation for the given stored tuple is halted, and the process moves on to the next stored tuple. The **editing** method removes training tuples that are proven useless. This method is also referred to as **pruning** or **condensing** because it reduces the total number of tuples stored. Another technique to speed up nearest-neighbor search is via **locality-sensitive-hashing** (LSH). The key idea is to hash the similar tuples into the same bucket with a high probability via *locality-preserving hash functions*. Then, given a test tuple, we first identify which bucket it belongs to, and then we only search the training tuples in the same bucket to identify its nearest neighbors.

### 6.4.2 Case-based reasoning

**Case-based reasoning** (CBR) classifiers use a database of problem solutions to solve new problems. Unlike  $k$ -nearest-neighbor classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or “cases” for problem solving as complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, where cases describe product-related diagnostic problems. CBR has also been applied to areas, such as engineering and law, where cases are either technical designs or legal rulings in the common law system, respectively. Medical education is another area for CBR, where patient case histories and treatments are used to help diagnose and treat new patients.

When given a new case to classify, a case-based reasoner will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the case-based reasoner will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbors of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The case-based reasoner tries to combine the solutions of the neighboring training cases to propose a solution for the new case. If incompatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary. The case-based reasoner may employ background knowledge and problem-solving strategies to propose a feasible combined solution.

Key challenges in case-based reasoning include finding a good similarity metric (e.g., for matching subgraphs) and suitable methods for combining solutions. Other challenges include the selection of salient features for indexing training cases and the development of efficient indexing techniques. A trade-off between accuracy and efficiency evolves as the number of stored cases becomes very large. As this number increases, the case-based reasoner becomes more intelligent. After a certain point, however, the system’s efficiency will suffer as the time required to search for and process relevant cases increases. As with nearest-neighbor classifiers, one solution is to edit the training database. Cases that are redundant or those that have not proved useful may be discarded for the sake of improved performance. These decisions, however, are not clear-cut, and their automation remains an active area of research.

---

## 6.5 Linear classifiers

So far, we have learned a few classifiers which are capable of generating complex decision boundaries. For example, a decision tree classifier might output a hyperrectangular-shaped decision boundary (Fig. 6.10(a)), and a  $k$ -nearest-neighbor classifier might output a hyperpolygonal-shaped decision boundary (Fig. 6.10(b)). However, what about a simple, linear decision boundary? For the example in Fig. 6.10, intuitively, a linear decision boundary (the straight line in Fig. 6.10(c)) is (almost) as good as decision tree classifiers and  $k$ -nearest-neighbor classifier in separating the positive training tuples from the negative training ones. Yet, such a linear decision boundary might offer additional advantages, such as efficient computation for training the classifier, better generalization performance, and better interpretability.

In this section, we introduce basic techniques to learn such linear classifiers. We will start with **linear regression**, which forms the basis for linear classifiers. Then, we will introduce two linear classi-

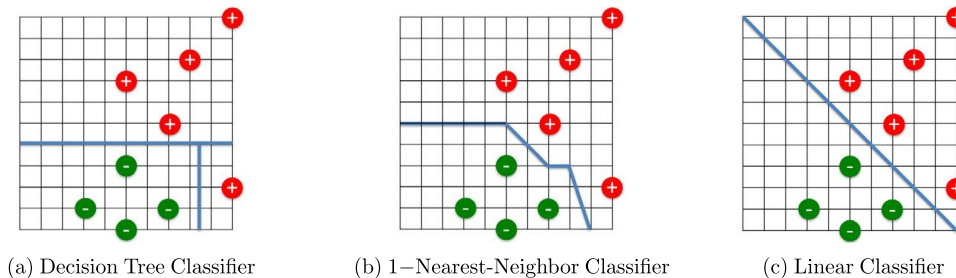


FIGURE 6.10

Decision boundaries by different classifiers. Note that this example is *linearly separable*, meaning that a linear classifier (c) can perfectly separate all the positive training tuples from all the negative training tuples. If the training set is *linearly inseparable*, we could still use a linear classifier, at the expense that some training tuples are on the ‘wrong’ side of the decision boundary. In Chapter 7, we will introduce techniques (e.g., support vector machines) to handle linearly inseparable case.

fiers, including (1) **perceptron**, which is one of the earliest linear classifiers, and (2) **logistic regression** which is one of the most widely used linear classifiers. Additional linear classifiers will be introduced in Chapter 7, such as linear support vector machines.

### 6.5.1 Linear regression

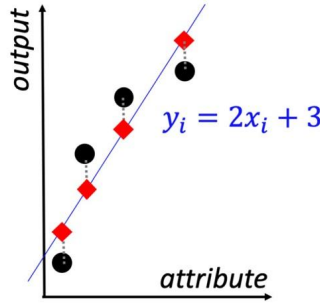
Linear regression is a statistical technique that predicts a continuous value based on one or more independent attributes. For example, we might want to predict the housing price based on the living area or to predict the future income of a student based on which college she attended, in which major and the overall GPA, etc. Since linear regression aims to predict a *continuous* value, it cannot be directly applied to the classification task, where the output is a categorical variable. Nonetheless, the core techniques in linear regression form the basis of linear classifiers. Therefore, let us first briefly introduce linear regression.

Suppose we have  $n$  tuples, each of which is represented by  $p$  attributes  $x_i = (x_{i,1}, \dots, x_{i,p})^T$  and a continuous output value  $y_i$  ( $i = 1, \dots, n$ ). In linear regression, we want to learn a linear function that maps the  $p$  input attributes  $x_i$  to the output variable  $y_i$ , that is,  $\hat{y}_i = w^T x_i + b = \sum_{j=1}^p w_j x_{i,j} + b$ , where  $\hat{y}_i$  is the predicted output value for the  $i$ th tuple,  $w = (w_1, \dots, w_p)^T$  is a  $p$ -dimensional weight vector and  $b$  is the bias scalar. In other words, linear regression assumes that the output value is a linear weighted summation of the  $p$  input attribute values, offset by the bias scalar  $b$ . The entries in the weight vector  $w_j$  ( $j = 1, \dots, p$ ) tell how important the corresponding attribute  $x_{i,j}$  is in predicting the output variable  $\hat{y}_i$ . In the aforementioned examples, a linear regression model would assume that the housing prices are linearly correlated with the living area; the future income of a student can be predicted by a linear weighted combination of the college she attended, the major, and the overall GPA (plus a bias scalar  $b$ ). If we know the weight vector  $w$  and the bias scalar  $b$ , we can make a prediction of the output value based on its  $p$  input attribute values.

“So, how can we determine the weight vector  $w$  and the bias scalar  $b$ ?” Intuitively, we want to learn the “best” weight vector  $w$  and the “best” bias scalar  $b$  from the training data, so that the lin-

Index ( $i$ )	1	2	3	4
Attribute ( $x_i$ )	1	3	5	7
Output ( $y_i$ )	4	10	14	16

(a) Training tuples



(b) Least square regression

FIGURE 6.11

An example of least square regression. (a) Four training tuples. (b) Scatter-plot of the training tuples (black dots) and least square regression model (the blue line). Red diamonds are the predicted output  $\hat{y}_i$  ( $i = 1, 2, 3, 4$ ) and dashed lines indicate the prediction errors ( $|y_i - \hat{y}_i|$ ) of the corresponding training tuples.

ear regression model can make the “best” prediction. That is, the predicted value  $\hat{y}_i = w^T x_i + b$  is as close as possible to the actual observed value  $y_i$  ( $i = 1, \dots, n$ ). One of the most common linear regression methods is called **least square regression**, which aims to minimize the following loss function  $L(w, b) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - (w^T x_i + b))^2$ . Therefore the best weight vector  $w$  and the bias scalar  $b$  are the ones that minimize the loss function  $L(w, b)$ , which measures the sum of the squared difference between the predicted output value  $\hat{y}_i$  and the actual observed value  $y_i$ . For example, if there is only one input attribute (i.e.,  $p = 1$ ), the optimal weight  $w = \frac{\sum_{i=1}^n x_i (y_i - \bar{y})}{\sum_{i=1}^n x_i^2 - \frac{1}{n} (\sum_{i=1}^n x_i)^2}$  and the

optimal bias scalar  $b = \frac{1}{n} \sum_{i=1}^n (y_i - w x_i)$ , where  $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$  is the average observed output value among all  $n$  training tuples.

**Example 6.7.** Let us look at an example of least square regression in Fig. 6.11. There are four training tuples, each represented by a single-dimensional attribute  $x_i$  and an output variable  $y_i$  ( $i = 1, 2, 3, 4$ ). We want to find least square regression model  $y = wx + b$  that predicts the output  $y$  based on the input attribute  $x$ . We use the two equations mentioned above to find the optimal weight  $w$  and the optimal bias scalar  $b$ . We first find the optimal weight  $w$  as follows. The average output of four training tuples is  $\bar{y} = (y_1 + y_2 + y_3 + y_4)/4 = (4 + 10 + 14 + 16)/4 = 11$ . Therefore we have that  $\sum_{i=1}^4 x_i (y_i - \bar{y}) = 1(4 - 11) + 3(10 - 11) + 5(14 - 11) + 7(16 - 11) = 40$ . In the meanwhile, we have that  $\sum_{i=1}^4 x_i^2 = 1^2 + 3^2 + 5^2 + 7^2 = 84$  and  $1/4(\sum_{i=1}^4 x_i)^2 = (1 + 3 + 5 + 7)^2/4 = 64$ . Therefore the optimal weight  $w = \frac{\sum_{i=1}^n x_i (y_i - \bar{y})}{\sum_{i=1}^n x_i^2 - \frac{1}{n} (\sum_{i=1}^n x_i)^2} = \frac{40}{84 - 64} = 2$ . Based on the optimal weight  $w$ , the optimal bias scalar  $b = \frac{\sum_{i=1}^4 (y_i - w x_i)}{4} = \frac{(4 - 2 \times 1) + (10 - 2 \times 3) + (14 - 2 \times 5) + (16 - 2 \times 7)}{4} = 3$ . □

“But, what if there are multiple  $p$  ( $p > 1$ ) attributes?” In this case (which is called **multilinear regression**), let us first change our notation a little bit. We assume there is an additional “dummy” attribute which always takes the value of 1 for any tuple. Let the weight for this dummy attribute be  $w_0$ . Then the overall weight vector  $w = (w_0, w_1, \dots, w_p)$  and the new input attribute vector  $x_i = (1, x_{i,1}, \dots, x_{i,p})$  are both  $(p + 1)$ -dimensional vectors. The multilinear regression model can be re-written as  $\hat{y}_i = w^T x_i = w_0 + w_1 x_{i,1} + \dots + w_p x_{i,p}$ . We use the same loss function as before, that is,  $L(w) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - (w^T x_i))^2$ . It turns out the optimal weight vector  $w$  can be computed as  $w = (XX^T)^{-1}Xy$ , where  $X = [x_1, x_2, \dots, x_n]$  is a  $(p + 1) \times n$  matrix, and  $y = [y_1, \dots, y_n]^T$  is an  $n \times 1$  vector. (How to derive the closed form solutions for single linear regression as well as multilinear regression are left as exercises.)

In least square regression, we measure the “goodness” of the learned regression model by the sum of the squared difference between predicted and actual output values. The squared loss might be sensitive to the outliers in the training set. In *robust regression*, it uses alternative loss functions that are less sensitive to such outliers. For example, the Huber method in robust regression uses the following loss:  $L(w) = \sum_{i=1}^n l_H(y_i - \hat{y}_i)$ , where  $l_H(y_i - \hat{y}_i) = (y_i - \hat{y}_i)^2$  if  $|y_i - \hat{y}_i| < \theta$ ,  $l_H(y_i - \hat{y}_i) = 2\theta|y_i - \hat{y}_i| - \theta^2$  otherwise, and  $\theta > 0$  is a user-specified parameter. Notice that the optimal weight vector  $w$  for multilinear regression involves a matrix inverse (i.e.,  $(XX^T)^{-1}$ ). In case  $p > n$  (i.e., the number of attributes is more than the number of training tuples), such a matrix inverse does not exist. An effective way to address this issue is to introduce a *regularization term* regarding the norm of the weight vector  $w$ . For example, if we use  $l_2$  norm of the weight vector  $w$ , the corresponding regression model is called Ridge regression; if we use  $l_1$  norm of the weight vector  $w$  instead, the corresponding regression model is called Lasso regression which often learns a *sparse* weight vector. This means that some entries of the learned weight vector  $w$  are zeros, which indicates that those attributes are not used in the regression model. In Section 7.1, we will use Lasso regression for feature selection.

## 6.5.2 Perceptron: turning linear regression to classification

“How can we modify a linear regression model to perform classification task?” Suppose we have a binary classification task.<sup>10</sup> The output value  $y_i$  for a given tuple is a binary variable:  $y_i = +1$  indicates the  $i$ th tuple is a positive tuple (e.g., *buy computer*) and  $y_i = 0$  indicates the  $i$ th tuple is a negative one (e.g., *not buy computer*). One way to modify the linear regression model for such a binary classification task is to use the *sign* of the output of the linear regression model as the predicted class label, that is,  $\hat{y}_i = \text{sign}(w^T x_i)$ , where  $\hat{y}_i$  is the predicted class label for  $i$ th tuple,  $\text{sign}(z) = 1$  if  $z > 0$  and  $\text{sign}(z) = 0$  otherwise. Notice that we use the same notation as multilinear regression where we have introduced a “dummy” attribute which always takes the value of 1 for any tuple. Therefore if we know the weight vector  $w$ , we can use it to predict the class label of a given tuple as follows. We compute a linear combination of the attribute values of the given tuple, weighted by the corresponding entries of the weight vector  $w$ . If the resulting value of such a linear combination is positive, we predict that the given tuple is a positive tuple. Otherwise, we predict that it is a negative one.

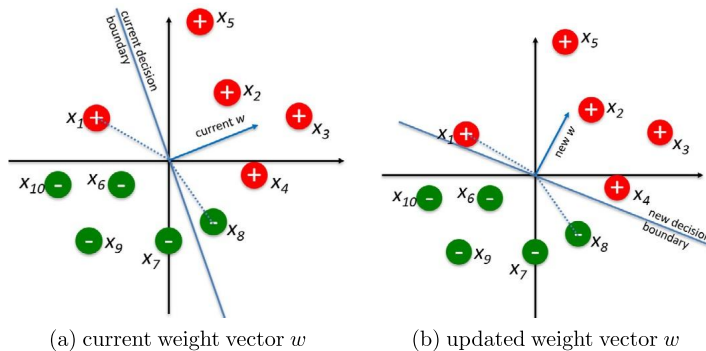
“How can we find the optimal weight vector  $w$  from a set of training tuples?” The classic learning algorithm to train a perceptron is as follows. We start with an initial guess of the weight vector  $w$  (e.g.,

<sup>10</sup> For both perceptron and logistic regression classifiers that we will introduce next, we focus on binary classification task. However, the techniques we introduce can be generalized to handle multiclass classification task for both classifiers.

we can simply set  $w = 0$ ). Then, the learning algorithm will iterate until it converges, or the maximum iteration number or some other preset stopping criteria are met. In each iteration, we do the following for each training tuple  $x_i$ . We try to predict the class label of  $x_i$  using the current weight vector  $w$ , that is,  $\hat{y}_i = \text{sign}(w^T x_i)$ . If the prediction is correct (i.e.,  $\hat{y}_i = y_i$ ), we do nothing about the weight vector. However, if the prediction is incorrect (i.e.,  $\hat{y}_i \neq y_i$ ), we update the current weight vector in one of the following two ways. If  $y_i = +1$  (i.e., the  $i$ th tuple is a positive tuple, but the current classifier predicts it is a negative tuple), we update weight vector as  $w \leftarrow w + \eta x_i$ . If  $y_i = -1$  (i.e., the  $i$ th tuple is a negative tuple, which is wrongly predicted by the current classifier as a positive tuple), we update weight vector as  $w \leftarrow w - \eta x_i$ , where  $\eta > 0$  is the user-specified learning rate. So, the intuition is that in each iteration of the training process, the algorithm will focus on those wrongly predicted training tuples by the current weight vector  $w$ . If the wrongly predicted training tuple  $x_i$  is a positive tuple, we update the weight vector  $w$  by moving it *towards* the attribute vector  $x_i$  of this training tuple (i.e.,  $w \leftarrow w + \eta x_i$ ). On the other hand, if the wrongly predicted training tuple  $x_i$  is a negative tuple, we update the weight vector  $w$  by moving it *away from* the attribute vector  $x_i$  of this training tuple (i.e.,  $w \leftarrow w - \eta x_i$ ).

**Example 6.8.** Let us look at an example in Fig. 6.12 for training a perceptron classifier. In Fig. 6.12, we assume the bias  $w_0 = 0$  for illustration clarity. Fig. 6.12(a) (left) shows the current decision boundary and the weight vector  $w$ , where two training tuples are wrongly classified, including a positive tuple  $x_1$  and a negative tuple  $x_8$ . Therefore only these two tuples are used to update the weight vector in the current iteration, that is,  $w \leftarrow w + \eta x_1 - \eta x_8$ . The updated weight vector  $w$  and the corresponding decision boundary are shown in Fig. 6.12(b) (right), where all training tuples are correctly classified.  $\square$

“How effective is the perceptron learning algorithm?” If the training tuples are linearly separable (e.g., the example in Fig. 6.12), the perceptron algorithm is guaranteed to find a weight vector (i.e., a hyperplane decision boundary) that perfectly separates all the positive training tuples from all the negative training tuples. However, if the training tuples are not linearly separable, this algorithm will fail to converge.



**FIGURE 6.12**

Training a perceptron classifier.



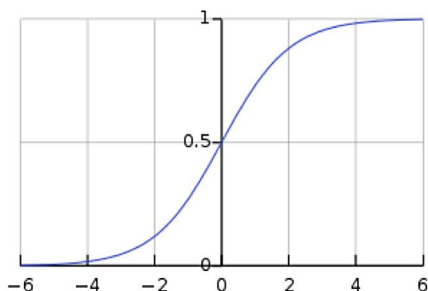
Perceptron, one of the earliest linear classifiers, was first invented back in 1958. It can also be used as a building block (called a “neuron”) in deep neural networks that will be introduced in Chapter 10.

### 6.5.3 Logistic regression

Perceptron that we have just introduced in the previous section is capable of predicting the binary class label of a given tuple. However, can we also tell how confident such a prediction is? Again, let us consider a binary classification task, and we assume that there are two possible class labels, that is,  $y = 1$  for a positive tuple and  $y = 0$  for a negative tuple. Recall that in (naïve) Bayes classifier, we can estimate the posterior probability  $P(y_i = 1|x_i)$ , which can be directly used to indicate how confident the predicted classification result is. For example, if  $P(y_i = 1|x_i)$  is close to 1, the classifier is highly confident that the tuple  $x_i$  is a positive example.

*How can we make a linear classifier not only predict which class label a tuple has, but also tell how confident it is in making such a prediction?* An effective way to this end is via **logistic regression** classifier. Let us first introduce an important function called **sigmoid** function, which is defined as  $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$ . From Fig. 6.13, we can see that the sigmoid function maps a real number in  $(-\infty, +\infty)$  (i.e., the x-axis of Fig. 6.13) to an output value in the range of  $(0, 1)$  (i.e., the y-axis of Fig. 6.13). Therefore if we leverage the sigmoid function to map the output of a linear regression model to a number between 0 and 1, we can interpret the mapping result as the posterior probability of observing a positive class label. This is exactly what logistic regression classifier tries to do!

Formally, we have  $P(\hat{y}_i = 1|x_i, w) = \sigma(w^T x_i) = \frac{1}{1+e^{-w^T x_i}}$ , where  $\hat{y}_i$  is the predicted class label for the tuple with attributes  $x_i$ , and  $w$  is the weight vector. Notice that we have absorbed the bias term  $b$  into the weight vector  $w$  by introducing a dummy attribute to simplify the notation, as we did in the multilinear regression model and in perceptron. Naturally, if  $P(\hat{y}_i = 1|x_i, w) > 0.5$ , the classifier predicts that the tuple  $x_i$  is a positive tuple (i.e.,  $\hat{y}_i = 1$ ), otherwise, it predicts a negative tuple (i.e.,  $\hat{y}_i = 0$ ). This (details are left as an exercise) is equivalent to the following linear classifier: predict  $\hat{y}_i = 1$  (i.e., positive tuple) if  $w^T x_i > 0$ , and predict  $\hat{y}_i = 0$  (i.e., negative tuple) if  $w^T x_i < 0$ . Therefore if we know the weight vector  $w$ , the classification task for a given tuple is quite simple. That is, we



**FIGURE 6.13**

Illustration of sigmoid function. The sigmoid function “squashes” an input from a larger range  $(-\infty, +\infty)$  to a smaller range  $(0, 1)$ . For this reason, sigmoid function is also called *squash function*. In Chapter 10, we will see other types of squash functions, which are called activation functions in the deep learning terminology.

only need to multiply the attribute vector  $x_i$  of the given tuple with the weight vector  $w$ , and then make a prediction based on the sign of  $w^T x_i$ . If  $w^T x_i$  is a positive number, we predict that the given tuple is a positive tuple. Otherwise, we predict that it is a negative tuple.

“How can we determine the optimal weight vector  $w$  from a set of training tuples?” The classic method to train a logistic regression classifier (i.e., to determine the best weight vector  $w$  from the training set) is via *maximum likelihood estimation* (MLE). Again, let us assume there are  $n$  training tuples  $(x_i, y_i)$  ( $i = 1, \dots, n$ ). Since we have a binary classification task, we can view the predicted class label  $\hat{y}_i$  as a Bernoulli random variable, which can only take two possible values, including  $P(\hat{y}_i = 1|x_i, w) = p_i$  and  $P(\hat{y}_i = 0|x_i, w) = 1 - p_i$ , where  $p_i = \sigma(w^T x_i) = \frac{1}{1 + e^{-w^T x_i}}$  is determined by the sigmoid function and it describes the probability of observing a positive outcome for the predicted class label (i.e.,  $\hat{y}_i = 1$ ). Notice that the true class label  $y_i$  for the  $i$ th tuple is a binary variable. Therefore we have that  $P(\hat{y}_i = y_i) = p_i^{y_i} (1 - p_i)^{1 - y_i}$ . The maximum likelihood estimation method aims to solve the following optimization problem, which says that we should choose the best weight vector  $w$  that maximizes the likelihood of the training set. The intuition is that we want to find the optimal model parameter (i.e., the weight vector  $w$ ) so that there is the highest “chance” (i.e., the likelihood or the probability) of observing the entire training set.

$$w^* = \operatorname{argmax}_w L(w) = \prod_{i=1}^n p_i^{y_i} (1 - p_i)^{1 - y_i} = \prod_{i=1}^n \left( \frac{e^{w^T x_i}}{1 + e^{w^T x_i}} \right)^{y_i} \left( \frac{1}{1 + e^{w^T x_i}} \right)^{1 - y_i} \quad (6.19)$$

“But, how can we develop an algorithm to solve this optimization problem to find the optimal weight vector  $w$ ?” First, we notice that the likelihood function  $L(w)$  has many nonnegative terms that are multiplied with each other. In practice, it is often more convenient to work with the logarithm of such a complicated function. Thus we have the following equivalent optimization problem, where  $l(w)$  is called the log likelihood

$$w^* = \operatorname{argmax}_w l(w) = \sum_{i=1}^n y_i x_i^T w - \log(1 + e^{w^T x_i}). \quad (6.20)$$

From the optimization perspective, the good news is that the log likelihood function in Eq. (6.20) is a strictly concave function, and therefore its maximum (the optimal solution) uniquely exists. However, the bad news is that the closed-form solution for the above optimization problem does not exist. In this case, a common strategy is to find the optimal solution  $w^*$  iteratively as follows. In each iteration, we try to improve the current weight vector  $w$  so that the objective function we wish to maximize (the log likelihood function  $l(w)$ ) is improved most. In order to increase the current objective function  $l(w)$  most, it turns out the best direction to update the current estimation of the weight vector  $w$  is to follow its *gradient*. This leads to the following algorithm to learn the optimal weight vector  $w^*$  from the training set. We start with an initial guess of the weight vector  $w$  (e.g., we can simply set  $w = 0$ ). Then, the learning algorithm will iterate until it converges, or the maximum iteration number or some other preset stopping criteria are met. In each iteration, it updates the weight vector  $w$  as follows  $w \leftarrow w + \eta \sum_{i=1}^n (y_i - P(\hat{y}_i = 1|x_i, w)) x_i$ , where  $\eta > 0$  is the user-specified learning rate.

“So, what is the intuition of the above algorithm?” Let us analyze the impact of each training tuple  $(x_i, y_i)$  on updating the estimation of the weight vector  $w$ . We consider two situations depending on whether it is a positive tuple (i.e.,  $y_i = 1$ ) or a negative tuple (i.e.,  $y_i = 0$ ). For the former, the

impact of the given tuple on updating the weight vector  $w$  can be calculated as  $w \leftarrow w + \eta(1 - P(\hat{y}_i = 1|x_i, w))x_i$ . The intuition is that we want update the current weight vector  $w$  towards the direction of the attribute vector  $x_i$  of this positive tuple. For the latter case (i.e.,  $y_i = 0$ ), the impact of the given tuple on updating the weight vector  $w$  can be calculated as  $w \leftarrow w - \eta P(\hat{y}_i = 1|x_i, w)x_i$ . The intuition is that we want update the current weight vector  $w$  away from the direction of the attribute vector  $x_i$  of this negative tuple. From this perspective, the learning algorithm for training a logistic regression classifier bears some similarities to the perceptron algorithm. That is, both algorithms try to update the current weight vector  $w$  so that is (1) more aligned with the attribute vectors of positive tuples and (2) more mis-aligned with (i.e., towards the opposite direction of) the attribute vectors of negative tuples.

However, the two algorithms (perceptron vs. logistic regression) differ regarding to what extent the algorithms update the weight vector  $w$ . In perceptron, it uses a *fixed* learning rate  $\eta$  for all wrongly predicted tuples by the current weight vector  $w$ . On the other hand, in logistic regression, it depends on the learning rate  $\eta$  as well as  $P(\hat{y}_i = 1|x_i, w)$  (i.e., the probability that the given tuple belongs to the positive class based on the current weight vector  $w$ ). This makes the logistic regression algorithm *adaptive* in the following sense. For example, if  $P(\hat{y}_i = 1|x_i, w)$  is high for a positive tuple, it means that the prediction by the current weight vector  $w$  for this positive tuple is not only correct (i.e.,  $P(\hat{y}_i = 1|x_i, w) > 0.5$ ), but also quite confident (i.e.,  $P(\hat{y}_i = 1|x_i, w)$  is close to 1). Then, the impact of this positive tuple (i.e.,  $\eta(1 - P(\hat{y}_i = 1|x_i, w))$ ) on updating the weight vector is relatively small. On the other hand, if  $P(\hat{y}_i = 1|x_i, w)$  is high for a negative tuple, it means that the prediction by the current weight vector  $w$  for this negative tuple is either wrong (i.e.,  $P(\hat{y}_i = 1|x_i, w) > 0.5$ ), or correct but with low confidence (i.e.,  $P(\hat{y}_i = 1|x_i, w)$  is barely below 0.5). Then, the impact of this negative tuple (i.e.,  $\eta P(\hat{y}_i = 1|x_i, w)$ ) on updating the weight vector will be relatively large. In other words, the logistic regression learning algorithm pays more attention to those “hard” training tuples, which are either wrongly predicted or correctly predicted with a low confidence by the current weight vector  $w$ . Recall that for the example in Fig. 6.12(a), perceptron only uses  $x_1$  and  $x_8$  to update the current weight vector  $w$  since these two tuples are wrongly classified by the current  $w$ . In contrast, logistic regression uses *all* training tuples to update the weight vector  $w$ . Among them,  $x_1$  and  $x_8$  have the highest impact on updating  $w$  since they are both wrongly classified by the current classifier;  $x_2, x_3, x_5, x_9$  and  $x_{10}$  have the least impact since they are all correctly classified by the current weight vector  $w$  with a high confidence;  $x_4, x_6$  and  $x_7$  have the moderate impact since they are correctly classified but with a relatively low confidence.

“How good is the logistic regression algorithm? What are the potential limitations and how to mitigate?” Since the log likelihood function  $l(w)$  is a concave function, the algorithm for training a logistic regression classifier described above is guaranteed to converge to its optimal solution. However, if the training set is linearly separable, the algorithm might converge to a weight vector  $w$  with an infinitely large norm. (See an illustrative example in Fig. 6.14.) A “large” weight vector  $w$  could make the trained classifier prone to the noise of certain attributes of a given tuple. This will, in turn, lead to a poor generalization performance of the learned logistic regression classifier. In other words, the learned logistic regression classifier *overfits* the training set. An effective way to mitigate the overfitting is to introduce a regularization term  $\|w\|_2^2$  into the objective function  $l(w)$  to prevent the learned weight vector from

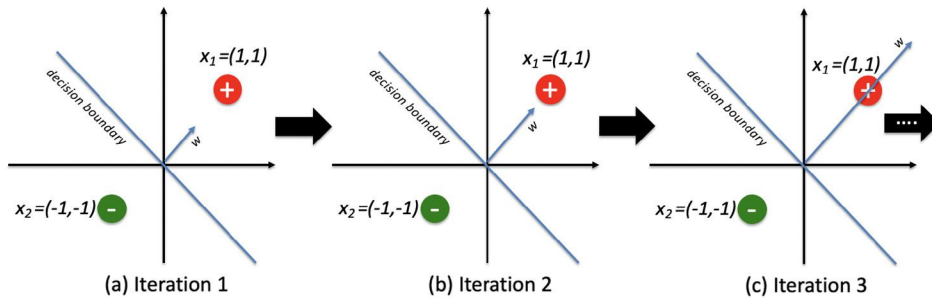


FIGURE 6.14

Illustration of the infinitely large weight vector of logistic regression in linearly separable case. There are two training tuples in 2d space, with one positive training tuple  $x_1 = (1, 1)$  and one negative training tuple  $x_2 = (-1, -1)$ . For simplicity, we let the bias scalar  $b = 0$  and the learning rate  $\eta = 1$ . Suppose that at iteration 1, the weight vector  $w = (1, 1)$ . Then, logistic regression algorithm introduced above will update the weight vector as  $w_{\text{new}} = w_{\text{old}} + (1 - P(\hat{y}_1 = 1|x_1, w_{\text{old}}))x_1 - P(\hat{y}_2 = 1|x_2, w_{\text{old}})x_2 = (0.5, 0.5) + a(1, 1) = (1 + 2a)w_{\text{old}}$ , where  $a = 1 - P(\hat{y}_1 = 1|x_1, w_{\text{old}}) + P(\hat{y}_2 = 1|x_2, w_{\text{old}}) > 0$ . As such, the new weight vector  $w_{\text{new}}$  shares the same direction as the old  $w_{\text{old}}$ . Therefore, the decision boundary remains the same, but new weight vector  $w_{\text{new}}$  grows in the magnitude by a factor of  $(1 + 2a)$ . This trend will continue as the logistic regression algorithm progresses, leading to a weight vector with an infinitely large magnitude.

becoming “too large.”<sup>11</sup> The second potential limitation is the *independence* assumption behind logistic regression. Recall that when we calculate the likelihood  $L(w)$  of the training set, we simply multiply the likelihood of each training tuple together (Eq. (6.19)). This means that we have implicitly assumed that different training tuples are independent of each other. However, this assumption might be violated in some applications (e.g., users on a social network are interconnected with each other). The graph-based classification might provide a natural remedy for this issue. The third potential limitation lies in the computational challenge. Notice that in the updating rule  $w \leftarrow w + \eta \sum_{i=1}^n (y_i - P(\hat{y}_i = 1|x_i, w))x_i$  described above, we need to calculate the gradients  $(y_i - P(\hat{y}_i = 1|x_i, w))$  for *all* training tuples and then sum them up to update the weight vector  $w$ . If there are millions of training tuples, it is computationally very expensive to perform such computation. An efficient way to address this issue is to use *stochastic gradient descent* method to train a logistic regression classifier. That is, at each iteration, we will randomly sample a small subset of training tuples (this is often referred to as a *minibatch*) and *only* use the sampled tuples (instead of all training tuples) to update the weight vector. It is worth pointing out that the stochastic gradient descent is extensively used in many other data mining algorithms, such as deep learning methods, which will be introduced in Chapter 10.

<sup>11</sup> From the statistical parameter estimation perspective, we are switching from the maximum likelihood estimation (MLE) to maximum a posterior estimation (MAP). Adding a regularization term  $\|w\|_2^2$  into  $l(w)$  is equivalent to imposing a Gaussian prior with the mean vector at the origin for the weight vector  $w$ .

## 6.6 Model evaluation and selection

Now that you may have built a classification model, there may be many questions going through your mind. For example, suppose you have used data from previous sales to build a classifier to predict customer purchasing behavior. You would like an estimate of how accurately the classifier can predict the purchasing behavior of future customers, that is, future customer data on which the classifier has not been trained. You may even have tried different methods to build more than one classifier and now wish to compare their accuracy. But what is accuracy? How can we estimate it? Are some measures of a classifier's accuracy more appropriate than others? How can we obtain a *reliable* accuracy estimate? These questions are addressed in this section.

Section 6.6.1 describes various evaluation metrics for the predictive accuracy of a classifier. Based on randomly sampled partitions of the given data, holdout and random subsampling (Section 6.6.2), cross-validation (Section 6.6.3), and bootstrap methods (Section 6.6.4) are common techniques for assessing accuracy. What if we have more than one classifier and want to choose the “best” one? This is referred to as **model selection** (i.e., choosing one classifier over another). The last two sections address this issue. Section 6.6.5 discusses how to use tests of statistical significance to assess whether the difference in accuracy between two classifiers is due to chance. Section 6.6.6 presents how to compare classifiers based on cost–benefit and receiver operating characteristic (ROC) curves.

### 6.6.1 Metrics for evaluating classifier performance

This section presents measures for assessing how good or how “accurate” your classifier is at predicting the class label of tuples. We will consider the case where the class tuples are more or less evenly distributed, as well as the case where classes are unbalanced (e.g., where an important class of interest is rare such as in medical tests). The classifier evaluation measures presented in this section are summarized in Fig. 6.15. They include accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision,  $F_1$ , and  $F_\beta$ . Note that although accuracy is a specific measure, the word “accuracy” is also used as a general term to refer to a classifier's predictive abilities.

Using training data to derive a classifier and then estimate the accuracy of the learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. (We will say more on this in a moment!) Instead, it is better to measure the classifier's accuracy on a *test set* consisting of class-labeled tuples that were not used to train the model.

Before we discuss the various measures, we need to become comfortable with some terminology. Recall that we can talk in terms of **positive tuples** (tuples of the main class of interest) and **negative tuples** (all other tuples).<sup>12</sup> Given two classes, for example, the positive tuples may be *buys\_computer = yes* while the negative tuples are *buys\_computer = no*. Suppose we use our classifier on a test set of labeled tuples.  $P$  is the number of positive tuples, and  $N$  is the number of negative tuples. For each tuple, we compare the classifier's class label prediction with the tuple's known class label.

There are four additional terms we need to know that are the “building blocks” used in computing various evaluation measures. Understanding them will make it easy to grasp the meaning of the various measures.

---

<sup>12</sup> In the machine learning and pattern recognition literature, these are referred to as *positive samples* and *negative samples*, respectively.

Measure	Formula
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$
$F$ , $F_1$ , $F$ -score, harmonic mean of precision and recall	$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
$F_\beta$ , where $\beta$ is a nonnegative real number	$\frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$

FIGURE 6.15

Evaluation measures. Note that some measures are known by more than one name.  $TP$ ,  $TN$ ,  $FP$ ,  $FN$ ,  $P$ ,  $N$  refer to the number of true positive, true negative, false positive, false negative, positive, and negative samples, respectively (see text).

		Predicted class		
		yes	no	Total
Actual class	yes	$TP$	$FN$	$P$
	no	$FP$	$TN$	$N$
Total		$P'$	$N'$	$P + N$

FIGURE 6.16

Confusion matrix, shown with totals for positive and negative tuples.

- **True positives** ( $TP$ ): These refer to the positive tuples that were correctly labeled by the classifier. Let  $TP$  be the number of true positives.
- **True negatives** ( $TN$ ): These are the negative tuples that were correctly labeled by the classifier. Let  $TN$  be the number of true negatives.
- **False positives** ( $FP$ ): These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class `buys_computer = no` for which the classifier predicted `buys_computer = yes`). Let  $FP$  be the number of false positives.
- **False negatives** ( $FN$ ): These are the positive tuples that were mislabeled as negative (e.g., tuples of class `buys_computer = yes` for which the classifier predicted `buys_computer = no`). Let  $FN$  be the number of false negatives.

These terms are summarized in the **confusion matrix** of Fig. 6.16.

A confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes.  $TP$  and  $TN$  tell us when the classifier is getting things right, whereas  $FP$  and  $FN$  tell us when the classifier is getting things wrong (i.e., mislabeling). Given  $m$  classes (where  $m \geq 2$ ), a **confusion matrix** is a table of at least size  $m$  by  $m$ . An entry,  $CM_{i,j}$  at the  $i$ th row and the  $j$ th column indicates the number of tuples of class  $i$  that were labeled by the classifier as class  $j$ . For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry  $CM_{1,1}$  to entry  $CM_{m,m}$ , with the rest of the entries being zero or close to zero. That is, ideally,  $FP$  and  $FN$  are around zero.

Classes	<i>buys_computer = yes</i>	<i>buys_computer = no</i>	Total	Recognition (%)
<i>buys_computer = yes</i>	6954	46	7000	99.34
<i>buys_computer = no</i>	412	2588	3000	86.27
Total	7366	2634	10,000	95.42

FIGURE 6.17

Confusion matrix for the classes *buys\_computer = yes* and *buys\_computer = no*, where an entry in row  $i$  and column  $j$  shows the number of tuples of class  $i$  that were labeled by the classifier as class  $j$ . Ideally, the nondiagonal entries should be zero or close to zero.

The table may have additional rows or columns to provide totals. For example, in the confusion matrix of Fig. 6.16,  $P$  and  $N$  are shown. In addition,  $P'$  is the number of tuples that were labeled as positive ( $TP + FP$ ), and  $N'$  is the number of tuples that were labeled as negative ( $TN + FN$ ). The total number of tuples is  $TP + TN + FP + FN$ , or  $P + N$ , or  $P' + N'$ . Note that although the confusion matrix shown is for a binary classification problem, confusion matrices can be easily drawn for multiple classes in a similar manner.

Now let's look at the evaluation measures, starting with accuracy. The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. That is,

$$accuracy = \frac{TP + TN}{P + N}. \quad (6.21)$$

In the pattern recognition literature, this is also referred to as the overall **recognition rate** of the classifier; that is, it reflects how well the classifier recognizes tuples of the various classes. An example of a confusion matrix for the two classes *buys\_computer = yes* (positive) and *buys\_computer = no* (negative) is given in Fig. 6.17. Totals are shown, as well as the recognition rates per class and overall. By glancing at a confusion matrix, it is easy to see if the corresponding classifier is confusing two classes.

For example, we see that it mislabeled 412 “no” tuples as “yes.” Accuracy is most effective when the class distribution is relatively balanced.

We can also speak of the **error rate** or **misclassification rate** of a classifier,  $M$ , which is simply  $1 - accuracy(M)$ , where  $accuracy(M)$  is the accuracy of  $M$ . This also can be computed as

$$error\ rate = \frac{FP + FN}{P + N}. \quad (6.22)$$

If we were to use the training set (instead of a test set) to estimate the error rate of a model, this quantity is known as the **resubstitution error**.<sup>13</sup> This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.

We now consider the **class imbalance problem**, where the main class of interest is rare. That is, the data set distribution reflects a significant majority of the negative class and a minority positive class. For example, in fraud detection applications, the class of interest (or positive class) is “*fraud*” which occurs much less frequently than the negative “*nonfraudulent*” class. In medical data, there may

<sup>13</sup> In machine learning literature, it is often referred to as the training error.

Classes	yes	no	Total	Recognition (%)
yes	90	210	300	30.00
no	140	9560	9700	98.56
Total	230	9770	10,000	96.40

FIGURE 6.18

Confusion matrix for the classes *cancer* = *yes* and *cancer* = *no*.

be a rare class, such as “*cancer*.” Suppose that you have trained a classifier to classify medical data tuples, where the class label attribute is “*cancer*” and the possible class values are “*yes*” and “*no*.” An accuracy rate of, say, 97% may make the classifier seem quite accurate, but what if only, say, 3% of the training tuples are actually cancer? Clearly, an accuracy rate of 97% may not be acceptable—the classifier could be correctly labeling only the noncancer tuples, for instance, and misclassifying all the cancer tuples. Instead, we need other measures, which assess how well the classifier can recognize the positive tuples (*cancer* = *yes*) and how well it can recognize the negative tuples (*cancer* = *no*).

The **sensitivity** and **specificity** measures can be used, respectively, for this purpose. Sensitivity is also referred to as the *true positive (recognition) rate* (i.e., the proportion of positive tuples that are correctly identified), whereas specificity is the *true negative rate* (i.e., the proportion of negative tuples that are correctly identified). These measures are defined as

$$\text{sensitivity} = \frac{TP}{P} \quad (6.23)$$

$$\text{specificity} = \frac{TN}{N}. \quad (6.24)$$

It can be shown that accuracy is a function of sensitivity and specificity:

$$\text{accuracy} = \text{sensitivity} \frac{P}{(P + N)} + \text{specificity} \frac{N}{(P + N)}. \quad (6.25)$$

**Example 6.9. Sensitivity and specificity.** Fig. 6.18 shows a confusion matrix for medical data where the class values are *yes* and *no* for a class label attribute, *cancer*. The sensitivity of the classifier is  $\frac{90}{300} = 30.00\%$ . The specificity is  $\frac{9560}{9700} = 98.56\%$ . The classifier’s overall accuracy is  $\frac{9650}{10,000} = 96.50\%$ . Thus we note that although the classifier has a high accuracy, it’s ability to correctly label the positive (rare) class is poor given its low sensitivity. It has high specificity, meaning that it can accurately recognize negative tuples. Techniques for handling class-imbalanced data are given in Section 6.7.5. □

The *precision* and *recall* measures are also widely used in classification. **Precision** can be thought of as a measure of *exactness* (i.e., what percentage of tuples labeled as positive are actually such), whereas **recall** is a measure of *completeness* (what percentage of positive tuples are labeled as such). If recall seems familiar, that’s because it is the same as sensitivity (or the *true positive rate*). These measures can be computed as

$$\text{precision} = \frac{TP}{TP + FP} \quad (6.26)$$



$$\text{recall} = \frac{TP}{TP + FN} = \frac{TP}{P}. \quad (6.27)$$

**Example 6.10. Precision and recall.** The precision of the classifier in Fig. 6.18 for the *yes* class is  $\frac{90}{230} = 39.13\%$ . The recall is  $\frac{90}{300} = 30.00\%$ , which is the same calculation for sensitivity in Example 6.9.  $\square$

A perfect precision score of 1.0 for a class  $C$  means that every tuple that the classifier labeled as belonging to class  $C$  does indeed belong to class  $C$ . However, it does not tell us anything about the number of class  $C$  tuples that the classifier mislabeled. A perfect recall score of 1.0 for  $C$  means that every item from class  $C$  was labeled as such, but it does not tell us how many other tuples were incorrectly labeled as belonging to class  $C$ . There tends to be an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. For example, our medical classifier may achieve high precision by labeling all cancer tuples that present a certain way as *cancer* but may have low recall if it mislabels many other instances of *cancer* tuples. Precision and recall scores are typically used together, where precision values are compared for a fixed value of recall, or vice versa. For example, we may compare precision values at a recall value of, say, 0.75.

An alternative way to use precision and recall is to combine them into a single measure. This is the approach of the  $F$  measure (also known as the  $F_1$  score or  $F$ -score) and the  $F_\beta$  measure. They are defined as

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (6.28)$$

$$F_\beta = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}, \quad (6.29)$$

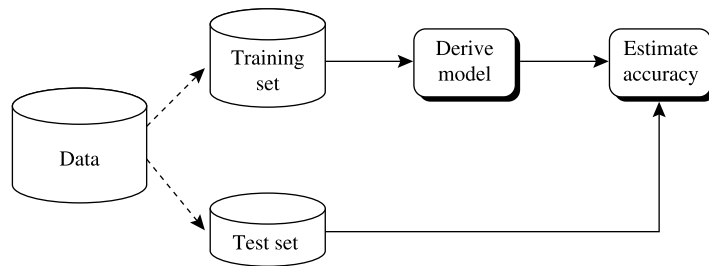
where  $\beta$  is a nonnegative real number. The  $F$  measure is the *harmonic mean* of precision and recall (the proof of which is left as an exercise). It gives equal weights to precision and recall. The  $F_\beta$  measure is a weighted measure of precision and recall. It assigns  $\beta$  times as much weight to recall as to precision. Commonly used  $F_\beta$  measures are  $F_2$  (which weights recall twice as much as precision) and  $F_{0.5}$  (which weights precision twice as much as recall).

“Are there other cases where accuracy may not be appropriate?” In classification problems, it is commonly assumed that all tuples are uniquely classifiable, that is, each training tuple can belong to only one class. Yet, owing to the wide diversity of data in large databases, it is not always reasonable to assume that all tuples are uniquely classifiable. Rather, it is more probable to assume that each tuple may belong to more than one class. How then can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, because it does not take into account the possibility of tuples belonging to more than one class.

Rather than returning a class label, it is useful to return a class probability distribution. Accuracy measures may then use a **second guess** heuristic, whereby a class prediction is judged as correct if it agrees with the first or second most probable class. Although this does take into consideration, to some degree, the nonunique classification of tuples, it is not a complete solution.

In addition to accuracy-based measures, classifiers can also be compared with respect to the following additional aspects:

- **Speed:** This refers to the computational cost involved in generating and using the given classifier.

**FIGURE 6.19**

Estimating accuracy with the holdout method.

- **Robustness:** This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.
- **Scalability:** This refers to the ability to construct the classifier efficiently given large amounts of data. Scalability is typically assessed with a series of data sets of increasing size.
- **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability could be subjective and therefore more difficult to assess. Decision trees and classification rules can be easy to interpret, yet their interpretability may diminish the more they become complex. We will introduce some basic techniques to improve the interpretability of classification models in Chapter 7.

In summary, we have presented several evaluation measures. The accuracy measure works best when the data classes are fairly evenly distributed. Other measures, such as sensitivity (or recall), specificity, precision,  $F$ , and  $F_\beta$ , are better suited to the class imbalance problem, where the main class of interest is rare. The remaining subsections focus on obtaining reliable classifier accuracy estimates.

### 6.6.2 Holdout method and random subsampling

The **holdout** method is what we have alluded to so far in our discussions about accuracy. In this method, the given data are randomly partitioned into two independent sets, a *training set* and a *test set*. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model. The model's accuracy is then estimated with the test set (Fig. 6.19). The estimate is pessimistic because only a portion of the initial data is used to derive the model.

**Random subsampling** is a variation of the holdout method in which the holdout method is repeated  $k$  times. The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.

### 6.6.3 Cross-validation

In  **$k$ -fold cross-validation**, the initial data are randomly partitioned into  $k$  mutually exclusive subsets or “folds”  $D_1, D_2, \dots, D_k$ , each of approximately equal size. Training and testing are performed  $k$

times. In iteration  $i$ , partition  $D_i$  is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets  $D_2, \dots, D_k$  collectively serve as the training set to obtain the first model, which is tested on  $D_1$ ; the second iteration is trained on subsets  $D_1, D_3, \dots, D_k$  and tested on  $D_2$ ; and so on. Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and once for testing. For classification, the accuracy estimate is the overall number of correct classifications from the  $k$  iterations, divided by the total number of tuples in the initial data.

**Leave-one-out-cross-validation** is a special case of  $k$ -fold cross-validation where  $k$  is set to the number of initial tuples. That is, only one sample is “left out” at a time for the test set. Leave-one-out-cross-validation is often used when the initial data set is small. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data.

In practice, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

### 6.6.4 Bootstrap

Unlike the accuracy estimation methods just mentioned, the **bootstrap method** samples the given training tuples uniformly *with replacement*. That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set. For instance, imagine a machine that randomly selects tuples for our training set. In *sampling with replacement*, the machine is allowed to select the same tuple more than once.

There are several bootstrap methods. A commonly used one is the **.632 bootstrap**, which works as follows. Suppose we are given a data set of  $d$  tuples. The data set is sampled  $d$  times, with replacement, resulting in a *bootstrap sample* or training set of  $d$  samples. Some of the original data tuples will likely occur more than once in this sample. The data tuples that did not make it into the training set end up forming the test set. Suppose we were to try this out several times. As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap sample, and the remaining 36.8% will form the test set (hence, the name, .632 bootstrap).

“Where does the figure, 63.2%, come from?” Each tuple has a probability of  $1/d$  of being selected, so the probability of not being chosen is  $(1 - 1/d)$ . We have to select  $d$  times, so the probability that a tuple will not be chosen during this whole time is  $(1 - 1/d)^d$ . If  $d$  is large, the probability approaches  $e^{-1} = 0.368$ .<sup>14</sup> Thus 36.8% of tuples will not be selected for training and thereby end up in the test set, and the remaining 63.2% will form the training set.

We can repeat the sampling procedure  $k$  times, wherein each iteration, we use the current test set to obtain an estimated accuracy of the model obtained from the current bootstrap sample. The overall accuracy of the model,  $M$ , is then estimated as

$$Acc(M) = \frac{1}{k} \sum_{i=1}^k (0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set}), \quad (6.30)$$

<sup>14</sup>  $e$  is the base of natural logarithms, that is,  $e = 2.718$ .

where  $Acc(M_i)_{test\_set}$  is the accuracy of the model obtained with bootstrap sample  $i$  when it is applied to test set  $i$ .  $Acc(M_i)_{train\_set}$  is the accuracy of the model obtained with bootstrap sample  $i$  when it is applied to the original set of data tuples. Bootstrapping tends to be overly optimistic. It works best with small data sets.

### 6.6.5 Model selection using statistical tests of significance

Suppose that we have generated two classification models,  $M_1$  and  $M_2$ , from our data. We have performed 10-fold cross-validation to obtain a mean error rate<sup>15</sup> for each. How can we determine which model is best? It may seem intuitive to select the model with the lowest error rate; however, the mean error rates are just *estimates* of the error on the true population of future data cases. There can be considerable variance between error rates within any given 10-fold cross-validation experiment. Although the mean error rates obtained for  $M_1$  and  $M_2$  may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance? This section addresses these questions.

To determine if there is any “real” difference in the mean error rates of two models, we need to employ a *test of statistical significance*. In addition, we want to obtain some confidence limits for our mean error rates so that we can make statements like, “Any observed mean will not vary by  $\pm$  two standard errors 95% of the time for future samples” or “One model is better than the other by a margin of error of  $\pm$  4%.”

What do we need to perform the statistical test? Suppose that for each model, we did 10-fold cross-validation, say, 10 times, each time using a different 10-fold data partitioning. Each partitioning is independently drawn. We can average the 10 error rates obtained each for  $M_1$  and  $M_2$ , respectively, to obtain the mean error rate for each model. For a given model, the individual error rates calculated in the cross-validations may be considered different, independent samples from a probability distribution. In general, they follow a *t-distribution with  $k - 1$  degrees of freedom* where, here,  $k = 10$ . (This distribution looks very similar to a normal, or Gaussian, distribution even though the functions defining the two are quite different. Both are unimodal, symmetric, and bell-shaped.) This allows us to do hypothesis testing where the significance test used is the ***t-test***, or **Student’s *t-test***. Our hypothesis is that the two models are the same, or in other words, that the difference in mean error rate between the two is zero. If we can reject this hypothesis (referred to as the *null hypothesis*), then we can conclude that the difference between the two models is statistically significant, in which case we can select the model with the lower error rate.

In data mining practice, we may often employ a single test set, that is, the same test set can be used for both  $M_1$  and  $M_2$ . In such cases, we do a **pairwise comparison** of the two models *for each* 10-fold cross-validation round. That is, for the  $i$ th round of 10-fold cross-validation, the same cross-validation partitioning is used to obtain an error rate for  $M_1$  and  $M_2$ . Let  $err(M_1)_i$  (or  $err(M_2)_i$ ) be the error rate of model  $M_1$  (or  $M_2$ ) on round  $i$ . The error rates for  $M_1$  are averaged to obtain a mean error rate for  $M_1$ , denoted  $\overline{err}(M_1)$ . Similarly, we can obtain  $\overline{err}(M_2)$ . The variance of the difference between the two models is denoted  $var(M_1 - M_2)$ . The *t-test* computes the *t-statistic with  $k - 1$  degrees of freedom* for  $k$  samples. In our example, we have  $k = 10$  since, here, the  $k$  samples are our error rates obtained

<sup>15</sup> Recall that the error rate of a model,  $M$ , is  $1 - accuracy(M)$ .

from ten 10-fold cross-validations for each model. The  $t$ -statistic for pairwise comparison is computed as follows:

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{\text{var}(M_1 - M_2)/k}}, \quad (6.31)$$

where

$$\text{var}(M_1 - M_2) = \frac{1}{k} \sum_{i=1}^k [\text{err}(M_1)_i - \text{err}(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))]^2. \quad (6.32)$$

To determine whether  $M_1$  and  $M_2$  are significantly different, we compute  $t$  and select a **significance level**,  $sig$ . In practice, a significance level of 5% or 1% is typically used. We then consult a table for the  $t$ -distribution, available in standard textbooks on statistics. This table is usually shown arranged by degrees of freedom as rows and significance levels as columns. Suppose we want to ascertain whether the difference between  $M_1$  and  $M_2$  is significantly different for 95% of the population, that is,  $sig = 5\%$  or 0.05. We need to find the  $t$ -distribution value corresponding to  $k - 1$  degrees of freedom (or 9 degrees of freedom for our example) from the table. However, because the  $t$ -distribution is symmetric, typically only the upper percentage points of the distribution are shown. Therefore we look up the table value for  $z = sig/2$ , which, in this case, is 0.025, where  $z$  is also referred to as a **confidence limit**. If  $t > z$  or  $t < -z$ , then our value of  $t$  lies in the rejection region, within the distribution's tails. This means that we can reject the null hypothesis that the means of  $M_1$  and  $M_2$  are the same and conclude that there is a statistically significant difference between the two models. Otherwise, if we cannot reject the null hypothesis, we conclude that any difference between  $M_1$  and  $M_2$  can be attributed to chance.

If two test sets are available instead of a single test set, then a nonpaired version of the  $t$ -test is used, where the variance between the means of the two models is estimated as

$$\text{var}(M_1 - M_2) = \frac{\text{var}(M_1)}{k_1} + \frac{\text{var}(M_2)}{k_2}, \quad (6.33)$$

and  $k_1$  and  $k_2$  are the number of cross-validation samples (in our case, 10-fold cross-validation rounds) used for  $M_1$  and  $M_2$ , respectively. This is also known as the **two sample  $t$ -test**. When consulting the table of  $t$ -distribution, the number of degrees of freedom used is taken as the minimum number of degrees of the two models.

### 6.6.6 Comparing classifiers based on cost–benefit and ROC curves

The true positives, true negatives, false positives, and false negatives are also useful in assessing the **costs and benefits** (or risks and gains) associated with a classification model. The cost associated with a false negative (such as incorrectly predicting that a cancerous patient is not cancerous) is far greater than those of a false positive (incorrectly yet conservatively labeling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. These costs may consider the danger to the patient, financial costs of resulting therapies, and other hospital costs. Similarly, the benefits associated with a true positive decision may be different from those of a true negative. Up to now, to compute the classifier's accuracy, we have assumed equal costs and essentially divided the sum of true positives and true negatives by the total number of test tuples.

Alternatively, we can incorporate costs and benefits by computing the average cost (or benefit) per decision. Other applications involving cost–benefit analysis include loan application decisions and target marketing mailouts. For example, the cost of loaning to a defaulter greatly exceeds that of the lost business incurred by denying a loan to a nondefaulter. Similarly, in an application that tries to identify households that are likely to respond to mailouts of certain promotional material, the cost of mailouts to numerous households that do not respond may outweigh the cost of lost business from not mailing to households that would have responded. Other costs to consider in the overall analysis include the costs to collect the data and to develop the classification tools.

**Receiver operating characteristic curves** are a useful visual tool for comparing two classification models. ROC curves come from signal detection theory that was developed during World War II for the analysis of radar images. A ROC curve for a given model shows the trade-off between the *true positive rate* ( $TPR$ ) and the *false positive rate* ( $FPR$ ).<sup>16</sup> Given a test set and a model,  $TPR$  is the proportion of positive (or “yes”) tuples that are correctly labeled by the model;  $FPR$  is the proportion of negative (or “no”) tuples that are mislabeled as positive. Recall that  $TP$ ,  $FP$ ,  $P$ , and  $N$  are the number of true positive, false positive, positive, and negative tuples, respectively. From Section 6.6.1, we know that  $TPR = \frac{TP}{P}$ , which is sensitivity. Furthermore,  $FPR = \frac{FP}{N}$ , which is  $1 - \text{specificity}$ .

For a two-class problem, a ROC curve allows us to visualize the trade-off between the rate at which the model can accurately recognize positive cases vs. the rate at which it mistakenly identifies negative cases as positive for different portions of the test set. Any increase in  $TPR$  occurs at the cost of an increase in  $FPR$ . The area under the ROC curve is a measure of the accuracy of the model.

To plot a ROC curve for a given classification model,  $M$ , the model must be able to return a probability of the predicted class for each test tuple. With this information, we rank and sort the tuples so that the tuple that is most likely to belong to the positive or “yes” class appears at the top of the list, and the tuple that is least likely to belong to the positive class lands at the bottom of the list. Naïve Bayesian (Section 6.3) and logistic regression (Section 6.5) classifiers return a class probability distribution for each prediction and, therefore, are appropriate, although other classifiers, such as decision tree classifiers (Section 6.2), can easily be modified to return class probability predictions. Let the value that a probabilistic classifier returns for a given tuple  $X$  be  $f(X) \rightarrow [0, 1]$ . For a binary problem, a threshold  $t$  is typically selected so that tuples where  $f(X) \geq t$  are considered positive and all the other tuples are considered negative. Note that the number of true positives and the number of false positives are both functions of  $t$ , so that we could write  $TP(t)$  and  $FP(t)$ . Both are monotonic nonincreasing functions.

We first describe the general idea behind plotting a ROC curve and then follow up with an example. The vertical axis of a ROC curve represents  $TPR$ . The horizontal axis represents  $FPR$ . To plot a ROC curve for  $M$ , we begin as follows. Starting at the bottom left corner (where  $TPR = FPR = 0$ ), we check the tuple’s actual class label at the top of the list. If we have a true positive (i.e., a positive tuple that was correctly classified), then  $TP$  and thus  $TPR$  increase. On the graph, we move up and plot a point. If, instead, the model classifies a negative tuple as positive, we have a false positive, and so both  $FP$  and  $FPR$  increase. On the graph, we move right and plot a point. This process is repeated for each of the test tuples in ranked order, each time moving up on the graph for a true positive or toward the right for a false positive.

<sup>16</sup>  $TPR$  and  $FPR$  are the two operating characteristics being compared.

Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	P	0.90	1	0	5	4	0.2	0
2	P	0.80	2	0	5	3	0.4	0
3	N	0.70	2	1	4	3	0.4	0.2
4	P	0.60	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.50	5	4	1	0	1.0	0.8
10	N	0.40	5	5	0	0	1.0	1.0

FIGURE 6.20

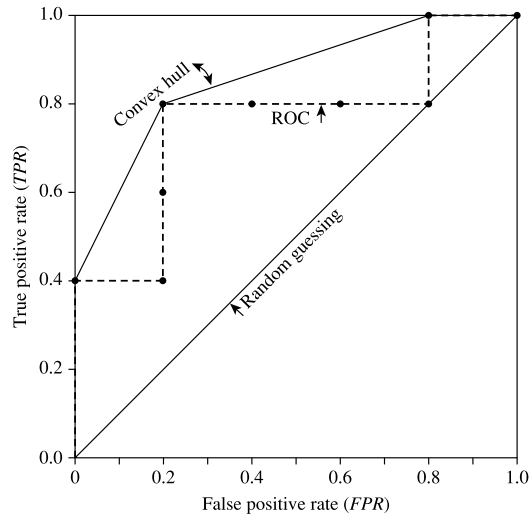
Tuples sorted by decreasing score, where the score is the value returned by a probabilistic classifier.

**Example 6.11. Plotting a ROC curve.** Fig. 6.20 shows the probability value (column 3) returned by a probabilistic classifier for each of the 10 tuples in a test set, sorted in the decreasing probability order. Column 1 is merely a tuple identification number, which aids in our explanation. Column 2 is the actual class label of the tuple. There are five positive tuples and five negative tuples; thus  $P = 5$  and  $N = 5$ . As we examine the known class label of each tuple, we can determine the values of the remaining columns,  $TP$ ,  $FP$ ,  $TN$ ,  $FN$ ,  $TPR$ , and  $FPR$ . We start with tuple 1, which has the highest probability score, and take that score as our threshold, that is,  $t = 0.9$ . Thus the classifier considers tuple 1 to be positive, and all the other tuples are considered negative. Since the actual class label of tuple 1 is positive, we have a true positive, hence  $TP = 1$  and  $FP = 0$ . Among the remaining nine tuples, which are all classified as negative, five actually are negative (thus,  $TN = 5$ ). The remaining four are all actually positive; thus,  $FN = 4$ . We can therefore compute  $TPR = \frac{TP}{P} = \frac{1}{5} = 0.2$ , whereas  $FPR = 0$ . Thus we have the point (0.2, 0) for the ROC curve.

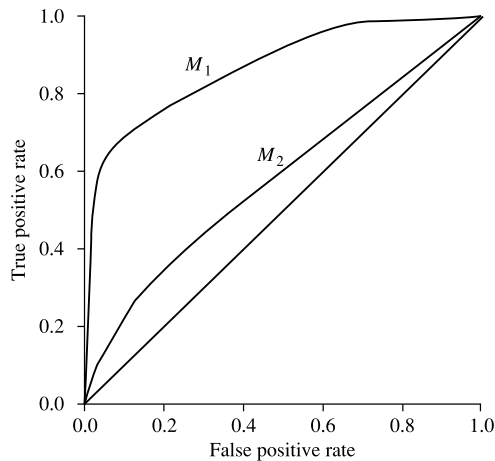
Next, threshold  $t$  is set to 0.8, the probability value for tuple 2, so this tuple is now also considered positive, whereas tuples 3 through 10 are considered negative. The actual class label of tuple 2 is positive, thus now  $TP = 2$ . The rest of the row can easily be computed, resulting in the point (0.4, 0). Next, we examine the class label of tuple 3 and let  $t$  be 0.7, the probability value returned by the classifier for that tuple. Thus tuple 3 is considered positive, yet its actual label is negative, and so it is a false positive. Thus  $TP$  stays the same and  $FP$  increments so that  $FP = 1$ . The rest of the values in the row can also be easily computed, yielding the point (0.4, 0.2). The resulting ROC graph, from examining each tuple, is the jagged line shown in Fig. 6.21.

There are many methods to obtain a curve out of these points, the most common of which is to use a convex hull. The plot also shows a diagonal line where for every true positive of such a model, we are just as likely to encounter a false positive. For comparison, this line represents random guessing.  $\square$

Fig. 6.22 shows the ROC curves of two classification models. The diagonal line representing random guessing is also shown. Thus the closer the ROC curve of a model is to the diagonal line, the less accurate the model. If the model is really good, initially we are more likely to encounter true positives as we move down the ranked list. Thus the curve moves steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve eases off and becomes more horizontal.

**FIGURE 6.21**

ROC curve for the data in Figure 6.20.

**FIGURE 6.22**

ROC curves of two classification models,  $M_1$  and  $M_2$ . The diagonal shows where, for every true positive, we are equally likely to encounter a false positive. The closer a ROC curve is to the diagonal line, the less accurate the model is. Thus  $M_1$  is more accurate here.

To assess the accuracy of a model, we can measure the area under the curve (AUC). Several software packages are able to perform such calculation. The closer the area is to 0.5, the less accurate the corresponding model is. A model with perfect accuracy will have an AUC of 1.0.



## 6.7 Techniques to improve classification accuracy

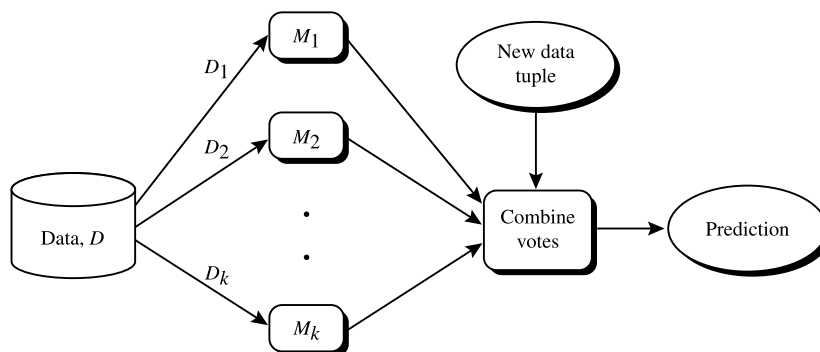
In this section, you will learn some tricks for increasing classification accuracy. We focus on *ensemble methods*. An ensemble for classification is a composite model, made up of a combination of classifiers. The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes. Ensembles tend to be more accurate than their component classifiers. We start off in Section 6.7.1 by introducing ensemble methods in general. Bagging (Section 6.7.2), boosting (Section 6.7.3), and random forests (Section 6.7.4) are popular ensemble methods.

Traditional learning models assume that the data classes are well distributed. In many real-world data domains, however, the data are class-imbalanced, where the main class of interest is represented by only a few tuples. This is known as the *class imbalance problem*. We also study techniques for improving the classification accuracy of class-imbalanced data. These are presented in Section 6.7.5.

### 6.7.1 Introducing ensemble methods

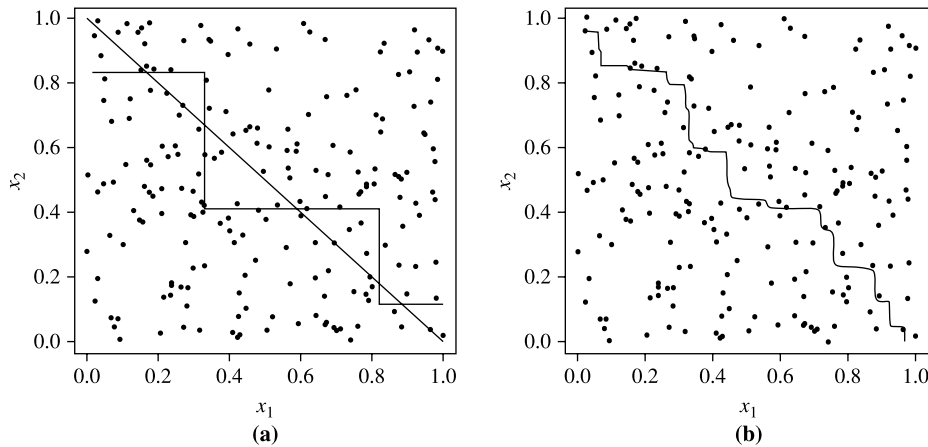
*Bagging*, *boosting*, and *random forests* are examples of **ensemble methods** (Fig. 6.23). An ensemble combines a series of  $k$  learned models (or *base classifiers*),  $M_1, M_2, \dots, M_k$ , with the aim of creating an improved composite classification model,  $M^*$ . A given data set,  $D$ , is used to create  $k$  training sets,  $D_1, D_2, \dots, D_k$ , where  $D_i$  ( $1 \leq i \leq k$ ) is used to generate classifier  $M_i$ . Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.

An ensemble tends to be more accurate than its base classifiers. For example, consider an ensemble that performs majority voting. That is, given a tuple  $X$  to classify, it collects the class label predictions returned from the base classifiers and outputs the class in the majority. The base classifiers may make mistakes, but the ensemble will misclassify  $X$  only if over half of the base classifiers are in error. Ensembles yield better results when there is significant diversity among the models. That is, ideally,



**FIGURE 6.23**

Increasing classifier accuracy. Ensemble methods generate a set of classification models,  $M_1, M_2, \dots, M_k$ . Given a new data tuple to classify, each classifier “votes” for the class label of that tuple. The ensemble combines the votes to return a class prediction.



**FIGURE 6.24**

Decision boundary by (a) a single decision tree and (b) an ensemble of decision trees for a linearly separable problem (i.e., where the actual decision boundary is a straight line). The decision tree struggles with approximating a linear boundary. The decision boundary of the ensemble is closer to the true boundary. *Source:* From Seni and Elder [SE10]. © 2010 Morgan & Claypool Publishers; used with permission.

there is little correlation among classifiers. The base classifiers should also perform better than random guessing. Each base classifier can be allocated to a different CPU and so ensemble methods are parallelizable.

To help illustrate the power of an ensemble, consider a simple two-class problem described by two attributes,  $x_1$  and  $x_2$ . The problem has a linear decision boundary. Fig. 6.24(a) shows the decision boundary of a decision tree classifier on the problem. Fig. 6.24(b) shows the decision boundary of an ensemble of decision tree classifiers on the same problem. Although the ensemble's decision boundary is still piecewise constant, it has a finer resolution and is better than that of a single tree.

### 6.7.2 Bagging

We now take an intuitive look at how bagging works as a method of increasing accuracy. Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than any other, you may choose this as the final or best diagnosis. That is, the final diagnosis is made based on a majority vote, where each doctor gets an equal vote. Now replace each doctor by a classifier, and you have the basic idea behind bagging. Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group.

Given a set,  $D$ , of  $d$  tuples, **bagging** works as follows. For iteration  $i$  ( $i = 1, 2, \dots, k$ ), a training set,  $D_i$ , of  $d$  tuples is sampled with replacement from the original set of tuples,  $D$ . Note that the term *bagging* stands for *bootstrap aggregation*. Each training set is a bootstrap sample, as described in Section 6.6.4. Because sampling with replacement is used, some of the original tuples of  $D$  may not

**Algorithm: Bagging.** The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

**Input:**

- $D$ , a set of  $d$  training tuples;
- $k$ , the number of models in the ensemble;
- a classification learning scheme (e.g., decision tree algorithm, naïve Bayesian, etc.).

**Output:** The ensemble—a composite model,  $M^*$ .

**Method:**

- (1) **for**  $i = 1$  to  $k$  **do** // create  $k$  models:
- (2)     create bootstrap sample,  $D_i$ , by sampling  $D$  with replacement;
- (3)     use  $D_i$  and the learning scheme to derive a model,  $M_i$ .
- (4) **endfor**

**To use the ensemble to classify a tuple,  $X$ :**

let each of the  $k$  models classify  $X$  and return the majority vote;

**FIGURE 6.25**

Bagging.

be included in  $D_i$ , whereas others may occur more than once. A classifier model,  $M_i$ , is learned for each training set,  $D_i$ . To classify an unknown tuple,  $X$ , each classifier,  $M_i$ , returns its class prediction, which counts as one vote. The bagged classifier,  $M^*$ , counts the votes and assigns the class with the most votes to  $X$ . Bagging can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple. The algorithm is summarized in Fig. 6.25.

The bagged classifier often has significantly greater accuracy than a single classifier derived from  $D$ , the original training data. It is often more robust to the effects of noisy data and overfitting. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.

### 6.7.3 Boosting

We now look at the ensemble method of boosting. As in the previous section, suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor’s diagnosis based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting.

In **boosting**, weights are also assigned to each training tuple. A series of  $k$  classifiers is iteratively learned. After a classifier,  $M_i$ , is learned, the weights are updated to allow the subsequent classifier,  $M_{i+1}$ , to “pay more attention” to the training tuples that were misclassified by  $M_i$ . The final boosted classifier,  $M^*$ , combines the votes of each individual classifier, where the weight of each classifier’s vote is a function of its accuracy.

**AdaBoost** (short for Adaptive Boosting) is a popular boosting algorithm. Suppose we want to boost the accuracy of a learning method. We are given  $D$ , a data set of  $d$  class-labeled tuples,  $(X_1, y_1), (X_2, y_2), \dots, (X_d, y_d)$ , where  $y_i$  is the class label of tuple  $X_i$ . Initially, AdaBoost assigns each training tuple an equal weight of  $1/d$ . Generating  $k$  classifiers for the ensemble requires  $k$  rounds

**Algorithm: AdaBoost.** A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

**Input:**

- $D$ , a set of  $d$  class-labeled training tuples;
- $k$ , the number of rounds (one classifier is generated per round);
- a classification learning scheme.

**Output:** A composite model.

**Method:**

- (1) initialize the weight of each tuple in  $D$  to  $1/d$ ;
- (2) **for**  $i = 1$  to  $k$  **do** // for each round:
  - (3) sample  $D$  with replacement according to the tuple weights to obtain  $D_i$ ;
  - (4) use training set  $D_i$  to derive a model,  $M_i$ ;
  - (5) compute  $error(M_i)$ , the error rate of  $M_i$  (Eq. (6.34))
  - (6) **if**  $error(M_i) > 0.5$  **then**
    - (7) abort the loop;
  - (8) **endif**
  - (9) **for** each tuple in  $D$  that was correctly classified **do**
    - (10) multiply the weight of the tuple by  $error(M_i)/(1 - error(M_i))$ ; // update weights
  - (11) normalize the weight of each tuple.
- (12) **endfor**

**To use the ensemble to classify tuple,  $X$ :**

- (1) initialize weight of each class to 0;
- (2) **for**  $i = 1$  to  $k$  **do** // for each classifier:
  - (3)  $w_i = \log \frac{1 - error(M_i)}{error(M_i)}$ ; // weight of the classifier's vote
  - (4)  $c = M_i(X)$ ; // get class prediction for  $X$  from  $M_i$
  - (5) add  $w_i$  to the weight for class  $c$
- (6) **endfor**
- (7) return the class with the largest weight.

**FIGURE 6.26**

AdaBoost, a boosting algorithm.

through the rest of the algorithm. We can sample to form any sized training set, not necessarily of size  $d$ . Sampling with replacement is used—the same tuple may be selected more than once. Each tuple's chance of being selected is based on its weight. A classifier model,  $M_i$ , is derived from the training tuples of  $D_i$ . Its error is then calculated using  $D$  as the test set. The weights of the tuples are then adjusted according to how they were classified.

If a tuple was incorrectly classified, its weight is increased. If a tuple was correctly classified, its weight is decreased. A tuple's weight reflects how difficult it is to classify—the higher the weight, the more often it has been misclassified. These weights will be used to generate the training samples for the classifier of the next round. The basic idea is that when we build a classifier, we want it to focus more on the misclassified tuples of the previous round. Some classifiers may be better at classifying some “difficult” tuples than others. In this way, we build a series of classifiers that complement each other. The algorithm is summarized in Fig. 6.26.

Now, let's look at some of the math that's involved in the algorithm. To compute the error rate of model  $M_i$ , we sum the weights of each of the tuples in  $D$  that  $M_i$  misclassified. That is,

$$\text{error}(M_i) = \sum_{j=1}^d w_j \times \text{err}(X_j), \quad (6.34)$$

where  $\text{err}(X_j)$  is the misclassification error of tuple  $X_j$ : If the tuple was misclassified, then  $\text{err}(X_j)$  is 1; otherwise, it is 0. If the performance of classifier  $M_i$  is so poor that its error exceeds 0.5, then we abandon it. Instead, we try again by generating a new  $D_i$  training set, from which we derive a new  $M_i$ .

The error rate of  $M_i$  affects how the weights of the training tuples are updated. If a tuple in the round  $i$  was correctly classified, its weight is multiplied by  $\text{error}(M_i)/(1 - \text{error}(M_i))$ . Once the weights of all the correctly classified tuples are updated, the weights for all tuples (including the misclassified ones) are normalized so that their sum remains the same as it was before. To normalize a weight, we multiply it by the sum of the old weights, divided by the sum of the new weights. As a result, the weights of misclassified tuples are increased, and the weights of correctly classified tuples are decreased, as described before.

*“Once boosting is complete, how is the ensemble of classifiers used to predict the class label of a tuple,  $X$ ?”* Unlike bagging, where each classifier was assigned an equal vote, boosting assigns a weight to each classifier's vote, based on how well the classifier performed. The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be. The weight of classifier  $M_i$ 's vote is

$$\log \frac{1 - \text{error}(M_i)}{\text{error}(M_i)}. \quad (6.35)$$

For each class,  $c$ , we sum the weights of each classifier that assigned class  $c$  to  $X$ . The class with the highest sum is the “winner” and is returned as the class prediction for tuple  $X$ .

**Gradient boosting** is another powerful boosting technique, which can be used for classification, regression, and ranking. If we use a tree (e.g., decision tree for classification, regression tree for regression) as the base model (i.e., the weak learner), it is called **gradient tree boosting**, or **gradient boosted tree**. Fig. 6.27 presents the gradient tree boosting algorithm for the regression task. It works as follows.

Gradient tree boosting algorithm starts with a simple regression model  $F(x)$  (line 1), which outputs a constant (i.e., the average output of all training tuples). Then, similar to Adaboost, it tries to find a new base model (i.e., weak learner)  $M_t(x)$  at each round (line 3). The newly constructed base model  $M_t(x)$  is added to the regression model  $F(x)$  (line 8). In other words, the composite regression model  $F(x)$  consists of  $k$  additive base models  $M_t(x)$  ( $t = 1, \dots, k$ ). When we search for a new base model  $M_t(x)$ , all the previously constructed base models (i.e.,  $M_1(x), \dots, M_{t-1}(x)$ ) are kept unchanged.

In order to construct a new base model  $M_t(x)$ , we first compute the predicted output  $\hat{y}_i$  of each training tuple by the current regression model  $F(x)$  (line 4) and calculate the **negative gradient**  $r_i$  of the loss function with respect to the predicted output  $\hat{y}_i$  (line 5). Then, we fit a regression tree model for the training set  $\{(x_1, r_1), \dots, (x_n, r_n)\}$ , where the negative gradient  $r_i$  is treated as the targeted output value of the  $i$ th training tuple. Since the negative gradient  $r_i$  ( $i = 1, \dots, n$ ) changes in different rounds, we end up with different base models  $M_t(x)$  ( $t = 1, \dots, k$ ).

*“But, why do we use the negative gradients to construct the new base model?”* Suppose the loss function  $L(y_i, F(x_i)) = \frac{1}{2}(y_i - \hat{y}_i)^2$  (recall that we have used the similar loss function for the regression tree and the least square linear regression model). Then, we can show that the negative gradient

**Algorithm: Gradient Tree Boosting for Regression.****Input:**

- $D$ , a set of  $n$  training tuples  $\{(x_1, y_1), \dots, (x_n, y_n)\}$ , where  $x_i$  is the attribute vector of the  $i$ th training tuple and  $y_i$  is its true target output value;
- $k$ , the number of rounds (one base regression model is generated per round);
- a differential loss function  $\text{Loss} = \sum_{i=1}^n L(y_i, F(x_i))$ .

**Output:** A composite regression model  $F(x)$ .**Method:**

- (1) initialize the regression model  $F(x) = \frac{\sum_{i=1}^n y_i}{n}$ ;
- (2) **for**  $t = 1$  to  $k$  **do** // construct a new weak learner  $M_t(x)$  for each round:
- (3)     **for**  $i = 1$  to  $n$  //each training tuple:
- (4)         calculate  $\hat{y}_i = F(x_i)$ ; //predicted value by the current model  $F(x)$
- (5)         calculate the negative gradient  $r_i = -\frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$ ;
- (6)     **endfor**
- (7)     fit a regression tree model  $M_t(x)$  for the training set  $\{(x_1, r_1), \dots, (x_n, r_n)\}$ ;
- (8)     update the composite regression model  $F(x) \leftarrow F(x) + M_t(x)$ .
- (9) **endfor**

**FIGURE 6.27**

Gradient tree boosting for regression.

$r_i = y_i - \hat{y}_i$ , which is the difference between the actual output value and predicted output value by the current regression model  $F(x)$  (i.e., the residual). In other words, the negative gradient  $r_i$  reveals the “shortcoming” of the current regression model  $F(x)$  (i.e., how far away the predicted output is from its actual output value). If we use other loss functions (e.g., the Huber loss in robust regression), the negative gradient is no longer equal to the residual  $y_i - \hat{y}_i$ , but still provides a good indicator in terms of the prediction quality of the current regression model  $F(x)$  on the  $i$ th training tuple. For this reason, the negative gradients are also referred to as *pseudo residuals*. By fitting a regression tree model with respect to the negative gradients (i.e., where the “shortcoming” of the current regression model  $F(x)$  is), the newly constructed base model,  $M_t(x)$ , is expected to dramatically improve the composite regression model  $F(x)$ .

In addition to the algorithm in Fig. 6.27, several alternative design choices for gradient tree boosting exist. For example, similar to Adaboost, we can learn a weight for each base model  $M_t(x)$ , and then the composite regression model  $F(x)$  becomes the *weighted sum* of the  $k$  base models. In practice, it was found that shrinking the newly constructed base model helps improve the generalization performance of the composite model  $F(x)$  (i.e.  $F(x) \leftarrow F(x) + \eta M_t(x)$  in line 8, where  $0 < \eta < 1$  is the shrinkage constant.). The number of leaf nodes  $T$  of the regression tree  $M_t(x)$  plays an important role in the learning performance of the composite model  $F(x)$ . That is,  $F(x)$  might underfit the training set if  $T$  is too small, but could overfit the training set with a large  $T$ . The typical choice for  $T$  is between 4 and 8. At a given round  $t$ , we could use a subsample of the entire training set to construct the base model  $M_t(x)$ . Gradient tree boosting equipped with such a subsampling strategy is referred to as *stochastic gradient (tree) boosting* and it was found to significantly improve the accuracy of the composite model  $F(x)$ . A highly scalable end-to-end gradient tree boosting system is called **XGBoost**, which is capable to handle a billion-scale training set. XGBoost has made a number of innovations for training gradient

tree boosting, including a new tree construction algorithm designed for sparse data, feature subsampling (as opposed to training tuple subsampling in stochastic gradient boosting), and a highly efficient cache-aware block structure. XGBoost has been successfully used by data scientists in many data mining challenges, often leading to top competitive results.

“How does boosting compare with bagging?” Because of the way boosting focuses on the misclassified tuples, it risks overfitting the resulting composite model to such data. Therefore sometimes the resulting “boosted” model may be less accurate than a single model derived from the same data. Bagging is less susceptible to model overfitting. While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

### 6.7.4 Random forests

We now present another ensemble method called **random forests**. Imagine that each of the classifiers in the ensemble is a *decision tree* classifier so that the collection of classifiers is a “forest.” The individual decision trees are generated using a random selection of attributes at each node to determine the split. More formally, each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. During classification, each tree votes, and the most popular class is returned.

Random forests can be built using bagging (Section 6.7.2) in tandem with random attribute selection. A training set,  $D$ , of  $d$  tuples is given. The general procedure to generate  $k$  decision trees for the ensemble is as follows. For each iteration,  $i$  ( $i = 1, 2, \dots, k$ ), a training set,  $D_i$ , of  $d$  tuples is sampled with replacement from  $D$ . That is, each  $D_i$  is a bootstrap sample of  $D$  (Section 6.6.4), so that some tuples may occur more than once in  $D_i$ , while others may be excluded. Let  $F$  be the number of attributes to be used to determine the split at each node, where  $F$  is much smaller than the number of available attributes. To construct a decision tree classifier,  $M_i$ , randomly select, at each node,  $F$  attributes as candidates for the split at the node. The CART methodology is used to grow the trees. The trees are grown to maximum size and are not pruned. Random forests formed this way, with *random input selection*, are called Forest-RI.

Another form of random forest, called Forest-RC, uses *random linear combinations* of the input attributes. Instead of randomly selecting a subset of the attributes, it creates new attributes (or features) that are a linear combination of the existing attributes. That is, an attribute is generated by specifying  $L$ , the number of original attributes to be combined. At a given node,  $L$  attributes are randomly selected and added together with coefficients that are uniform random numbers on  $[-1, 1]$ .  $F$  linear combinations are generated, and a search is made over these for the best split. This form of random forest is useful when there are only a few attributes available, so as to reduce the correlation between individual classifiers.

Random forests are comparable in accuracy to AdaBoost, yet are more robust to errors and outliers. The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is less likely to be a problem. The accuracy of a random forest depends on the strength of the individual classifiers and a measure of the dependence between them. The ideal is to maintain the strength of individual classifiers without increasing their correlation. Random forests are insensitive to the number of attributes selected for consideration at each split. Typically, up to  $\log_2 d + 1$  are chosen. (An interesting empirical observation was that using a single random input attribute may result in good accuracy that is often higher than when using several attributes.) Because random forests consider much

fewer attributes for each split, they are efficient on very large databases. They can be faster than either bagging or boosting. Random forests give internal estimates of variable importance.

### 6.7.5 Improving classification accuracy of class-imbalanced data

In this section, we revisit the *class imbalance problem*. In particular, we study approaches to improving the classification accuracy of class-imbalanced data.

Given two-class data, the data are class-imbalanced if the main class of interest (the positive class) is represented by only a few tuples, while the majority of tuples represent the negative class. For multiclass-imbalanced data, the data distribution of each class differs substantially where, again, the main class or classes of interest are rare. The class imbalance problem is closely related to cost-sensitive learning, wherein the costs of errors per class are not equal. In medical diagnosis, for example, it is much more costly to falsely diagnose a cancerous patient as healthy (a false negative) than to misdiagnose a healthy patient as having cancer (a false positive). A false negative error could lead to the loss of life and therefore is much more expensive than a false positive error. Other applications involving class-imbalanced data include fraud detection, the detection of oil spills from satellite radar images, and fault monitoring.

Traditional classification algorithms aim to minimize the number of errors made during classification. They assume that the costs of false positive and false negative errors are equal. By assuming a balanced distribution of classes and equal error costs, they are therefore not suitable for class-imbalanced data. Earlier parts of this chapter presented ways of addressing the class imbalance problem. Although the accuracy measure assumes that the cost of classes are equal, alternative evaluation metrics can be used that consider the different types of classifications. Section 6.6.1, for example, presented *sensitivity* or recall (the true positive rate) and *specificity* (the true negative rate), which help to assess how well a classifier can predict the class label of imbalanced data. Additional relevant measures discussed include  $F_1$  and  $F_\beta$ . Section 6.6.6 showed how ROC curves plot *sensitivity* vs.  $1 - \textit{specificity}$  (i.e., the false positive rate). Such curves can provide insight when studying the performance of classifiers on class-imbalanced data.

In this section, we look at general approaches for *improving* the classification accuracy of class-imbalanced data. These approaches include (1) oversampling, (2) undersampling, (3) threshold moving, and (4) ensemble techniques. The first three do not involve any changes to the construction of the classification model. That is, oversampling and undersampling change the distribution of tuples in the training set; threshold moving affects how the model makes decisions when classifying new data. Ensemble methods follow the techniques described in Section 6.7.2 through Section 6.7.4. For ease of explanation, we describe these general approaches with respect to the two-class imbalanced data problem, where the higher-cost classes are rarer than the lower-cost classes.

Both oversampling and undersampling change the training data distribution so that the rare (positive) class is well represented. **Oversampling** works by resampling the positive tuples so that the resulting training set contains an equal number of positive and negative tuples. **Undersampling** works by decreasing the number of negative tuples. It randomly eliminates tuples from the majority (negative) class until there are an equal number of positive and negative tuples.

**Example 6.12. Oversampling and undersampling.** Suppose the original training set contains 100 positive and 1000 negative tuples. In oversampling, we replicate tuples of the rare class to form a new training set containing 1000 positive tuples and 1000 negative tuples. In undersampling, we randomly



eliminate negative tuples so that the new training set contains 100 positive tuples and 100 negative tuples.  $\square$

Several variations to oversampling and undersampling exist. They may vary, for instance, in how tuples are added or eliminated. For example, the SMOTE algorithm uses oversampling where synthetic tuples are added, which are “close to” the given positive tuples in tuple space.

The **threshold-moving** approach to the class imbalance problem does not involve any sampling. It applies to classifiers that, given an input tuple, return a continuous output value (just like in Section 6.6.6, where we discussed how to construct ROC curves). That is, for an input tuple,  $X$ , such a classifier returns as output a mapping,  $f(X) \rightarrow [0, 1]$ . Rather than manipulating the training tuples, this method returns a classification decision based on the output values. In the simplest approach, tuples for which  $f(X) \geq t$ , for some threshold,  $t$ , are considered positive, while all other tuples are considered negative. Other approaches may involve manipulating the outputs by weighting. In general, threshold moving moves the threshold,  $t$ , so that the rare class tuples are easier to classify (and hence, there is less chance of costly false negative errors). Examples of such classifiers include naïve Bayesian classifiers (Section 6.3) and neural networks (Chapter 10). The threshold-moving method, although not as popular as over- and undersampling, is simple and has shown some success for the two-class-imbalanced data.

Ensemble methods (Section 6.7.2 through Section 6.7.4) have also been applied to the class imbalance problem. The individual classifiers making up the ensemble may include versions of the approaches described here, such as oversampling and threshold moving.

These methods work relatively well for the class imbalance problem on two-class tasks. Threshold-moving and ensemble methods were empirically observed to outperform oversampling and undersampling. Threshold moving works well even on extremely imbalanced data sets. The class imbalance problem on multiclass tasks is much more difficult where oversampling and threshold moving are less effective. Although threshold-moving and ensemble methods show promise, finding a solution for the multiclass imbalance problem remains an area of future work.

---

## 6.8 Summary

- **Classification** is a form of data analysis that extracts models describing data classes. A classifier, or classification model, predicts categorical labels (classes). **Numeric prediction** models continuous-valued functions. Classification and numeric prediction are the two major types of prediction problems.
- **Decision tree induction** is a top-down recursive tree induction algorithm, which uses an attribute selection measure to select the attribute tested for each nonleaf node in the tree. **ID3**, **C4.5**, and **CART** are examples of such algorithms using different attribute selection measures. **Tree pruning** algorithms attempt to improve accuracy by removing tree branches reflecting noise in the data.
- **Naïve Bayesian classification** is based on Bayes’ theorem of the posterior probability. It assumes class-conditional independence—that the effect of an attribute value on a given class is independent of the values of other attributes.
- **Linear classifiers** compute a linear weighted combination of the input attribute values, based on which it predicts the class label for a given tuple. **Perceptron** and **logistic regression** are two classic examples of linear classifiers.

- Decision tree classifiers, Bayesian classifiers, and linear classifiers are all examples of **eager learners** in that they use training tuples to construct a generalization model and in this way are ready for classifying new tuples. This contrasts with **lazy learners** or **instance-based** methods of classification, such as nearest-neighbor classifiers and case-based reasoning classifiers, which store all of the training tuples in pattern space and wait until presented with a test tuple before performing generalization. Hence lazy learners require efficient indexing techniques.
- A **confusion matrix** can be used to evaluate a classifier's quality. For a two-class problem, it shows the *true positives*, *true negatives*, *false positives*, and *false negatives*. Measures that assess a classifier's predictive ability include **accuracy**, **sensitivity** (also known as **recall**), **specificity**, **precision**,  $F$ , and  $F_\beta$ . Reliance on the accuracy measure can be deceiving when the main class of interest is in the minority.
- Construction and evaluation of a classifier require partitioning labeled data into a training set and a test set. **Holdout**, **random sampling**, **cross-validation**, and **bootstrapping** are typical methods used for such partitioning.
- Significance tests and ROC curves are useful tools for model selection. **Significance tests** can be used to assess whether the difference in accuracy between two classifiers is due to chance. **ROC curves** plot the true positive rate (or sensitivity) vs. the false positive rate (or  $1 - \textit{specificity}$ ) of one or more classifiers.
- **Ensemble methods** can be used to increase overall accuracy by learning and combining a series of individual (base) classifier models. **Bagging**, **boosting**, and **random forests** are popular ensemble methods.
- The **class imbalance problem** occurs when the main class of interest is represented by only a few tuples. Strategies to address this problem include **oversampling**, **undersampling**, **threshold moving**, and **ensemble techniques**.

---

## 6.9 Exercises

- 6.1. Briefly outline the major steps of *decision tree classification*.
- 6.2. Why is *tree pruning* useful in decision tree induction? What is a drawback of using a separate set of tuples to evaluate pruning?
- 6.3. Given a decision tree, you have the option of (a) *converting* the decision tree to rules and then pruning the resulting rules, or (b) *pruning* the decision tree and then converting the pruned tree to rules. What advantage does (a) have over (b)?
- 6.4. It is important to calculate the worst-case computational complexity of the decision tree algorithm. Given data set,  $D$ , the number of attributes,  $n$ , and the number of training tuples,  $|D|$ , analyze the computational complexity in terms of  $n$  and  $|D|$ .
- 6.5. Given a 5-GB data set with 50 attributes (each containing 100 distinct values) and 512 MB of main memory in your laptop, outline an efficient method that constructs decision trees in such large data sets. Justify your answer by a rough calculation of your main memory usage.
- 6.6. Why is *naïve Bayesian classification* called “naïve”? Briefly outline the major ideas of naïve Bayesian classification.
- 6.7. The following table consists of training data from an employee database. The data have been generalized. For example, “31 . . . 35” for *age* represents the age range of 31 to 35. For a given

row entry, *count* represents the number of data tuples having the values for *department*, *status*, *age*, and *salary* given in that row.

<i>department</i>	<i>status</i>	<i>age</i>	<i>salary</i>	<i>count</i>
sales	senior	31...35	46K...50K	30
sales	junior	26...30	26K...30K	40
sales	junior	31...35	31K...35K	40
systems	junior	21...25	46K...50K	20
systems	senior	31...35	66K...70K	5
systems	junior	26...30	46K...50K	3
systems	senior	41...45	66K...70K	3
marketing	senior	36...40	46K...50K	10
marketing	junior	31...35	41K...45K	4
secretary	senior	46...50	36K...40K	4
secretary	junior	26...30	26K...30K	6

Let *status* be the class label attribute.

- a. How would you modify the basic decision tree algorithm to take into consideration the *count* of each generalized data tuple (i.e., of each row entry)?
  - b. Use your algorithm to construct a decision tree from the given data.
  - c. Given a data tuple having the values “*systems*,” “*26...30*,” and “*46–50K*” for the attributes *department*, *age*, and *salary*, respectively, what would a naïve Bayesian classification of the *status* for the tuple be?
- 6.8. Compare the advantages and disadvantages of *eager* classification (e.g., decision tree, Bayesian, neural network) vs. *lazy* classification (e.g., *k*-nearest neighbor, case-based reasoning).
  - 6.9. Write an algorithm for *k-nearest-neighbor classification* given *k*, the nearest number of neighbors, and *n*, the number of attributes describing each tuple.
  - 6.10. RainForest is a scalable algorithm for decision tree induction. Develop a scalable naïve Bayesian classification algorithm that requires just a single scan of the entire data set for most databases. Discuss whether such an algorithm can be refined to incorporate *boosting* to further enhance its classification accuracy.
  - 6.11. Design an efficient method that performs effective naïve Bayesian classification over an *infinite* data stream (i.e., you can scan the data stream only once). If we wanted to discover the *evolution* of such classification schemes (e.g., comparing the classification scheme at this moment with earlier schemes such as one from a week ago), what modified design would you suggest?
  - 6.12. The perceptron model  $y = f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$  can be used to learn a binary classifier from training data.
    - a. Assume there are two training samples. The positive one is  $\mathbf{x}_1 = (2, 1)^T$ ; the negative one is  $\mathbf{x}_2 = (1, 0)^T$ . The learning rate  $\eta = 1$ . Starting from  $\mathbf{w} = (1, 1)^T$  and  $b = 0$ , solve the parameters of the classifier.
    - b. Assume there are four training samples. The positive samples are  $\mathbf{x}_1 = (1, 1)^T$  and  $\mathbf{x}_2 = (0, 0)^T$ ; the negative samples are  $\mathbf{x}_3 = (1, 0)^T$  and  $\mathbf{x}_4 = (0, 1)^T$ . Can we classify all training samples correctly using the perceptron model? Why?
  - 6.13. Suppose we have three positive examples  $x_1 = (1, 0, 0)$ ,  $x_2 = (0, 0, 1)$  and  $x_3 = (0, 1, 0)$  and three negative examples  $x_4 = (-1, 0, 0)$ ,  $x_5 = (0, -1, 0)$  and  $x_6 = (0, 0, -1)$ . Apply standard gradient ascent method to train a logistic regression classifier (without regularization terms).

Tuple #	Class	Probability
1	<i>P</i>	0.95
2	<i>N</i>	0.85
3	<i>P</i>	0.78
4	<i>P</i>	0.66
5	<i>N</i>	0.60
6	<i>P</i>	0.55
7	<i>N</i>	0.53
8	<i>N</i>	0.52
9	<i>N</i>	0.51
10	<i>P</i>	0.40

FIGURE 6.28

Tuples sorted by decreasing score, where the score is the value returned by a probabilistic classifier.

- Initialize the weight vector with two different values and set  $w_0^0 = 0$  (e.g.  $w_0 = (0, 0, 0, 0)'$ ,  $w_0 = (0, 0, 1, 0)'$ ). Would the final weight vector ( $w^*$ ) be the same for the two different initial values? What are the values? Please explain your answer in detail. You may assume the learning rate to be a positive real constant  $\eta$ .
- 6.14.** Suppose that we are training a naïve Bayes classifier and a logistic regression classifier:  $f : \mathbf{X} \rightarrow Y$ , which maps a  $d$ -dimensional real-valued feature vector  $\mathbf{X} \in \mathbb{R}^d$  to a binary class label  $Y \in \{0, 1\}$ . In the naïve Bayes classifier, we assume that all  $\mathbf{X}_i$  where  $i = 1, \dots, n$  are conditionally independent given the class label  $Y$  and the class prior  $P(Y)$  follow the Bernoulli distribution with  $P(Y = 1) = \theta$ . Now, prove the equivalence of logistic regression and naïve Bayes under these two assumptions.
- For each  $\mathbf{X}_i$ , we assume it is drawn from the Gaussian distribution  $P(\mathbf{X}_i | Y = k) \sim \mathcal{N}(\mu_{ik}, \sigma_{ik})$  where  $k = 0, 1$ . We also assume that  $\sigma_{i0} = \sigma_{i1} = \sigma_i$ .
  - For each  $\mathbf{X}_i$ , we assume it is drawn from the Bernoulli distribution  $P(\mathbf{X}_i = 1 | Y = k) = p_k$  where  $k = 0, 1$ .
- 6.15.** Show that accuracy is a function of *sensitivity* and *specificity*, that is, prove Eq. (6.25).
- 6.16.** The harmonic mean is one of several kinds of averages. Chapter 2 discussed how to compute the *arithmetic mean*, which is what most people typically think of when they compute an average. The **harmonic mean**,  $H$ , of the positive real numbers,  $x_1, x_2, \dots, x_n$ , is defined as

$$\begin{aligned}
 H &= \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}} \\
 &= \frac{n}{\sum_{i=1}^n \frac{1}{x_i}}.
 \end{aligned}$$

- The  $F$  measure is the harmonic mean of precision and recall. Use this fact to derive Eq. (6.28) for  $F$ . In addition, write  $F_\beta$  as a function of true positives, false negatives, and false positives.
- 6.17.** The data tuples of Fig. 6.28 are sorted by decreasing probability value, as returned by a classifier. For each tuple, compute the values for the number of true positives ( $TP$ ), false positives ( $FP$ ), true negatives ( $TN$ ), and false negatives ( $FN$ ). Compute the true positive rate ( $TPR$ ) and false positive rate ( $FPR$ ). Plot the ROC curve for the data.

- 6.18.** It is difficult to assess classification *accuracy* when individual data objects may belong to more than one class at a time. In such cases, comment on what criteria you would use to compare different classifiers modeled after the same data.
- 6.19.** Suppose that we want to *select between two prediction models*,  $M_1$  and  $M_2$ . We have performed 10 rounds of 10-fold cross-validation on each model, where the same data partitioning in round  $i$  is used for both  $M_1$  and  $M_2$ . The error rates obtained for  $M_1$  are 30.5, 32.2, 20.7, 20.6, 31.0, 41.0, 27.7, 26.0, 21.5, 26.0. The error rates for  $M_2$  are 22.4, 14.5, 22.4, 19.6, 20.7, 20.4, 22.1, 19.4, 16.2, 35.0. Comment on whether one model is significantly better than the other considering a significance level of 1%.
- 6.20.** What is *boosting*? State why it may improve the accuracy of decision tree induction.
- 6.21.** Outline methods for addressing the *class imbalance problem*. Suppose a bank wants to develop a classifier that guards against fraudulent credit card transactions. Illustrate how you can induce a quality classifier based on a large set of legitimate examples and a very small set of fraudulent cases.
- 6.22.** XGBoost is a scalable machine learning system for tree boosting. Its objective function has a training loss and a regularization term:  $\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$ . Read the XGBoost paper and answer the following questions:
- What is  $\hat{y}_i$ ? At the  $t$ th iteration, XGBoost fixes  $f_1, \dots, f_{t-1}$  and trains the  $t$ th tree model  $f_t$ . How does XGBoost approximate the training loss  $l(y_i, \hat{y}_i)$  here?
  - What is  $\Omega(f_k)$ ? Which part in the regularization term needs to be considered at the  $t$ th iteration?

---

## 6.10 Bibliographic notes

Classification is a fundamental topic in machine learning, statistics, and pattern recognition. Many textbooks from these fields highlight classification methods such as Mitchell [Mit97]; Bishop [Bis06a]; Duda, Hart, and Stork [DHS01]; Theodoridis and Koutroumbas [TK08]; Hastie, Tibshirani, and Friedman [HTF09]; Alpaydin [Alp11]; Marsland [Mar09]; and Aggarwal [Agg15a].

For decision tree induction, the C4.5 algorithm is described in a book by Quinlan [Qui93]. The CART system is detailed in *Classification and Regression Trees* by Breiman, Friedman, Olshen, and Stone [BFOS84]. Both books give an excellent presentation of many of the issues regarding decision tree induction. C4.5 has a commercial successor, known as C5.0, which can be found at <http://www.rulequest.com>. ID3, a predecessor of C4.5, is detailed in Quinlan [Qui86]. It expands on pioneering work on concept learning systems, described by Hunt, Marin, and Stone [HMS66].

Other algorithms for decision tree induction include FACT (Loh and Vanichsetakul [LV88]), QUEST (Loh and Shih [LS97]), PUBLIC (Rastogi and Shim [RS98]), and CHAID (Kass [Kas80] and Magidson [Mag94]). INFERULE (Uthurusamy, Fayyad, and Spangler [UFS91]) learns decision trees from inconclusive data, where probabilistic rather than categorical classification rules are obtained. KATE (Manago and Kodratoff [MK91]) learns decision trees from complex structured data. Incremental versions of ID3 include ID4 (Schlimmer and Fisher [SF86]) and ID5 (Utgoff [Utg88]), the latter of which is extended in Utgoff, Berkman, and Clouse [UBC97]. An incremental version of CART is described in Crawford [Cra89]. BOAT (Gehrke, Ganti, Ramakrishnan, and Loh [GGRL99]), a decision tree algorithm that addresses the scalability issue in data mining, is also incremental. Other decision tree

algorithms that address scalability include SLIQ (Mehta, Agrawal, and Rissanen [MAR96]), SPRINT (Shafer, Agrawal, and Mehta [SAM96]), RainForest (Gehrke, Ramakrishnan, and Ganti [GRG98]), and earlier approaches, such as Catlet [Cat91] and Chan and Stolfo [CS93a,CS93b].

For a comprehensive survey of many salient issues relating to decision tree induction, such as attribute selection and pruning, see Murthy [Mur98]. Perception-based classification (PBC), a visual and interactive approach to decision tree construction, is presented in Ankerst, Elsen, Ester, and Kriegel [AEEK99].

For a detailed discussion on attribute selection measures, see Kononenko and Hong [KH97]. Information gain was proposed by Quinlan [Qui86] and is based on pioneering work on information theory by Shannon and Weaver [SW49]. The gain ratio, proposed as an extension to information gain, is described as part of C4.5 (Quinlan [Qui93]). The Gini impurity was proposed for CART by Breiman, Friedman, Olshen, and Stone [BFOS84]. The G-statistic, based on information theory, is given in Sokal and Rohlf [SR81]. Comparisons of attribute selection measures include Buntine and Niblett [BN92], Fayyad and Irani [FI92], Kononenko [Kon95], Loh and Shih [LS97], and Shih [Shi99]. Fayyad and Irani [FI92] show limitations of impurity-based measures, such as information gain and the Gini impurity. They propose a class of attribute selection measures called C-SEP (Class SEPARation), which outperform impurity-based measures in certain cases.

Kononenko [Kon95] notes that attribute selection measures based on the minimum description length principle have the least bias toward multivalued attributes. Martin and Hirschberg [MH95] proved that the time complexity of decision tree induction increases exponentially with respect to tree height in the worst case, and under fairly general conditions in the average case. Fayad and Irani [FI90] found that shallow decision trees tend to have many leaves and higher error rates for a large variety of domains. Attribute (or feature) construction is described in Liu and Motoda [LM98,Le98].

There are numerous algorithms for decision tree pruning, including cost complexity pruning (Breiman, Friedman, Olshen, and Stone [BFOS84]), reduced error pruning (Quinlan [Qui87]), and pessimistic pruning (Quinlan [Qui86]). PUBLIC (Rastogi and Shim [RS98]) integrates decision tree construction with tree pruning. MDL-based pruning methods can be found in Quinlan and Rivest [QR89]; Mehta, Agrawal, and Rissanen [MAR96]; and Rastogi and Shim [RS98]. Other methods include Niblett and Bratko [NB86] and Hosking, Pednault, and Sudan [HPS97]. For an empirical comparison of pruning methods, see Mingers [Min89] and Malerba, Floriana, and Semeraro [MFS95]. For a survey on simplifying decision trees, see Breslow and Aha [BA97].

Thorough presentations of Bayesian classification can be found in Duda, Hart, and Stork [DHS01], Weiss and Kulikowski [WK91], and Mitchell [Mit97]. For an analysis of the predictive power of naïve Bayesian classifiers when the class-conditional independence assumption is violated, see Domingos and Pazzani [DP96]. Experiments with kernel density estimation for continuous-valued attributes, rather than Gaussian estimation, have been reported for naïve Bayesian classifiers in John [Joh97].

Nearest-neighbor classifiers were introduced in 1951 by Fix and Hodges [FH51]. A comprehensive collection of articles on nearest-neighbor classification can be found in Dasarathy [Das91]. Additional references can be found in many texts on classification, such as Duda, Hart, and Stork [DHS01] and James [Jam85], as well as articles by Cover and Hart [CH67] and Fukunaga and Hummels [FH87]. Their integration with attribute weighting and the pruning of noisy instances is described in Aha [Aha92]. The use of search trees to improve nearest-neighbor classification time is detailed in Friedman, Bentley, and Finkel [FBF77]. The partial distance method was proposed by researchers in vector quantization and compression. It is outlined in Gersho and Gray [GG92]. The editing method for re-

moving “useless” training tuples was first proposed by Hart [Har68]. For speeding-up the computation of  $k$ -nearest neighbor based on locality sensitive hashing, see Pan and Manocha [PM12,PM11]; and Zhang, Huang, Geng and Liu [ZHGL13].

The computational complexity of nearest-neighbor classifiers is described in Preparata and Shamos [PS85]; and Haghani, Sebastian, and Karl [HMA09]. References on case-based reasoning include the texts by Riesbeck and Schank [RS89] and Kolodner [Kol93], as well as Leake [Lea96] and Aamodt and Plazas [AP94]. For a list of business applications, see Allen [All94]. Examples in medicine include CASEY by Koton [Kot88] and PROTOS by Bareiss, Porter, and Weir [BPW88], whereas Rissland and Ashley [RA87] is an example of CBR for law. CBR is available in several commercial software products.

Linear regression and its numerous variants, such as RIDGE regression, robust regression are covered in most statistics textbooks, such as Freedman [Fre09]; Draper and Smith [DS98]; and Fox [Fox97]. For LASSO, see Tibshirani [Tib11]. Perceptron was first invented by Rosenblatt [Ros58], and Novikoff [Nov63] analyzed its convergence property. The Perceptron is one of the earliest linear classifiers, proposed in 1958 by Rosenblatt [Ros58], which became a landmark in early machine learning history. In 1969, Minsky and Papert [MP69] showed that Perceptrons are incapable of learning concepts that are linearly inseparable. A general introduction to logistic regression can be found in most machine learning textbooks, such as Mitchell [Mit97]; Hastie, Tibshirani, and Friedman [HTF09]; and Aggarwal [Agg15a]. Ng and Jordan [NJ02] conducted a thorough comparison between naïve Bayesian classifier and logistic regression. The relationship between logistic regression and log-linear model can be found in Christensen [Chr06].

Issues involved in estimating classifier accuracy are described in Weiss and Kulikowski [WK91] and Witten and Frank [WF05]. Sensitivity, specificity, and precision are discussed in most information retrieval textbooks. For the  $F$  and  $F_\beta$  measures, see van Rijsbergen [vR90]. The use of stratified 10-fold cross-validation for estimating classifier accuracy is recommended over the holdout, cross-validation, leave-one-out (Stone [Sto74]), and bootstrapping (Efron and Tibshirani [ET93]) methods, based on a theoretical and empirical study by Kohavi [Koh95]. See Freedman, Pisani, and Purves [FPP07] for the confidence limits and statistical tests of significance.

For ROC analysis, see Egan [Ega75], Swets [Swe88], and Vuk and Curk [VC06]. Bagging is proposed in Breiman [Bre96]. Freund and Schapire [FS97] proposed AdaBoost. This boosting technique has been applied to several different classifiers, including decision tree induction (Quinlan [Qui96]) and naïve Bayesian classification (Elkan [Elk97]). Friedman [Fri01] proposed the gradient boosting machine. Chen and Guestrin designed a highly scalable system called Xgboost [CG16]. The ensemble technique of random forests is described by Breiman [Bre01]. Seni and Elder [SE10] proposed the Importance Sampling Learning Ensembles (ISLE) framework, which views bagging, AdaBoost, random forests, and gradient boosting as special cases of a generic ensemble generation procedure. There are many online software packages for ensemble routines, including bagging, AdaBoost, gradient boosting, and random forests. Studies on the class imbalance problem and/or cost-sensitive learning include Weiss [Wei04], Zhou and Liu [ZL06], Zapkowicz and Stephen [ZS02], Elkan [Elk01], Domingos [Dom99], and Huang, Li, Loy and Tang [HLLT16].

The University of California at Irvine (UCI) maintains a Machine Learning Repository of data sets for the development and testing of classification algorithms. It also maintains a Knowledge Discovery in Databases (KDD) Archive, an online repository of large data sets that encompasses a wide vari-

ety of data types, analysis tasks, and application areas. For information on these two repositories, see <http://www.ics.uci.edu/~mlearn/MLRepository.html> and <http://kdd.ics.uci.edu>.

No classification method is superior to all others for all data types and domains. Empirical comparisons of classification methods include Quinlan [Qui88]; Shavlik, Mooney, and Towell [SMT91]; Brown, Coruble, and Pittard [BCP93]; Curram and Mingers [CM94]; Michie, Spiegelhalter, and Taylor [MST94]; Brodley and Utgoff [BU95]; and Lim, Loh, and Shih [LLS00].



This page intentionally left blank

# Classification: advanced methods

# 7

**In this chapter, you will learn** advanced techniques for data classification. We start with **feature selection and engineering** (Section 7.1). Then, we will introduce **Bayesian belief networks** (Section 7.2), which unlike naïve Bayesian classifiers, do not assume class conditional independence. A powerful approach to classification known as support vector machines is presented in Section 7.3. A **support vector machine** transforms training data into a higher dimensional space, where it finds a hyperplane that separates the data by class using essential training tuples called *support vectors*. Section 7.4 describes **rule-based and pattern-based classification**. For the former, our classifier is in the form of a set of IF-THEN rules, whereas the latter explores relationships between attribute–value pairs that occur frequently in data. This methodology builds on research on frequent pattern mining (Chapters 4 and 5). Classification with weak supervision is introduced in Section 7.5. Section 7.6 introduces various techniques for classification on rich data types, such as stream data, sequence data, and graph data. Other related techniques to classification, such as multiclass classification, distance metric learning, interpretability of classification, reinforcement learning, and genetic algorithms are introduced in Section 7.7.

## 7.1 Feature selection and engineering

For the classification setting introduced in Chapter 6, in order to train a classifier (e.g., naïve Bayes Classifier,  $k$ -nearest-neighbor classifier), we assume that there exists a training set with  $n$  tuples, each of which is represented by  $p$  attributes or features. “*But, where do these  $p$  features come from at the first place?*” Let us consider two scenarios. In the first scenario (**Feature Selection**), you might have collected a large number of (say hundreds or thousands or even more) features. However, most of them might be irrelevant with respect to the classification task or redundant with each other. For example, in order to predict whether an online student will drop out before finishing the program, the student ID is an irrelevant feature. In another example of predicting whether a customer will buy a computer, one of the two features, namely yearly income and monthly income, is redundant since one (e.g., yearly income) can be inferred from the other (e.g., monthly income). Including such irrelevant or redundant features during the classifier training process will not help improve the classification accuracy, yet they are likely to make the trained classifier sensitive to noise, leading to degraded generalization performance. “*How can we select a subset of most relevant features from the initial  $p$  input features to train a classification model?*” This is the main focus of this section.

In the second scenario (**Feature Engineering**), you might wonder “*How can I construct  $p$  features so that all of them are critical for the classification task I have?*” or “*Given the initial  $p$  features, how can I transform them into another  $p'$  attributes so that these transformed features will be more*

*effective for the given classification task?*” These are the questions that *feature engineering* tries to answer. For example, in order to predict whether a regional disease outbreak will occur, one might have collected a large number of features from the health surveillance data, including the number of daily positive cases, the number of daily tests, and the number of daily hospitalization. It turns out a powerful indicator (or feature) to predict the disease outbreak is *weekly positive rate*. In this example, the weekly positive rate, which is the ratio of the number of positive cases and the number of tests of a week, can be constructed (or engineered) based on the initial features (e.g., daily positive cases, daily test cases). In practice, feature engineering plays a very important role in the performance of the classification model. Traditionally, feature engineering requires substantial domain knowledge. Some data transformation techniques (e.g., DWT, DFT, and PCA), that were introduced in Chapter 2 can be viewed as feature engineering methods. The deep learning techniques that we will introduce in Chapter 10 provide an automatic way for feature engineering, capable of generating powerful features from the initial input features. The engineered features are often semantically more meaningful with a significant classification performance improvement.

In this section, we will introduce three types of feature selection methods, namely **filter methods**, **wrapper methods**, and **embedded methods**. A filter method selects features based on some goodness measure that is independent of the specific classification model. A wrapper method combines the feature selection and classifier model construction steps together, and it iteratively uses the currently selected feature subset to construct a classification model, which is in turn used to update the selected feature subset. An embedded method simultaneously constructs the classification model and selects the relevant features. In other words, it *embeds* the feature selection step during the classification model construction step. Fig. 7.1 provides a pictorial comparison of these three methods.

Feature selection can be used for both classification and regression. It can also be applied to unsupervised data mining tasks, such as clustering. For both filter and wrapper methods, we will illustrate them with classification tasks. We will mainly use the linear regression task, which was introduced in Section 6.5, to explain the embedded methods.

### 7.1.1 Filter methods

A **filter method** selects “good” features based on a certain “goodness” measure of the input features. A filter method is independent of the specific classification model and is often used as a preprocessing step of other feature selection methods (e.g., wrapper or embedded methods). The idea is quite straightforward. Suppose we have  $p$  initial features and we wish to select  $k$  out of  $p$  features (where  $k < p$ ). If we have a goodness score for each feature, we can simply select  $k$  features with the highest goodness scores.

“*So, how shall we measure the goodness of a feature?*” Intuitively, we might say that a feature is good if it is highly *correlated* with the class label we want to predict. Suppose there are  $n$  training tuples. We wish to measure the correlation between a feature (i.e., attribute)  $x$  and the class label  $y$ . How can we measure the correlation between the given feature  $x$  and the class label  $y$ ? If the given feature  $x$  is a categorical attribute (e.g., job title), a natural choice is  $\chi^2$  test, which was introduced in Section 2.2.3. To be specific, a higher  $\chi^2$  value indicates a stronger correlation between the given feature  $x$  and the class label  $y$ . We select  $k$  features with the highest  $\chi^2$  values.

“*But, what if the given feature  $x$  is a continuous attribute (e.g., yearly income)?*” We have two choices. First, we can discretize the continuous attribute  $x$  into a categorical attribute (e.g., high,

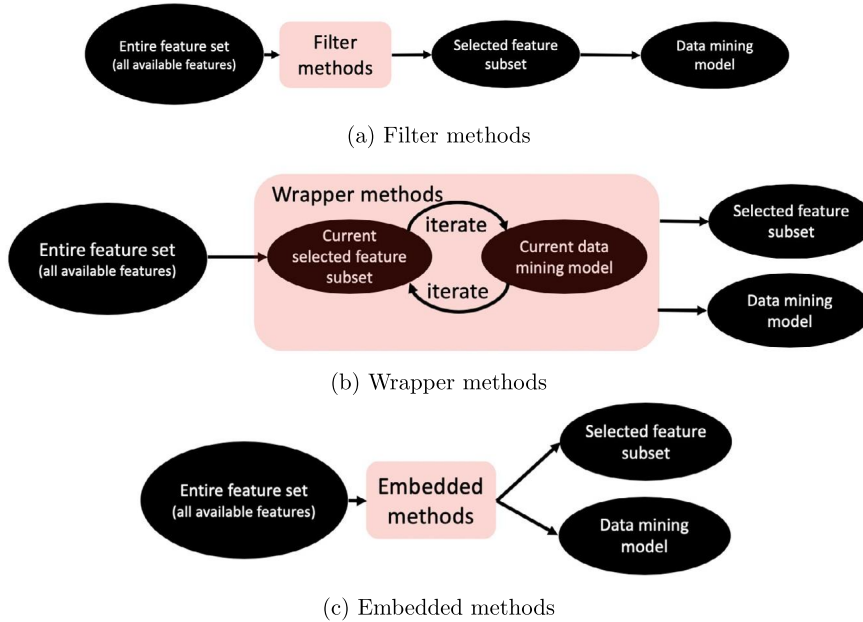


FIGURE 7.1

An overview of three feature selection methods.

medium vs. low income) and then use the  $\chi^2$  test to measure the correlation between the discretized attribute and the class label to select  $k$  most correlated features. Second, we can resort to **Fisher score** to directly measure the correlation between a continuous variable (the given feature  $x$ ) and a categorical variable (the class label  $y$ ).

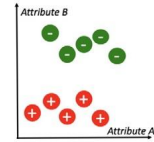
Suppose we have a binary class label  $y$  (i.e., whether or not the customer will buy a computer). Intuitively, the feature  $x$  (e.g., income) is strongly correlated with the class label  $y$  if (1) the average income of all customers who buy a computer is significantly different from the average income of all customers who do not buy a computer, (2) all customers who buy a computer share similar income, and (3) all customers who do not buy a computer share similar income. Formally, Fisher score is defined as follows:

$$s = \frac{\sum_{j=1}^c n_j (\mu_j - \mu)^2}{\sum_{j=1}^c n_j \sigma_j^2}, \quad (7.1)$$

where  $c$  is the total number of classes ( $c = 2$  in our example),  $n_j$  is the number of training tuples in class  $j$ ,  $\mu_j$  and  $\sigma_j^2$  are the mean value and variance of feature  $x$  among all tuples that belong to class  $j$ , respectively, and  $\mu$  is the mean value of feature  $x$  among all training tuples. Therefore a feature  $x$  would have a high Fisher score if the following conditions hold. First, the *class-specific mean* values  $\mu_j (j = 1, \dots, c)$  are dramatically different from each other (e.g., a large numerator of the Fisher score in Eq. (7.1)). Intuitively, this implies that on average, the feature values from different classes are quite different from each other. Second, the *class-specific variance*  $\sigma_j^2$  is small (e.g., a small denominator of

tuple index	1	2	3	4	5	6	7	8	9	10
Attribute A	1	2	3	4	5	2	3	4	5	6
Attribute B	1.0	0.9	0.8	1.2	1.1	5.1	4.8	4.9	5.2	5.0
Class label	+	+	+	+	+	-	-	-	-	-

(a) Training tuples



(b) Scatter-plot

**FIGURE 7.2**

Feature selection by Fisher score. (a) Ten training tuples, each of which is represented by two attributes (attribute A and attribute B) and a binary class label (+ vs. -). (b) Scatter-plot of training tuples. Intuitively, attribute B better separates the positive training tuples from negative ones than attribute A. This is consistent with Fisher scores:  $s(\text{attribute B}) = 200 > s(\text{attribute A}) = 0.125$ .

the Fisher score in Eq. (7.1)). This indicates that, within a class, different training tuples share similar feature values.

**Example 7.1.** We are given 10 training tuples in Fig. 7.2(a), each of which is represented by two attributes (attribute A and attribute B) and a binary class label (+ vs. -). We want to use Fisher scores to decide which attribute is more correlated with the class label. There are five positive tuples and five negative tuples  $n_1 = n_2 = 5$ . For attribute A, the mean value among all training tuples  $\mu = (1 + 2 + 3 + 4 + 5 + 2 + 3 + 4 + 5 + 6)/10 = 3.5$ , the mean value among positive training tuples  $\mu_1 = (1 + 2 + 3 + 4 + 5)/5 = 3$ , the mean value among negative training tuples  $\mu_2 = (2 + 3 + 4 + 5 + 6)/5 = 4$ , the variance of the positive tuples  $\sigma_1^2 = ((1 - 3)^2 + (2 - 3)^2 + (3 - 3)^2 + (4 - 3)^2 + (5 - 3)^2)/5 = 2$ , and the variance of the negative tuples  $\sigma_2^2 = ((2 - 4)^2 + (3 - 4)^2 + (4 - 4)^2 + (5 - 4)^2 + (6 - 4)^2)/5 = 2$ . Therefore the Fisher score for attribute A is  $s(\text{attribute A}) = (5 \times (3 - 3.5)^2 + 5 \times (4 - 3.5)^2)/(5 \times 2 + 5 \times 2) = 0.125$ . We compute the Fisher score for attribute B in a similar way and have that  $s(\text{attribute B}) = 200$ . According to Fisher scores, attribute B is more correlated with the class label than attribute A. This is consistent with the scatter-plot in Fig. 7.2(b), where the positive tuples are well separated from negative tuples along the vertical axis (attribute B), whereas they are mixed together along the horizontal axis (attribute A).  $\square$

In addition to correlation measures (e.g.,  $\chi^2$  test for categorical feature, Fisher score for continuous feature), we might say that a feature  $x$  is good if it contains “a lot of information” about the class label  $y$  that we want to predict. This suggests **information-theoretic goodness measures** for feature selection. For example, we can use *information gain* as the goodness measure for feature selection. The information gain, entropy, and conditional entropy were introduced in Section 6.2. In a nutshell, let  $H(y)$  be the entropy of the class label  $y$  and  $H(y|x)$  be the conditional entropy of the class label  $y$  given the feature  $x$ . The information gain of the feature  $x$  is defined as the difference between  $H(y)$  and  $H(y|x)$ . The intuition is that a feature  $x$  with a larger information gain can better reduce the impurity (e.g., entropy) of the class label  $y$ . Thus it contains “more information” about predicting the class label  $y$ . In addition to information gain, another commonly used information theoretic goodness measure for feature selection is *mutual information (MI)*. Intuitively, the mutual information between a feature  $x$  and the class label measures how much information the feature  $x$  provides to make the correct prediction of the class label  $y$ . Therefore features with the largest mutual information should be selected. The details about how to compute mutual information can be found in Appendix A.

With filter methods, the general process of training a classification model is as follows (Fig. 7.1(a)). Given a set of  $p$  initial features, we first use a filter method to select  $k$  features (e.g.,  $k$  out of  $p$  features with the highest Fisher scores). Then, using these  $k$  selected features, we build a classifier (e.g., a logistic regression classifier). Finally, we evaluate the performance of the trained classifier, such as cross-validation accuracy. Notice that during the feature selection process, a filter method is *independent* of the specific classification model that will be trained with the selected features. Another potential limitation with a filter method is that it does not consider the interaction between different features, and thus might select *redundant* features.

### 7.1.2 Wrapper methods

A **wrapper method** adopts a different strategy for feature selection by combining the feature selection step and classifier training step together. A wrapper method is often an iterative process (Fig. 7.1(b)). At each iteration, it tries to build a classifier based on the *currently* selected feature subset, and then based on the built classifier, it updates (e.g., add, remove, swap) the selected feature subset. In other words, it *wraps* the feature selection and classifier training together, hence the name of wrapper.

The most important component of a wrapper method is how to search for the best feature subset. A straightforward way (i.e., exhaustive search) is to try all the possible subsets of the  $p$  given features. We use each subset of the feature to build a classification model and evaluate its performance, such as the classification accuracy using either the held-out set or cross-validation. The best feature subset is the one with the highest classification accuracy for the given classification model. This strategy is optimal since it finds the best feature subset with the highest classification accuracy. However, it is very expensive in terms of computation, since it needs to search and evaluate all  $(2^p - 1)$  possible subsets of the  $p$  given features—an exponential number!

In practice, a wrapper method often relies on some heuristic search strategy to avoid the  $(2^p - 1)$  exponential search space. Section 2.6.2 introduced different attribute subset selection strategies, which can be applied here. For example, in the *stepwise forward selection* method, it starts with an empty feature subset. At each iteration of the feature selection process, it selects an additional feature, which, when added into the current feature subset, will improve the classification model performance most (e.g., classification accuracy measured by the held-out method or cross-validation). The process will terminate when adding the extra features can no longer improve the classification model performance. In contrast, in the *stepwise backward elimination* method, it starts with all the  $p$  initial features, and then iteratively eliminates features from the current subset whose removal would increase the classification accuracy most. We can also combine these two strategies together. That is, at each iteration, we try to select one additional feature and meanwhile might eliminate one existing feature that will improve the classification accuracy most. In addition to these three typical search strategies, some wrapper methods leverage more sophisticated techniques, such as simulated annealing and genetic algorithm. Simulated annealing is a probabilistic optimization technique, often designed for complex (e.g., nonconvex) optimization problems. The latter will be introduced in Section 7.7.

By “wrapping” the feature selection and the classification model construction steps together, a wrapper method tends to have better performance than filtering methods. However, since it needs to iteratively search for the feature subset and (re-)train the classification model, the computational cost of a wrapper method is usually much more intense than filter methods. How can we simultaneously enjoy the advantages of both filter methods and wrapper methods? That is what embedded methods try to answer.

### 7.1.3 Embedded methods

Embedded methods aim to combine the advantages of both filter methods and wrapper methods. On the one hand, an embedded method performs feature selection and classification model construction simultaneously, so that the two can mutually benefit from each other. On the other hand, an embedded method tries to avoid the expensive, iterative search process in wrapper methods.

We actually have already seen an embedded method in Chapter 6! For decision tree induction that was introduced in Section 6.2, it is possible that only a fraction of all  $d$  initial attributes are present in the built decision tree model. For the example in Fig. 6.2, only three attributes (i.e., age, student, credit\_rating) are present in the decision tree model, albeit there might be tens of or hundreds of initial attributes. This could happen if the decision tree induction algorithm terminates before it exhausts all  $d$  initial attributes, or some attributes of the initially built decision tree are removed during tree pruning process. In either case, all the attributes that appear on the nonleaf tree nodes can be viewed as the selected feature subset, and decision tree induction itself can be viewed as an embedded method for feature selection. In other words, the feature selection process (i.e., to decide which attribute(s) are used as the nonleaf tree nodes) is *embedded* in the decision tree induction process. We simultaneously accomplish both the feature selection step and the classification model construction (i.e., decision tree induction) step. This is the essence of an embedded method.

Other powerful embedded methods often rely on a technique called *sparse learning*. Let us first introduce its high-level idea, and then we will explain the details using linear regression as an example. A handful of data mining models can be solved from the optimization perspective, such as the linear regression model and logistic regression. In a nutshell, we build these data mining models by minimizing some objective (or loss) function that directly or indirectly measures the performance of the corresponding data mining model. For example, in least square linear regression, we find the optimal weight vector  $w$  by minimizing the sum of the squared difference between the predicted output and the actual output; in logistic regression, we find the optimal weight vector  $w$  by minimizing the negative log likelihood. Now, let us modify the objective function so that it also “penalizes” the number of the features it uses. By minimizing the modified objective function, the trained data mining model might only use a subset of all the  $d$  initial features and thus accomplish the task of feature selection. In this way, we will be able to embed the feature selection process (by penalizing the number of features used in the final model) in the model training process.

“So, how can we penalize the number of features used in a data mining model, and how can we solve the modified optimization problem accordingly?” Let us explain the details using least square linear regression (which was introduced in Section 6.5) as an example. Recall that a multilinear regression model assumes  $\hat{y}_i = w^T x_i = w_0 + w_1 x_{i,1} + \dots + w_d x_{i,d}$ , where  $\hat{y}_i$  is the predicted output for the  $i$ th tuple,  $x_i = (1, x_{i,1}, \dots, x_{i,d})$  is the attribute (feature) vector of the  $i$ th tuple, and  $w = (w_0, w_1, \dots, w_d)$  is the weight vector. We find the optimal weight vector  $w$  by minimizing the loss function  $L(w) = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{2} \sum_{i=1}^n (y_i - w^T x_i)^2$ , which measures the sum of the squared difference between the predicted output ( $\hat{y}_i$ ) and the actual output ( $y_i$ ). “How can we ‘embed’ the feature selection in the process of training such a linear regression model?” For the  $j$ th feature, if the corresponding weight  $w_j = 0$ , then it has no contribution on the linear regression model. In other words, this feature is “unselected.” This naturally suggests that we can use the  $l_0$  norm<sup>1</sup> of the weight vector  $w$ , which

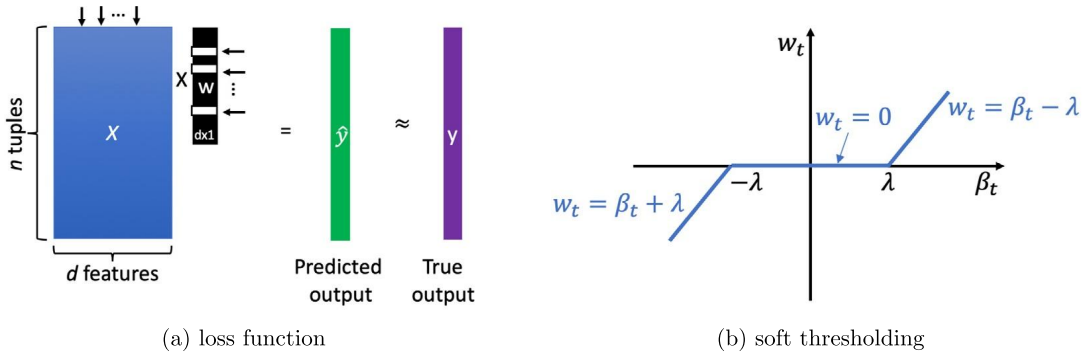
<sup>1</sup>  $l_0$  is a special case of the  $l_p$  norm when  $p$  approaches 0.  $l_p$  norm was introduced in Chapter 2.

counts the number of nonzero elements in the weight vector  $w$ , to measure how many features are used (i.e., selected) in the trained linear regression model. Therefore if we train a linear regression model by minimizing the following modified loss function  $\tilde{L}(w) = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \|w\|_0 = \frac{1}{2} \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_0$ , the optimal weight vector  $w$  is likely to contain some zero elements. Those features whose corresponding weights in vector  $w$  are nonzero are selected. The parameter  $\lambda > 0$  balances two terms in the modified loss function. Generally speaking, the larger the  $\lambda$ , the less number of the features are likely to be selected (i.e., more elements in the weight vector  $w$  are likely to be zeros).

However, finding the optimal weight vector  $w$  that minimizes the modified loss function  $\tilde{L}(w)$  is very hard. This is because the  $l_0$  norm of the weight vector  $w$ , which tells how many features are selected, is *nonconvex*. To address this issue, we replace the  $l_0$  norm by another norm that is convex. It turns out the  $l_1$  norm  $\|w\|_1 = \sum_{j=0}^d |w_j|$  is the best convex approximation of the  $l_0$  norm, where  $|w_j|$  is the absolute value of  $w_j$ . Thus we have a new loss function as follows.

$$\hat{L}(w) = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \|w\|_1 = \frac{1}{2} \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \sum_{j=0}^d |w_j| \quad (7.2)$$

The regression model that minimizes the new loss function  $\hat{L}(w)$  in Eq. (7.2) is called **LASSO**, which stands for Least Absolute Shrinkage and Selection Operator. The optimal weight vector  $w$  of  $\hat{L}$  is often *sparse*, meaning that some of its elements might be zeros. The nonzero elements of the optimal vector  $w$  tell that the corresponding features are selected by the linear regression model. Fig. 7.3(a) presents an illustration of the loss function of LASSO (Eq. (7.2)).



**FIGURE 7.3**

An illustration of LASSO. (a) Illustration of the loss function of LASSO (Eq. (7.2)). The training set is represented by an  $n \times d$  feature matrix  $X$  whose rows are tuples and columns are features, and an  $n \times 1$  output vector  $y$ . By minimizing the sum of the squared difference between the actual and predicted output (i.e., the first term of Eq. (7.2)), the trained linear regression models try to make the predicted output  $\hat{y}$  to be as close as possible to the actual output  $y$ . By minimizing the  $l_1$  norm of the weight vector  $w$  (i.e., the second term of Eq. (7.2)), some elements of the weight vector  $w$  are zeros (indicated by the arrows), and the corresponding features (the columns of the feature matrix  $X$ ) are “unselected.” (b) Soft thresholding pushes the coefficients with small magnitudes (between  $-\lambda$  and  $\lambda$ ) to be zeros while shrinking the remaining coefficients by  $\lambda$  in magnitude.



“So, how can we find the optimal weight vector  $w$  that minimizes  $\hat{L}$  in Eq. (7.2)?” The good news is that unlike function  $\tilde{L}$  that is nonconvex, the loss function  $\hat{L}$  in Eq. (7.2) is a convex function. There exist many numerical optimization packages that can be used to solve it. Here, we introduce one of them, called *coordinate descent*.

Unlike the least square regression that has a closed-form solution, the closed-form solution for LASSO does not exist. Coordinate descent finds the optimal weight vector  $w$  in an iterative way, and it works as follows. First, we initialize the weight vector  $w$ . (We can simply set each element in  $w$  as zero.) Then, the algorithm iterates until it converges or some stopping criterion is met, for example, a maximum iteration number has been reached. At each iteration, the algorithm tries to update each element in the weight vector  $w$  one-by-one, while fixing all the remaining elements in  $w$ . Therefore it boils down to the following question. “How can we update a single element (say  $w_t$  ( $0 \leq t \leq d$ )) while fixing all other elements?” We take the following three steps. First, we compute the residual for each training tuple  $r_i = y_i - \sum_{j=0, j \neq t}^d w_j x_{i,j}$ . The intuition of the residual  $r_i$  is that it measures the prediction error for the  $i$ th tuple if we use all but the  $t$ th features. Second, we train a least square regression model for all the input tuples, where each tuple is represented by a single input feature  $x_{i,t}$  and its output is the residual  $r_i$ . The weight (i.e., coefficient) for the  $t$ th feature from such a least square regression model is represented as  $\beta_t$ . (Recall that we can use the closed-form solution of least square regression to find the coefficient  $\beta_t$ , which was introduced in Section 6.5.) The intuition is that if we fix all but the  $t$ th features, the coefficient  $\beta_t$  is the best coefficient that minimizes the overall least square prediction error. Third, we update the weight  $w_t$  as follows.

$$w_t = \begin{cases} \beta_t - \lambda & \text{if } \beta_t \geq \lambda \\ \beta_t + \lambda & \text{if } \beta_t \leq -\lambda \\ 0 & \text{otherwise} \end{cases} \quad (7.3)$$

The third step is called *soft thresholding*, and its intuition is as follows. If  $|\beta_t|$  is greater than the regularization parameter  $\lambda$ , the soft thresholding step would reduce the magnitude of  $\beta_t$  by  $\lambda$ , which is used as the updated coefficient  $w_t$ ; otherwise the coefficient  $w_t$  is simply set as zeros. In other words, the soft thresholding pushes the coefficients with small magnitude as zero while shrinking the remaining coefficients. In this way, the final weight vector  $w$  is likely to be sparse with many zero elements, and thus achieves the purpose of feature selection. Fig. 7.3(b) presents an illustration of soft thresholding.

An earlier method for solving LASSO is called **LAR**, which stands for least angle regression. Recall that for linear regression introduced in Section 6.5, we could add the squared  $l_2$  norm of the weight vector into the loss function to prevent overfitting (i.e., Ridge regression). We can add both  $l_1$  norm and the squared  $l_2$  norm into the loss function  $L$ . Such a regression model is called **Elastic net**, and the features selected by Elastic net tend to be less correlated with each other, compared with LASSO. We can use a very similar idea as LASSO to embed the feature selection process in the classification model. For example, we can introduce an  $l_1$  norm regularization term in the objective function of logistic regression, so that the weight vector of the trained logistic regression classifier is likely to be sparse. In other words, it only uses a few selected features.

## 7.2 Bayesian belief networks

Chapter 6 introduced Bayes' theorem and naïve Bayesian classification. In this chapter, we describe *Bayesian belief networks*—probabilistic graphical models, which unlike naïve Bayesian classifiers allow the representation of dependencies among subsets of attributes. Bayesian belief networks can be used for classification. Section 7.2.1 introduces the basic concepts of Bayesian belief networks. In Section 7.2.2, you will learn how to train such models.

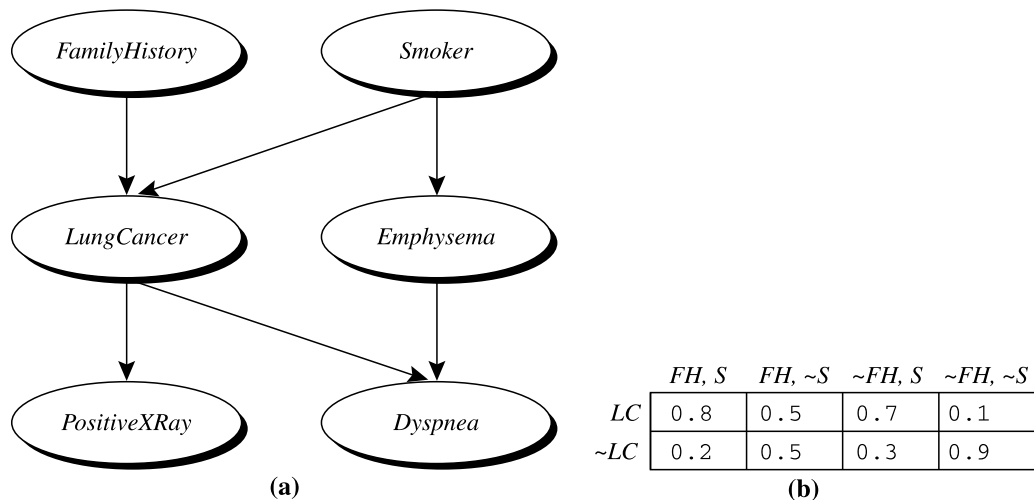
### 7.2.1 Concepts and mechanisms

The naïve Bayesian classifier makes the assumption of class conditional independence, that is, given the class label of a tuple, the values of the attributes are assumed to be conditionally independent of one another. The benefit of such an assumption is that it significantly simplifies computation. When the assumption holds true, the naïve Bayesian classifier is the most accurate in comparison with all other classifiers. In practice, however, dependencies can exist between variables (i.e., attributes). **Bayesian belief networks** specify joint probability distributions. They allow class conditional independence to be defined between subsets of variables. They provide a graphical model of causal relationships, on which learning can be performed. Trained Bayesian belief networks can be used for classification. Bayesian belief networks are also known as **belief networks**, **Bayesian networks**, and **probabilistic networks**. For brevity, we will refer to them as belief networks.

A belief network is defined by two components—a *directed acyclic graph* and a set of *conditional probability tables* (Fig. 7.4). Each node in the directed acyclic graph represents a random variable. The variables may be discrete- or continuous-valued. They may correspond to actual attributes given in the data or to “hidden variables” believed to form a relationship (e.g., in the case of medical data, a hidden variable may indicate a syndrome, representing a number of symptoms that, together, characterize a specific disease). Each arc represents a probabilistic dependence. If an arc is drawn from a node  $Y$  to a node  $Z$ , then  $Y$  is a **parent** or **immediate predecessor** of  $Z$ , and  $Z$  is a **descendant** of  $Y$ . *Each variable is conditionally independent of its nondescendants in the graph, given its parents.*

Fig. 7.4 is a simple belief network, adapted from Russell, Binder, Koller, and Kanazawa [RBKK95], for six Boolean variables. The arcs in Fig. 7.4(a) allow a representation of causal knowledge. For example, having lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. Note that the variable *PositiveXRay* is independent of whether the patient has a family history of lung cancer or is a smoker, given that we know the patient has lung cancer. In other words, once we know the outcome of the variable *LungCancer*, then the variables *FamilyHistory* and *Smoker* do not provide any additional information regarding *PositiveXRay*. The arcs also show that the variable *LungCancer* is conditionally independent of *Emphysema*, given its parents, *FamilyHistory* and *Smoker*. On the other hand, we cannot say that *LungCancer* is conditionally independent of *Dyspnea*, given its parents. Why? This is because *Dyspnea* is a child of *LungCancer* in the belief network.

A belief network has one **conditional probability table (CPT)** for each variable. The CPT for a variable  $Y$  specifies the conditional distribution  $P(Y|Parents(Y))$ , where  $Parents(Y)$  are the parents of  $Y$ . Fig. 7.4(b) shows a CPT for the variable *LungCancer*. The conditional probability for each known value of *LungCancer* is given for each possible combination of the values of its parents. For instance,

**FIGURE 7.4**

Simple Bayesian belief network. (a) A proposed causal model, represented by a directed acyclic graph. (b) The conditional probability table for the values of the variable *LungCancer* (*LC*) showing each possible combination of the values of its parent nodes, *FamilyHistory* (*FH*) and *Smoker* (*S*). Source: Adapted from Russell, Binder, Koller, and Kanazawa [RBKK95].

from the upper leftmost and bottom rightmost entries, respectively, we see that

$$P(\text{LungCancer} = \text{yes} \mid \text{FamilyHistory} = \text{yes}, \text{Smoker} = \text{yes}) = 0.8$$

$$P(\text{LungCancer} = \text{no} \mid \text{FamilyHistory} = \text{no}, \text{Smoker} = \text{no}) = 0.9.$$

Let  $X = (x_1, \dots, x_n)$  be a data tuple described by the variables or attributes  $Y_1, \dots, Y_n$ , respectively. Recall that each variable is conditionally independent of its nondescendants, given its parents. This allows the belief network to provide a complete representation of the joint probability distribution by the following equation:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{Parents}(Y_i)), \quad (7.4)$$

where  $P(x_1, \dots, x_n)$  is the probability of a particular combination of values of  $X$ , and the values for  $P(x_i \mid \text{Parents}(Y_i))$  correspond to the entries in the CPT for attribute  $Y_i$ .

A node within the belief network can be selected as an “output” node, representing a class label attribute. There may be more than one output node. Various algorithms for inference and learning can be applied to the network. Rather than returning a single class label, the classification process can return a probability distribution that gives the probability of each class. Belief networks can be used to answer probability of evidence queries (e.g., what is the probability that an individual will have *LungCancer*, given that they have both *PositiveXRay* and *Dyspnea*?) and most probable explanation queries (e.g., which group of the population is most likely to have both *PositiveXRay* and *Dyspnea*?).

Belief networks have been used to model a number of well-known problems. One example is genetic linkage analysis (e.g., the mapping of genes onto a chromosome). By casting the gene linkage problem in terms of inference on Bayesian networks, and using efficient algorithms, the scalability of such analysis has advanced considerably. Other applications that have benefited from the use of belief networks include computer vision (e.g., image restoration and stereo vision), document and text analysis, decision-support systems, financial fraud detection, and sensitivity analysis. The ease with which many applications can be reduced to Bayesian network inference is advantageous in that it curbs the need to invent specialized algorithms for each such application.

### 7.2.2 Training Bayesian belief networks

“How does a Bayesian belief network learn?” In the learning or training of a belief network, a number of scenarios are possible. The network **topology** (or “layout” of nodes and arcs) may be constructed by human experts or inferred from the data. The network variables may be *observable* or *hidden* in all or some of the training tuples. The hidden data case is also referred to as *missing values* or *incomplete data*.

Several algorithms exist for learning the network topology from the training data given observable variables. The problem is one of discrete optimization. For solutions, please see the bibliographic notes at the end of this chapter. Human experts usually have a good grasp of the direct conditional dependencies that hold in the domain under analysis, which helps in network design. Experts must specify conditional probabilities for the nodes that participate in direct dependencies. These probabilities can then be used to compute the remaining probability values.

If the network topology is known and the variables are observable, then training the network is straightforward. It consists of computing the CPT entries, as is similarly done when computing the probabilities involved in naïve Bayesian classification.

When the network topology is given and some of the variables are hidden, there are various methods to choose from for training the belief network. We will describe an effective method based on *gradient descent*, which was also used to train a logistic regression classifier in Chapter 6. For those without an advanced math background, the description of a gradient descent method may look rather intimidating with its calculus-packed formulae. However, packaged software exists to solve these equations. Let us recap the general idea behind a gradient descent method.

Let  $D$  be a training set of data tuples,  $X_1, X_2, \dots, X_{|D|}$ . Training the belief network means that we must learn the values of the CPT entries. Let  $w_{ijk}$  be a CPT entry for the variable  $Y_i = y_{ij}$  having the parents  $U_i = u_{ik}$ , where  $w_{ijk} \equiv P(Y_i = y_{ij} | U_i = u_{ik})$ . For example, if  $w_{ijk}$  is the upper leftmost CPT entry of Fig. 7.4(b), then  $Y_i$  is *LungCancer*;  $y_{ij}$  is its value, “yes”;  $U_i$  lists the parent nodes of  $Y_i$ , namely,  $\{FamilyHistory, Smoker\}$ ; and  $u_{ik}$  lists the values of the parent nodes, namely,  $\{“yes,” “yes”\}$ . The  $w_{ijk}$  are viewed as weights, analogous to the weights in logistic regression. The set of weights is collectively referred to as  $W$ . The weights are initialized to random probability values. A *gradient descent* strategy performs greedy hill-descending. At each iteration, the weights are updated and will eventually converge to a local optimum solution.

A **gradient descent** strategy is used to search for the optimal values of certain variables that best minimize an objective function, based on the assumption that each of the possible values is equally likely. Such a strategy is iterative. It searches for a solution along the negative of the gradient (i.e., steepest descent) of an objective function. In our setting, we want to find the set of weights,  $W$ , that

maximize an objective function.<sup>2</sup> To start with, the weights are initialized to random probability values. The gradient ascent method performs greedy hill-climbing in that, at each iteration or step along the way, the algorithm moves toward what appears to be the best solution at the moment, without backtracking. The weights are updated at each iteration. Eventually, they converge to a local optimum solution.

For our problem, we maximize the objective function  $P_w(D) = \prod_{d=1}^{|D|} P_w(\mathbf{X}_d)$ . This can be done by following the gradient of  $\ln P_w(D)$ , which makes the problem simpler. (Recall that we have used the same trick to train a logistic regression classifier in Chapter 6.) Given the network topology and initialized  $w_{ijk}$ , the algorithm proceeds as follows:

**1. Compute the gradients:** For each  $i, j, k$ , compute

$$\frac{\partial \ln P_w(D)}{\partial w_{ijk}} = \sum_{d=1}^{|D|} \frac{\partial \ln(P(Y_i = y_{ij}, U_i = u_{ik} | \mathbf{X}_d))}{\partial w_{ijk}}. \quad (7.5)$$

The probability on the right side of Eq. (7.5) is to be calculated for each training tuple,  $\mathbf{X}_d$ , in  $D$ . For brevity, let's refer to this probability simply as  $p$ . When the variables represented by  $Y_i$  and  $U_i$  are hidden for some  $\mathbf{X}_d$ , the corresponding probability  $p$  can be computed from the observed variables of the tuple using standard algorithms for Bayesian network inference such as those available in the commercial software package HUGIN (<http://www.hugin.dk>).

**2. Take a small step in the direction of the gradient:** The weights are updated by

$$w_{ijk} \leftarrow w_{ijk} + \eta \frac{\partial \ln P_w(D)}{\partial w_{ijk}}, \quad (7.6)$$

where  $\eta$  is the **learning rate** representing the step size, and  $\frac{\partial \ln P_w(D)}{\partial w_{ijk}}$  is computed from Eq. (7.5). The learning rate is set to a small constant and helps with convergence.

**3. Renormalize the weights:** Because the weights  $w_{ijk}$  are probability values, they must be between 0.0 and 1.0, and  $\sum_j w_{ijk}$  must equal 1 for all  $i, k$ . These criteria are achieved by renormalizing the weights after they have been updated by Eq. (7.6).

Algorithms that follow this learning form are called *adaptive probabilistic networks*. Other methods for training belief networks are referenced in the bibliographic notes at the end of this chapter. Belief networks could be computationally intensive. Because belief networks provide explicit representations of causal structure, a human expert can provide prior knowledge to the training process in the form of network topology or conditional probability values. This can significantly improve the learning speed.

---

### 7.3 Support vector machines

In this section, we study **support vector machines (SVMs)**, a method for the classification of both linear and nonlinear data. In a nutshell, an SVM is an algorithm that works as follows. It uses a nonlinear

---

<sup>2</sup> In order to apply gradient descent strategy to maximize, instead of minimize, an objective function, we actually do gradient *ascent* where we update the current solution along the direction of the gradient (i.e., gradient ascent).

mapping to transform the original training data into a higher-dimensional space. Within this new space, it searches for the linear optimal separating hyperplane (i.e., a “decision boundary” separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high-dimensional space, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using *support vectors* (“essential” training tuples) and *margins* (defined by the support vectors). We will delve more into these new concepts later.

“I’ve heard that SVMs have attracted a great deal of attention lately. Why?” The first paper on support vector machines was presented in 1992 by Vladimir Vapnik and colleagues Bernhard Boser and Isabelle Guyon, even though the groundwork for SVMs has been around since the 1960s (including early work by Vapnik and Alexei Chervonenkis on statistical learning theory). Although the training of even the fastest SVMs could be slow, they are highly accurate, owing to their ability to model complex nonlinear decision boundaries. They are much less prone to overfitting than other methods. The support vectors also provide a compact description of the learned model. SVMs can be used for numeric prediction and classification. They have been applied to a number of areas, including handwritten digit recognition, object recognition, emotion recognition, and speaker identification, as well as benchmark time-series prediction tasks.

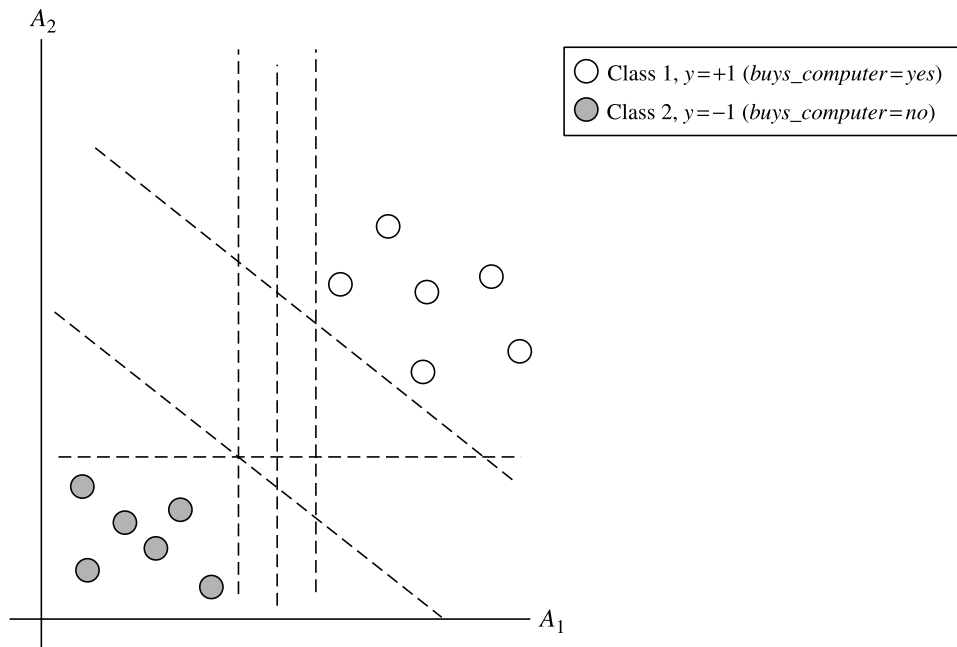
### 7.3.1 Linear support vector machines

To explain the mystery of SVMs, let’s first look at the simplest case—a two-class problem where the classes are linearly separable. Let the data set  $D$  be given as  $(X_1, y_1), (X_2, y_2), \dots, (X_{|D|}, y_{|D|})$ , where  $X_i$  is the set of training tuples with associated class labels,  $y_i$ . Each  $y_i$  can take one of two values, either  $+1$  or  $-1$  (i.e.,  $y_i \in \{+1, -1\}$ ), corresponding to the classes *buys\_computer = yes* and *buys\_computer = no*, respectively. To aid in visualization, let’s consider an example based on two input attributes,  $A_1$  and  $A_2$ , as shown in Fig. 7.5. From the graph, we see that the 2-D data are **linearly separable** (or “linear” for short), because a straight line can be drawn to separate all the tuples of class  $+1$  from all the tuples of class  $-1$ .

There are an infinite number of separating lines that could be drawn. We want to find the “best” one, that is, one that (we hope) will have the minimum classification error on previously unseen tuples. How can we find this best line? Note that if our data were 3-D (i.e., with three attributes), we would want to find the best separating *plane*. Generalizing to  $n$  dimensions, we want to find the best *hyperplane*. We will use “hyperplane” to refer to the decision boundary that we are seeking, regardless of the number of input attributes.

An SVM approaches this problem by searching for the **maximum margin hyperplane**. Consider Fig. 7.6, which shows two possible separating hyperplanes and their associated margins. Before we get into the definition of margins, let’s take an intuitive look at this figure. Both hyperplanes can correctly classify all the given data tuples. Intuitively, however, we expect the hyperplane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin. This is why (during the learning or training phase) the SVM searches for the hyperplane with the largest margin, that is, the *maximum marginal hyperplane* (MMH). The associated margin gives the largest separation between classes.

Getting to an informal definition of **margin**, we can say that the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyperplane to the other side of its margin, where the “sides” of the margin are parallel to the hyperplane. When dealing with the MMH, this distance is, in fact, the shortest distance from the MMH to the closest training tuple of either class.



**FIGURE 7.5**

The 2-D training data that are linearly separable. There are an infinite number of possible separating hyperplanes or “decision boundaries,” some of which are shown here as dashed lines. Which one is best?

A separating hyperplane is essentially a linear classifier. Similar to other linear classifiers (such as perceptron, logistic regression) that were introduced in Chapter 7, it can be written as

$$\mathbf{W} \cdot \mathbf{X} + b = 0, \quad (7.7)$$

where  $\mathbf{W}$  is a weight vector, namely,  $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ ;  $n$  is the number of attributes; and  $b$  is a scalar, often referred to as a bias. To aid in visualization, let’s consider two input attributes,  $A_1$  and  $A_2$ , as in Fig. 7.6(b). Training tuples are 2-D (e.g.,  $\mathbf{X} = (x_1, x_2)$ ), where  $x_1$  and  $x_2$  are the values of attributes  $A_1$  and  $A_2$ , respectively, for  $\mathbf{X}$ . Eq. (7.7) can be written as

$$b + w_1x_1 + w_2x_2 = 0. \quad (7.8)$$

Thus any point that lies above the separating hyperplane satisfies

$$b + w_1x_1 + w_2x_2 > 0. \quad (7.9)$$

Similarly, any point that lies below the separating hyperplane satisfies

$$b + w_1x_1 + w_2x_2 < 0. \quad (7.10)$$

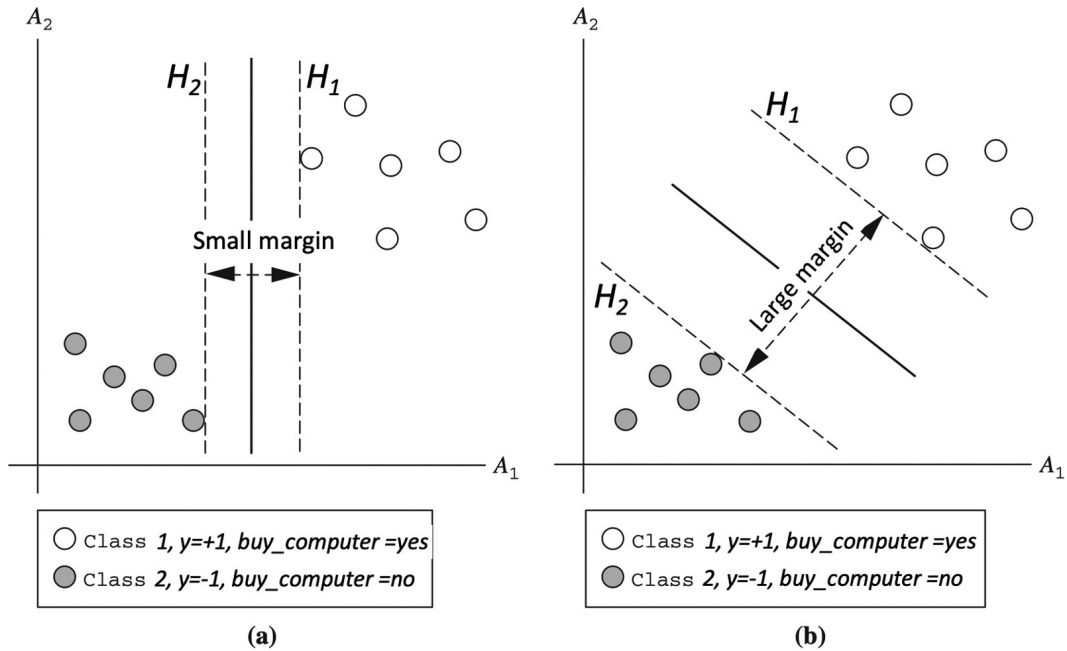


FIGURE 7.6

Here we see just two possible separating hyperplanes and their associated margins. Which one is better? The one with the larger margin (b) should have greater generalization accuracy.

The weights can be adjusted so that the hyperplanes defining the “sides” of the margin can be written as

$$H_1 : b + w_1x_1 + w_2x_2 \geq 1 \quad \text{for } y_i = +1, \quad (7.11)$$

$$H_2 : b + w_1x_1 + w_2x_2 \leq -1 \quad \text{for } y_i = -1. \quad (7.12)$$

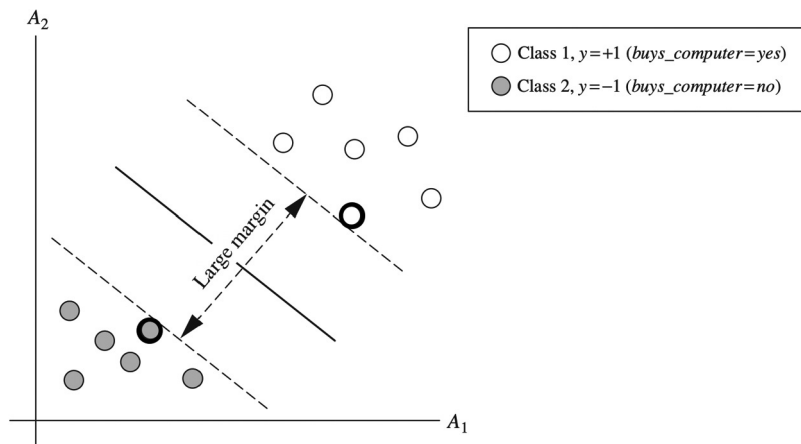
That is, any tuple that falls on or above  $H_1$  belongs to class +1, and any tuple that falls on or below  $H_2$  belongs to class -1. Combining the two inequalities of Eqs. (7.11) and (7.12), we get

$$y_i(b + w_1x_1 + w_2x_2) \geq 1, \quad \forall i. \quad (7.13)$$

Any training tuples that fall on hyperplanes  $H_1$  or  $H_2$  (i.e.,  $y_i(b + w_1x_1 + w_2x_2) = 1$ ) are called **support vectors**. That is, they are equally close to the (separating) MMH. In Fig. 7.7, the support vectors are shown encircled with a thicker border. Essentially, the support vectors are the most difficult tuples to classify and give the most information regarding classification.

From this, we can obtain a formula for the size of the maximal margin. The distance from the separating hyperplane to any point on  $H_1$  is  $\frac{1}{\|\mathbf{W}\|}$ , where  $\|\mathbf{W}\|$  is the Euclidean norm of  $\mathbf{W}$ , that is,





**FIGURE 7.7**

Support vectors. The SVM finds the maximum margin separating hyperplane, that is, the one with maximum distance between the nearest training tuples. The support vectors are shown with a thicker border.

$\sqrt{\mathbf{W} \cdot \mathbf{W}}$ .<sup>3</sup> By definition, this is equal to the distance from any point on  $H_2$  to the separating hyperplane. Therefore the maximal margin is  $\frac{2}{\|\mathbf{W}\|}$ . This suggests that we should minimize  $\|\mathbf{W}\|^2$  in order to make the margin as large as possible. Notice that if the tuples are in  $n$  dimensional space, Eq. (7.13) becomes  $y_i(\mathbf{W}'\mathbf{X}_i + b) \geq 1$ . Putting it together, we have the following mathematical formulation of SVM:

$$\begin{aligned} \min \quad & \|\mathbf{W}\|^2, \\ \text{s.t.} \quad & y_i(\mathbf{W}'\mathbf{X}_i + b) \geq 1, \quad \forall i. \end{aligned} \quad (7.14)$$

The intuition of the above formulation is that we want to find a linear classifier (i.e., hyperplane), such that (1) its margin is as large as possible (i.e.,  $\min \|\mathbf{W}\|^2$ ), and (2) each training tuple is correctly classified (i.e.,  $y_i(\mathbf{W}'\mathbf{X}_i + b) \geq 1, \forall i$ ). The corresponding classifier is often called *hard-margin linear SVM*.

“So, how does an SVM find the MMH and the support vectors?” Using some “fancy math tricks,” we can rewrite Eq. (7.14) so that it becomes what is known as a (convex) quadratic programming problem. Such fancy math tricks are beyond the scope of this book. Advanced readers may be interested to note that the tricks involve rewriting Eq. (7.14) as its dual form using Lagrangian formulation and then solving for the solution using Karush-Kuhn-Tucker (KKT) conditions. Details can be found in the bibliographic notes at the end of this chapter (Section 7.10).

If the data are relatively small (say, with a few thousand training tuples), any optimization software package for solving convex quadratic programming problems can then be used to find the support vectors and MMH. For larger data, special and more efficient algorithms for training SVMs can be

<sup>3</sup> If  $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ , then  $\sqrt{\mathbf{W} \cdot \mathbf{W}} = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$ .

used instead, the details of which exceed the scope of this book. Once we've found the support vectors and MMH (note that the support vectors define the MMH!), we have a trained support vector machine. The MMH is a linear class boundary, and so the corresponding SVM can be used to classify linearly separable data.

“Once I've got a trained support vector machine, how do I use it to classify test (i.e., new) tuples?” Based on the Lagrangian formulation mentioned before, the MMH can be rewritten as the decision boundary

$$d(\mathbf{X}) = \sum_{i=1}^l y_i \alpha_i \mathbf{X}' \mathbf{X}_i + b, \quad (7.15)$$

where  $y_i$  is the class label of support vector  $\mathbf{X}_i$ ;  $\mathbf{X}$  is a test tuple and  $'$  denotes the transpose of a vector;  $\alpha_i$  and  $b$  are numeric parameters that were determined automatically by the optimization or SVM algorithm noted before; and  $l$  is the number of support vectors, which is often much smaller than the total number of training tuples. Interested readers may note that the  $\alpha_i$  are Lagrangian multipliers. For linearly separable data, the support vectors are a subset of the actual training tuples. Slight twist regarding this when dealing with nonlinearly separable data, as we shall see in the following.

Given a test tuple,  $\mathbf{X}$ , we plug it into Eq. (7.15), and then check to see the sign of the result. This tells us on which side of the hyperplane the test tuple falls. If the sign is positive, then  $\mathbf{X}$  falls above the MMH, and so the SVM predicts that  $\mathbf{X}$  belongs to class  $+1$  (representing *buys\_computer = yes*, in our case). If the sign is negative, then  $\mathbf{X}$  falls below the MMH and the class prediction is  $-1$  (representing *buys\_computer = no*).

Notice that the Lagrangian formulation of our problem Eq. (7.15) contains a dot product between support vector  $\mathbf{X}_i$  and test tuple  $\mathbf{X}$ . This will prove very useful for finding the MMH and support vectors of a nonlinear SVM when the given data are linearly inseparable, as described further in the next section. However, before that, let's briefly introduce how we can modify the formulation of hard-margin linear SVM (Eq. (7.14)) for the nonlinear case. That is, we still wish to find a linear classifier (i.e. a hyperplane) when the training tuples are linearly inseparable. Here, the trick is that we allow some training tuples to be mis-classified. To be specific, we can introduce a nonnegative slack variable  $\xi_i \geq 0$  for each training tuple,  $\mathbf{X}_i$ . If  $\xi_i = 0$ , it means that the corresponding tuple  $\mathbf{X}_i$  is correctly classified by the hyperplane (i.e.,  $y_i(\mathbf{W}'\mathbf{X}_i + b) \geq 1$ ). In other words, a training example with  $\xi_i = 0$  is just like the one in the hard-margin linear SVM. However, if  $\xi_i > 0$ , it means that the tuple  $\mathbf{X}_i$  is incorrectly classified by the hyperplane and its magnitude  $|\xi_i|$  indicates how far the training tuple is away from its corresponding side (i.e.,  $H_1$  for a positive training example, and  $H_2$  for a negative training example). See Fig. 7.8(a) for an illustration.

Then we have the following alternative mathematical formulation of SVM. The corresponding classifier is often called *soft-margin linear SVM*. Different from hard-margin linear SVM, our new objective function has two terms, including (1)  $\|\mathbf{W}\|^2$ , which measures the size of margin (i.e., the smaller  $\|\mathbf{W}\|^2$ , the larger margin), and (2) the sum of all slack variables  $\sum_{i=1}^N \xi_i$ , which measures the (approximate) number of incorrectly classified training tuples (i.e., the training error). In Eq. (7.16),  $N$  is the total number of training tuples and  $C > 0$  is a user-tuned parameter that balances the size of margin and the training error. Note that we can use the same optimization technique (i.e., convex quadratic programming) to solve Eq. (7.16) as for hard-margin linear SVM. Likewise, the resulting soft-margin linear classifier uses the same equation (Eq. (7.15)) to classify a test tuple.

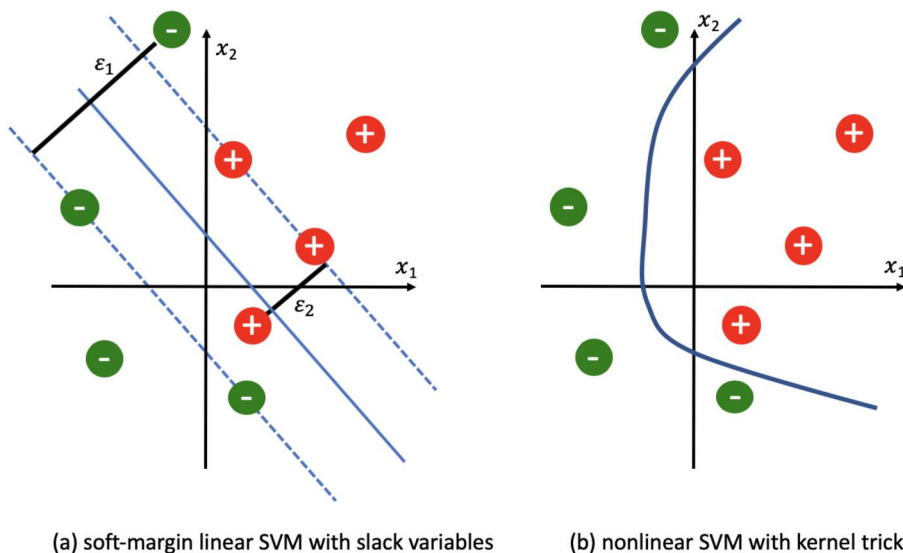


FIGURE 7.8

A simple 2-D case showing linearly inseparable data, where each tuple is represented by two attributes ( $x_1$  and  $x_2$ ). Unlike the linearly separable data of Fig. 7.5, here it is not possible to draw a straight line to perfectly separate the two classes. We could use a soft-margin linear SVM, with the help of slack variables ( $\epsilon_1$  and  $\epsilon_2$ ), to produce a linear decision boundary at the expense of two training tuples being mis-classified (a). Alternatively, we could seek for a nonlinear decision boundary (b).

$$\begin{aligned} \min \quad & \|\mathbf{W}\|^2 + C \sum_{i=1}^N \xi_i, \\ \text{s.t.} \quad & y_i (\mathbf{W}'\mathbf{X}_i + b) \geq 1 - \xi_i, \quad \forall i. \end{aligned} \quad (7.16)$$

We end this section with two important things to note. The complexity of the learned classifier is characterized by the number of support vectors rather than the dimensionality of the data. Hence SVMs tend to be less prone to overfitting than some other methods. The support vectors are the essential or critical training tuples—they lie closest to the decision boundary (MMH). If all other training tuples were removed and training were repeated, the same separating hyperplane would be found. Furthermore, the number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality. An SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high.

### 7.3.2 Nonlinear support vector machines

In Section 7.3.1, we learned about hard-margin linear SVMs for classifying linearly separable data. We also learned about soft-margin linear SVMs when the training data are linearly inseparable, by allowing

a small fraction of training tuples to be mis-classified. However, what if we want a “better” classifier to avoid such mis-classifications? For linearly inseparable cases (e.g., Fig. 7.8), no straight line can be found that would perfectly separate the classes.

The good news is that the approaches described for linear SVMs with both hard-margin and soft margin can be extended to create *nonlinear SVMs* for the classification of *linearly inseparable data* (also called *nonlinearly separable data*, or *nonlinear data* for short). Such SVMs are capable of finding nonlinear decision boundaries (i.e., nonlinear hypersurfaces) in input space.

“So,” you may ask, “*how can we extend the linear approach?*” We obtain a nonlinear SVM by extending the approach for linear SVMs as follows. There are two main steps. In the first step, we transform the original input data into a higher dimensional space using a nonlinear mapping. Several common nonlinear mappings can be used in this step, as we will further describe next. Once the data have been transformed into the new higher dimensional space, the second step searches for a linear separating hyperplane in the new space. We again end up with an optimization problem that can be solved using the linear SVM formulation (i.e., convex quadratic programming). The maximal margin hyperplane found in the new space corresponds to a nonlinear separating hypersurface in the original space.

**Example 7.2. Nonlinear transformation of original input data into a higher dimensional space.**

Consider the following example. A 3-D input vector  $\mathbf{X} = (x_1, x_2, x_3)$  is mapped into a 6-D space,  $\mathbf{Z}$ , using the mappings  $\phi_1(\mathbf{X}) = x_1$ ,  $\phi_2(\mathbf{X}) = x_2$ ,  $\phi_3(\mathbf{X}) = x_3$ ,  $\phi_4(\mathbf{X}) = (x_1)^2$ ,  $\phi_5(\mathbf{X}) = x_1x_2$ , and  $\phi_6(\mathbf{X}) = x_1x_3$ . A decision hyperplane in the new space is  $d(\mathbf{Z}) = \mathbf{W}'\mathbf{Z} + b$ , where  $\mathbf{W}$  and  $\mathbf{Z}$  are vectors. This is linear with respect to the new features  $\mathbf{Z}$ . We solve for  $\mathbf{W}$  and  $b$  and then substitute back so that the linear decision hyperplane in the new ( $\mathbf{Z}$ ) space corresponds to a nonlinear second-order polynomial in the original 3-D input space:

$$\begin{aligned} d(\mathbf{Z}) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b. \end{aligned}$$

□

However, there are some problems. First, how do we choose the nonlinear mapping to a higher dimensional space? Second, the computation involved will be costly. Refer to Eq. (7.15) for the classification of a test tuple,  $\mathbf{X}$ . Given the test tuple, we have to compute its dot product with every one of the support vectors.<sup>4</sup> In training, we have to compute a similar dot product for each pair of training tuples in order to find the MMH. This is especially expensive. Hence, the dot product computation required is very heavy and costly. We need another trick!

Luckily, we can use another math trick. It so happens that in solving the quadratic optimization problem of the linear SVM (i.e., when searching for a linear SVM in the new higher dimensional space), the training tuples appear only in the form of dot products,  $\phi(\mathbf{X}_i) \cdot \phi(\mathbf{X}_j)$ , where  $\phi(\mathbf{X})$  is simply the nonlinear mapping function applied to transform the training tuples. Instead of computing the dot product on the transformed data tuples, it turns out that it is mathematically equivalent to instead

<sup>4</sup> The dot product of two vectors,  $\mathbf{X} = (x_1, x_2, \dots, x_n)$  and  $\mathbf{X}_i = (x_{i1}, x_{i2}, \dots, x_{in})$  is  $x_1x_{i1} + x_2x_{i2} + \dots + x_nx_{in}$ . Note that this involves one multiplication and one addition for each of the  $n$  dimensions.

applying a *kernel function*,  $K(\mathbf{X}_i, \mathbf{X}_j)$ , to the original input data. That is,

$$K(\mathbf{X}_i, \mathbf{X}_j) = \phi(\mathbf{X}_i) \cdot \phi(\mathbf{X}_j). \quad (7.17)$$

In other words, everywhere that  $\phi(\mathbf{X}_i) \cdot \phi(\mathbf{X}_j)$  appears in the training algorithm, we can replace it with  $K(\mathbf{X}_i, \mathbf{X}_j)$ . In this way, all calculations are made in the original input space, which is of potentially much lower dimensionality! We can safely avoid the mapping—it turns out that we don't even have to know what the mapping is! We will talk more later about what kinds of functions can be used as kernel functions for this problem. After applying this trick, we can then proceed to find a maximal margin separating hyperplane. The procedure is similar to that described in Section 7.3.1.

**Example 7.3.** Fig. 7.9(a) shows a training set with four positive tuples and four negative tuples. In the original feature space, each tuple is represented by two features ( $x_1$  and  $x_2$ ), where the training set is linearly inseparable (Fig. 7.9(b)). If we transform the original feature space into a 3-D space:  $\Phi_1 = x_1^2$ ,  $\Phi_2 = x_2^2$  and  $\Phi_3 = \sqrt{2}x_1x_2$ . In the transformed feature space (Fig. 7.9(c)), the positive tuples are linearly separable from the negative tuples. In other words, we can use a hyperplane  $\Phi_1 + \Phi_2 = 2.5$  to perfectly separate all positive tuples from all negative tuples. The hyperplane in the transformed feature space is equivalent to a nonlinear decision boundary in the original 2-D space  $x_1^2 + x_2^2 = 2.5$ . Note that the dot product of two tuples ( $\mathbf{X}_i$  and  $\mathbf{X}_j$ ) in the transformed feature space can be computed directly from the original feature space:  $\Phi(\mathbf{X}_i) \cdot \Phi(\mathbf{X}_j) = (\mathbf{X}_i \cdot \mathbf{X}_j)^2$ .  $\square$

“What are some of the kernel functions that could be used?” Properties of the kinds of kernel functions that could be used to replace the dot product have been studied. Three admissible kernel functions are

$$\text{Polynomial kernel of degree } h: \quad K(\mathbf{X}_i, \mathbf{X}_j) = (\mathbf{X}_i \cdot \mathbf{X}_j + 1)^h,$$

$$\text{Gaussian radial basis function kernel:} \quad K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\|\mathbf{X}_i - \mathbf{X}_j\|^2 / 2\sigma^2},$$

$$\text{Sigmoid kernel:} \quad K(\mathbf{X}_i, \mathbf{X}_j) = \tanh(\kappa \mathbf{X}_i \cdot \mathbf{X}_j - \delta).$$

Each of these results in a different nonlinear classifier in (the original) input space. There are no golden rules for determining which admissible kernel will result in the most accurate SVM. In practice, the kernel chosen does not generally make a large difference in resulting accuracy.

So far, we have described linear and nonlinear SVMs for binary (i.e., two-class) classification. SVM classifiers can be combined for the multiclass case. See Section 7.7.1 for some strategies, such as training one classifier per class and the use of error-correcting codes.

A major research goal regarding SVMs is to improve the speed in training and testing so that SVMs may become a more feasible option for very large data sets (e.g., millions of support vectors). A very efficient strategy is to train SVMs in its prime form directly (e.g., Eqs. (7.14) and (7.16)) based on stochastic subgradient descent. Recall that in Chapter 7, we have used a similar technique called stochastic gradient descent to address the scalability issue of logistic regression classifier. Other issues include (1) determining the best kernel for a given data set and finding more efficient methods for the multiclass case, (2) making the SVMs more robust to the noise in the training data by using alternative norms of the weight vector  $\mathbf{W}$  (e.g.,  $l_1$  norm SVM,  $l_{2,1}$  norm SVM, capped  $l_p$  norm SVM). A key idea behind nonlinear SVM is the kernel trick, where we find a nonlinear classifier without explicitly constructing the nonlinear mapping. The kernel trick has been broadly applied to other data mining tasks, including regression, clustering, and so on.

tuple index	1	2	3	4	5	6	7	8
$x_1$	1	1	-1	-1	2	-2	2	-2
$x_2$	1	-1	1	-1	2	2	-2	-2
Class label	+	+	+	+	-	-	-	-
$x_1^2$	1	1	1	1	4	4	4	4
$x_2^2$	1	1	1	1	4	4	4	4
$\sqrt{2}x_1x_2$	$\sqrt{2}$	$-\sqrt{2}$	$-\sqrt{2}$	$\sqrt{2}$	$4\sqrt{2}$	$-4\sqrt{2}$	$-4\sqrt{2}$	$4\sqrt{2}$

(a) Training tuples in the original feature space  $(x_1, x_2)$  and in the mapped feature space (shaded)

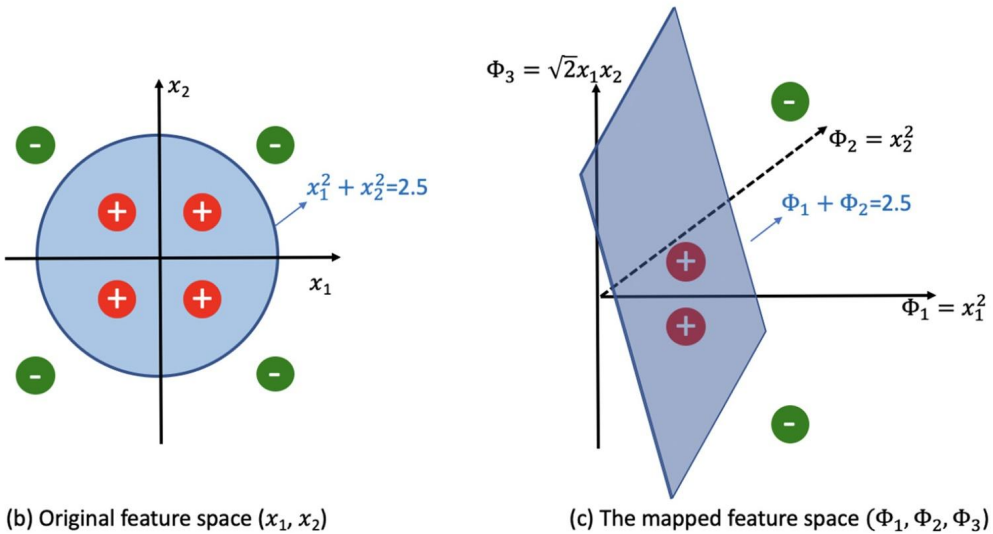


FIGURE 7.9

An example of kernel trick. (a) Training tuples in the original 2-D feature space and the transformed 3-D space (shaded). Training tuples in the original feature space are linearly inseparable (b), but become linearly separable in the transformed feature space (c). A linear decision boundary (i.e., a hyperplane) in the transformed feature space is equivalent to a nonlinear decision boundary in the original feature space. The dot product of two tuples in the transformed feature space can be computed directly from the original feature space.

## 7.4 Rule-based and pattern-based classification

In this section, we look at rule-based and pattern-based classifiers. For the former, the learned model is represented as a set of IF-THEN rules. We first examine how such rules are used for classification (Section 7.4.1). We then study ways in which they can be generated, either from a decision tree (Section 7.4.2) or directly from the training data using a *sequential covering algorithm* (Section 7.4.3). Based on that, we introduce pattern-based classifiers, where frequent patterns are used for classification. Section 7.4.4 explores **associative classification**, where association rules are generated from frequent

patterns and used for classification. The general idea is that we can search for strong associations between frequent patterns (conjunctions of attribute–value pairs) and class labels. Associative classification is a form of rule-based classifier, in that we often organize the mined association rule to form a rule-based classifier. Section 7.4.5 explores **discriminative frequent pattern–based classification**, where frequent patterns serve as combined features, which are considered in addition to single features when building a classification model. Because frequent patterns explore highly confident associations among multiple attributes, frequent pattern–based classification may overcome some constraints introduced by decision tree induction, which often only considers one attribute at a time. Studies have shown many frequent pattern–based classification methods to have greater accuracy and scalability than some traditional classification methods such as C4.5.

### 7.4.1 Using IF-THEN rules for classification

Rules are a good way of representing information or bits of knowledge. A **rule-based classifier** uses a set of IF-THEN rules for classification. An **IF-THEN** rule is an expression of the form

IF *condition* THEN *conclusion*.

An example is rule *R1*,

*R1*: IF *age* = *youth* AND *student* = *yes* THEN *buys\_computer* = *yes*.

The “IF” part (or left side) of a rule is known as the **rule antecedent** or **precondition**. The “THEN” part (or right side) is the **rule consequent**. In the rule antecedent, the condition consists of one or more *attribute tests* (e.g., *age* = *youth* and *student* = *yes*) that are logically ANDed. The rule’s consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). *R1* can also be written as

*R1*: (*age* = *youth*) ∧ (*student* = *yes*) ⇒ (*buys\_computer* = *yes*).

If the condition (i.e., all the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is **satisfied** (or simply, that the rule is satisfied) and that the rule **covers** the tuple.

A rule *R* can be assessed by its coverage and accuracy. Given a tuple, *X*, from a class-labeled data set, *D*, let *n<sub>covers</sub>* be the number of tuples covered by *R*; *n<sub>correct</sub>* be the number of tuples correctly classified by *R*; and |*D*| be the number of tuples in *D*. We can define the **coverage** and **accuracy** of *R* as

$$\text{coverage}(R) = \frac{n_{\text{covers}}}{|D|} \quad (7.18)$$

$$\text{accuracy}(R) = \frac{n_{\text{correct}}}{n_{\text{covers}}}. \quad (7.19)$$

That is, a rule’s coverage is the percentage of tuples that are covered by the rule (i.e., their attribute values hold true for the rule’s antecedent). For a rule’s accuracy, we look at the tuples that it covers and see what percentage of them the rule can correctly classify.

**Example 7.4. Rule accuracy and coverage.** Let's go back to our data in Section 6.2, Table 6.1. These are class-labeled tuples from the *AllElectronics* customer database. Our task is to predict whether a customer will buy a computer. Consider rule  $R1$ , which covers 2 of the 14 tuples. It can correctly classify both tuples. Therefore  $coverage(R1) = 2/14 = 14.28\%$  and  $accuracy(R1) = 2/2 = 100\%$ .  $\square$

Let's see how we can use rule-based classification to predict the class label of a given tuple,  $X$ . If a rule is satisfied by  $X$ , the rule is said to be **triggered**. For example, suppose we have

$$X = (age = youth, income = medium, student = yes, credit\_rating = fair).$$

We would like to classify  $X$  according to *buys\_computer*.  $X$  satisfies  $R1$ , which triggers the rule.

If  $R1$  is the only rule satisfied, then the rule **fires** by returning the class prediction for  $X$ . Note that triggering does not always mean firing because there may be more than one rule that is satisfied! If more than one rule is triggered, we have a potential problem. What if each of them specifies a different class? Or what if no rule is satisfied by  $X$ ?

We tackle the first question. If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to  $X$ . There are many possible strategies. We look at two, namely *size ordering* and *rule ordering*.

The **size ordering** scheme assigns the highest priority to the triggering rule that has the “toughest” requirements, where toughness is measured by the rule antecedent *size*. That is, the triggering rule with the most attribute tests is fired.

The **rule ordering** scheme prioritizes the rules beforehand. The ordering may be *class-based* or *rule-based*. With **class-based ordering**, the classes are sorted in order of decreasing “importance” such as by decreasing *order of prevalence*. That is, all the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on. Alternatively, they may be sorted based on the misclassification cost per class. Within each class, the rules are not ordered—they don't have to be because they all predict the same class (and so there can be no class conflict!).

With **rule-based ordering**, the rules are organized into one long priority list, according to some measure of rule quality, such as accuracy, coverage, size (number of attribute tests in the rule antecedent), or based on advice from domain experts. When rule ordering is used, the rule set is known as a **decision list**. With rule ordering, the triggering rule that appears earliest in the list has the highest priority, and so it gets to fire its class prediction. Any other rule that satisfies  $X$  is ignored. Most rule-based classification systems use a class-based rule-ordering strategy.

Note that in the first strategy, overall the rules are *unordered*. They can be applied in any order when classifying a tuple. That is, a disjunction (logical OR) is implied between different rules. Each rule represents a standalone nugget or piece of knowledge. This is in contrast to the rule ordering (decision list) scheme for which rules must be applied in the prescribed order so as to avoid conflicts. Each rule in a decision list implies the negation of the rules that come before it in the list. Hence rules in a decision list are more difficult to interpret.

Now that we have seen how we can handle conflicts, let's go back to the scenario where there is no rule satisfied by  $X$ . How, then, can we determine the class label of  $X$ ? In this case, a fallback or **default rule** can be set up to specify a default class, based on a training set. This may be the class in majority or the majority class of the tuples that were not covered by any rule. The default rule is evaluated at the



end, if and only if no other rule covers  $X$ . The condition in the default rule is empty. In this way, the rule fires when no other rule is satisfied.

In the following sections, we examine how to build a rule-based classifier.

## 7.4.2 Rule extraction from a decision tree

In Section 6.2, we learned how to build a decision tree classifier from a set of training data. Decision tree classifiers are a popular method of classification—it is easy to understand how decision trees work and they are known for their accuracy. Decision trees can become large and difficult to interpret. In this subsection, we look at how to build a rule-based classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent (“IF” part). The leaf node holds the class prediction, forming the rule consequent (“THEN” part).

**Example 7.5. Extracting classification rules from a decision tree.** The decision tree of Fig. 6.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Fig. 6.2 are as follows:

R1 : IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>no</i>	THEN <i>buys_computer</i> = <i>no</i>
R2 : IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>yes</i>	THEN <i>buys_computer</i> = <i>yes</i>
R3 : IF <i>age</i> = <i>middle_aged</i>		THEN <i>buys_computer</i> = <i>yes</i>
R4 : IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>excellent</i>	THEN <i>buys_computer</i> = <i>yes</i>
R5 : IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>fair</i>	THEN <i>buys_computer</i> = <i>no</i> .

□

A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are **mutually exclusive** and **exhaustive**. *Mutually exclusive* means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.) *Exhaustive* means there is one rule for each possible attribute–value combination, so that this set of rules does not require a default rule. Therefore the order of the rules does not matter—they are *unordered*.

Since we end up with one rule per leaf, the set of extracted rules is not much simpler than the corresponding decision tree! The extracted rules may be even more difficult to interpret than the original trees in some cases. As an example, Fig. 6.7 shows decision trees that suffer from subtree repetition and replication. The resulting set of rules extracted can be large and difficult to follow, because some of the attribute tests may be irrelevant or redundant. So, the plot thickens. Although it is easy to extract rules from a decision tree, we may need to do some more work by pruning the resulting rule set.

“How can we prune the rule set?” For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed), thereby generalizing the rule. C4.5 extracts rules from an unpruned tree, and then prunes the rules using a pessimistic approach similar to its tree pruning method. The training tuples and their associated class labels are used to estimate rule

accuracy. However, because this would result in an optimistic estimate, alternatively, the estimate is adjusted to compensate for the bias, resulting in a pessimistic estimate. In addition, any rule that does not contribute to the overall accuracy of the entire rule set can also be pruned.

Other problems arise during rule pruning, however, as the rules *will no longer be* mutually exclusive and exhaustive. For conflict resolution, C4.5 adopts a **class-based ordering scheme**. It groups together all rules for a single class, and then determines a ranking of these class rule sets. Within a rule set, the rules are not ordered. C4.5 orders the class rule sets to minimize the number of *false-positive errors* (i.e., where a rule predicts a class,  $C$ , but the actual class is not  $C$ ). The class rule set with the least number of false positives is examined first. Once pruning is complete, a final check is done to remove any duplicates. When choosing a default class, C4.5 does not choose the majority class, because this class will likely have many rules for its tuples. Instead, it selects the class that contains the most training tuples that were not covered by any rule.

### 7.4.3 Rule induction using a sequential covering algorithm

IF-THEN rules can be extracted directly from the training data (i.e., without having to generate a decision tree first) using a **sequential covering algorithm**. The name comes from the notion that the rules are learned *sequentially* (one at a time), where each rule for a given class will ideally *cover* many of the class's tuples (and hopefully none of the tuples of other classes). Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules and form the topic of this subsection.

There are many sequential covering algorithms. Popular variations include AQ, CN2, and the more recent RIPPER. The general strategy is as follows. Rules are learned one at a time. Each time a rule is learned, the tuples covered by the rule are removed, and the process repeats on the remaining tuples. This sequential learning of rules is in contrast to decision tree induction. Because the path to each leaf in a decision tree corresponds to a rule, we can consider decision tree induction as learning a set of rules *simultaneously*.

A basic sequential covering algorithm is shown in Fig. 7.10. Here, rules are learned for one class at a time. Ideally, when learning a rule for a class,  $C$ , we would like the rule to cover all (or as many as possible) of the training tuples of class  $C$  and none (or as few as possible) of the tuples from other classes. In this way, the rules learned should be of high accuracy. The rules need not necessarily be of high coverage. This is because we can have more than one rule for a class, so that different rules may cover different tuples within the same class. The process continues until the terminating condition is met, such as when there are no more training tuples or the quality of a rule returned is below a user-specified threshold. The *Learn\_One\_Rule* procedure finds the “best” rule for the current class, given the current set of training tuples.

“*How are rules learned?*” Typically, rules are grown in a *general-to-specific* manner (Fig. 7.11). We can think of this as a beam search, where we start off with an empty rule and then gradually keep appending attribute tests to it. We append by adding the attribute test as a logical conjunction to the existing condition of the rule antecedent. Suppose our training set,  $D$ , consists of loan application data. Attributes regarding each applicant include their age, income, education level, residence, credit rating, and the term of the loan. The classifying attribute is *loan\_decision*, which indicates whether a loan is accepted (considered safe) or rejected (considered risky). To learn a rule for the class “accept,” we start

**Algorithm: Sequential covering.** Learn a set of IF-THEN rules for classification.

**Input:**

- $D$ , a data set of class-labeled tuples;
- $Att\_vals$ , the set of all attributes and their possible values.

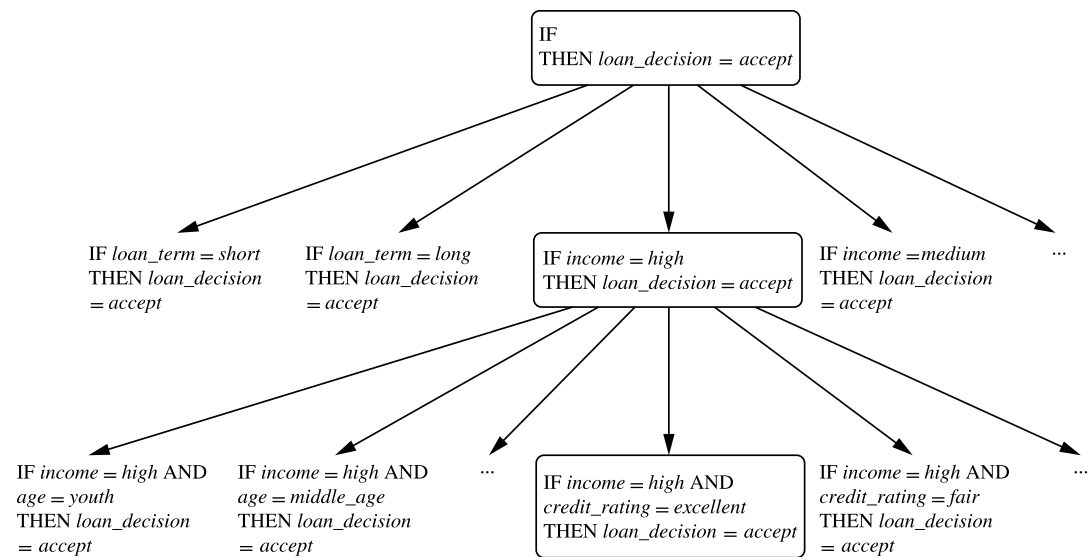
**Output:** A set of IF-THEN rules.

**Method:**

- (1)  $Rule\_set = \{\}$ ; // initial set of rules learned is empty
- (2) **for each** class  $c$  **do**
- (3)     **repeat**
- (4)         Rule = **Learn\_One\_Rule**( $D, Att\_vals, c$ );
- (5)         remove tuples covered by  $Rule$  from  $D$ ;
- (6)          $Rule\_set = Rule\_set + Rule$ ; // add new rule to rule set
- (7)     **until** terminating condition;
- (8) **endfor**
- (9) return  $Rule\_Set$ ;

**FIGURE 7.10**

Basic sequential covering algorithm.



**FIGURE 7.11**

A general-to-specific search through rule space.

off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is

IF THEN  $loan\_decision = accept$ .

We then consider each possible attribute test that may be added to the rule. These can be derived from the parameter *Att\_vals*, which contains a list of attributes with their associated values. For example, for an attribute–value pair (*att*, *val*), we can consider attribute tests such as  $att = val$ ,  $att \leq val$ ,  $att > val$ , and so on. Typically, the training data will contain many attributes, each of which may have several possible values. Finding an optimal rule set becomes computationally explosive. Instead, *Learn\_One\_Rule* adopts a greedy depth-first strategy. Each time it is faced with adding a new attribute test (conjunction) to the current rule, it picks the one that improves the rule quality most, based on the training samples. We will say more about rule quality measures in a minute. For now, let's say we use rule accuracy as our quality measure. Getting back to our example with Fig. 7.11, suppose *Learn\_One\_Rule* finds that the attribute test  $income = high$  best improves the accuracy of our current (empty) rule. We append it to the condition, so that the current rule becomes

IF  $income = high$  THEN  $loan\_decision = accept$ .

Each time we add an attribute test to a rule, the resulting rule should cover relatively more of the “accept” tuples. During the next iteration, we again consider the possible attribute tests and end up selecting  $credit\_rating = excellent$ . Our current rule grows to become

IF  $income = high$  AND  $credit\_rating = excellent$  THEN  $loan\_decision = accept$ .

The process repeats, where at each step we continue to greedily grow rules until the resulting rule meets an acceptable quality level.

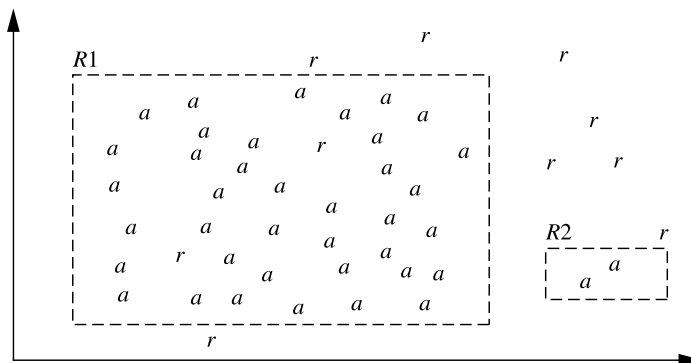
Greedy search does not allow for backtracking. At each step, we *heuristically* add what appears to be the best choice at the moment. What if we unknowingly made a poor choice along the way? To lessen the chance of this happening, instead of selecting the best attribute test to append to the current rule, we can select the best *k* attribute tests. In this way, we perform a beam search of width *k*, wherein we maintain the *k* best candidates overall at each step, rather than a single best candidate.

### Rule quality measures

*Learn\_One\_Rule* needs a measure of rule quality. Every time it considers an attribute test, it must check to see if appending such a test to the current rule's condition will result in an improved rule. Accuracy may seem like an obvious choice at first, but consider Example 7.6.

**Example 7.6. Choosing between two rules based on accuracy.** Consider the two rules as illustrated in Fig. 7.12. Both are for the class  $loan\_decision = accept$ . We use “*a*” to represent the tuples of class “accept” and “*r*” for the tuples of class “reject.” Rule *R1* correctly classifies 38 of the 40 tuples it covers. Rule *R2* covers only two tuples, which it correctly classifies. Their respective accuracies are 95% and 100%. Thus *R2* has greater accuracy than *R1*, but it is not the better rule because of its small coverage. □

From this example, we see that accuracy on its own is not a reliable estimate of rule quality. Coverage on its own is not useful either—for a given class we could have a rule that covers many tuples, most of which belong to other classes! Thus we seek other measures for evaluating rule quality, which may integrate aspects of accuracy and coverage. Here we will look at some, namely *entropy*, another based on *information gain*, and a *statistical test* that considers coverage. For our discussion, suppose we are learning rules for the class *c*. Our current rule is *R*: IF *condition* THEN  $class = c$ . We want to see



**FIGURE 7.12**

Rules for the class  $loan\_decision = accept$ , showing *accept* ( $a$ ) and *reject* ( $r$ ) tuples.

if logically ANDing a given attribute test to *condition* would result in a better rule. We call the new condition, *condition'*, where  $R'$ : IF *condition'* THEN  $class = c$  is our potential new rule. In other words, we want to see if  $R'$  is any better than  $R$ .

We have already seen entropy in our discussion of the information gain measure used for attribute selection in decision tree induction. It is also known as the *expected information* needed to classify a tuple in data set,  $D$ . Here,  $D$  is the set of tuples covered by *condition'* and  $p_i$  is the probability of class  $C_i$  in  $D$ . The lower the entropy, the better *condition'* is. Entropy prefers conditions that cover a large number of tuples of a single class and few tuples of other classes.

Another measure is based on information gain and was proposed in **FOIL** (First-Order Inductive Learner), a sequential covering algorithm that learns first-order logic rules. Learning first-order rules is more complex because such rules contain variables, whereas the rules we are concerned with in this section are propositional (i.e., variable-free).<sup>5</sup> In machine learning, the tuples of the class for which we are learning rules are called *positive* tuples, whereas the remaining tuples are *negative*. Let  $pos$  and  $neg$  be the number of positive and negative tuples covered by  $R$ , respectively. Let  $pos'$  and  $neg'$  be the number of positive (negative) tuples covered by  $R'$ , respectively. FOIL assesses the information gained by extending *condition'* as

$$FOIL\_Gain = pos' \times \left( \log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg} \right). \quad (7.20)$$

It favors rules that have high accuracy and cover many positive tuples.

We can also use a statistical test of significance to determine if the apparent effect of a rule is not attributed to chance but instead indicates a genuine correlation between attribute values and classes. The test compares the observed distribution among classes of tuples covered by a rule with the expected distribution that would result if the rule made predictions at random. We want to assess whether any

<sup>5</sup> Incidentally, FOIL was also proposed by Quinlan, the father of ID3.

observed differences between these two distributions may be attributed to chance. We can use the **likelihood ratio statistic**,

$$Likelihood\_Ratio = 2 \sum_{i=1}^m f_i \log \left( \frac{f_i}{e_i} \right), \quad (7.21)$$

where  $m$  is the number of classes.

For tuples satisfying the rule,  $f_i$  is the observed frequency of each class  $i$  among the tuples.  $e_i$  is what we would expect the frequency of each class  $i$  to be if the rule made random predictions. The statistic has a  $\chi^2$  distribution with  $m - 1$  degrees of freedom. The higher the likelihood ratio, the more likely that there is a *significant* difference in the number of correct predictions made by our rule in comparison with a “random guesser.” That is, the performance of our rule is not due to chance. The ratio helps identify rules with insignificant coverage.

CN2 uses entropy together with the likelihood ratio test, while FOIL’s information gain is used by RIPPER.

### Rule pruning

*Learn\_One\_Rule* does not employ a test set when evaluating rules. Assessments of rule quality as described previously are made with tuples from the original training data. These assessments are optimistic because the rules will likely overfit the data. That is, the rules may perform well on the training data but less well on subsequent unseen data (i.e., test data). To compensate for this, we can prune the rules. A rule is pruned by removing a conjunction (attribute test). We choose to prune a rule,  $R$ , if the pruned version of  $R$  has greater quality, as assessed on an independent set of tuples. As in decision tree pruning, we refer to this set as a *pruning set*.

FOIL uses a simple yet effective method. Given a rule,  $R$ ,

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}, \quad (7.22)$$

where  $pos$  and  $neg$  are the number of positive and negative tuples covered by  $R$ , respectively. This value will increase with the accuracy of  $R$  on a pruning set. Therefore if the *FOIL\_Prune* value is higher for the pruned version of  $R$ , then we prune  $R$ .

By convention, RIPPER starts with the most recently added conjunction when considering pruning. Conjunctions are pruned one at a time as long as this results in an improvement.

## 7.4.4 Associative classification

In this section, you will learn about associative classification. The methods discussed are CBA, CMAR, and CPAR.

Before we begin, however, let’s look at association rule mining in general. Association rules are mined in a two-step process consisting of *frequent itemset mining* followed by *rule generation*. The first step searches for patterns of attribute–value pairs that occur repeatedly in a data set, where each attribute–value pair is considered an *item*. The resulting attribute–value pairs form *frequent itemsets* (also referred to as *frequent patterns*). The second step analyzes the frequent itemsets to generate association rules. All association rules must satisfy certain criteria regarding their “accuracy” (or *confidence*)

and the proportion of the data set that they actually represent (referred to as *support*). For example, the following is an association rule mined from a data set,  $D$ , shown with its confidence and support:

$$\begin{aligned} \text{age} = \text{youth} \wedge \text{credit} = \text{OK} &\Rightarrow \text{buys\_computer} = \text{yes} \\ [\text{support} = 20\%, \text{confidence} = 93\%], \end{aligned} \quad (7.23)$$

where  $\wedge$  represents a logical “AND.” We will say more about confidence and support later.

More formally, let  $D$  be a data set of tuples. Each tuple in  $D$  is described by  $n$  attributes,  $A_1, A_2, \dots, A_n$ , and a class label attribute,  $A_{class}$ . All continuous attributes are discretized and treated as categorical (or nominal) attributes. An **item**,  $p$ , is an attribute–value pair of the form  $(A_i, v)$ , where  $A_i$  is an attribute taking a value,  $v$ . A data tuple  $\mathbf{X} = (x_1, x_2, \dots, x_n)$  satisfies an item,  $p = (A_i, v)$ , if and only if  $x_i = v$ , where  $x_i$  is the value of the  $i$ th attribute of  $\mathbf{X}$ . Association rules can have any number of items in the rule antecedent (left side) and any number of items in the rule consequent (right side). However, when mining association rules for use in classification, we are only interested in association rules of the form  $p_1 \wedge p_2 \wedge \dots \wedge p_l \Rightarrow A_{class} = C$ , where the rule antecedent is a conjunction of items,  $p_1, p_2, \dots, p_l$  ( $l \leq n$ ), associated with a class label,  $C$ . For a given rule,  $R$ , the percentage of tuples in  $D$  satisfying the rule antecedent that also has the class label  $C$  is called the **confidence** of  $R$ .

From a classification point of view, this is akin to rule accuracy. For example, a confidence of 93% for Rule in Eq. (7.23) means that 93% of the customers in  $D$  who are young and have an OK credit rating belong to the class  $\text{buys\_computer} = \text{yes}$ . The percentage of tuples in  $D$  satisfying the rule antecedent and having class label  $C$  is called the **support** of  $R$ . A support of 20% for Rule in Eq. (7.23) means that 20% of the customers in  $D$  are young, have an OK credit rating, and belong to the class  $\text{buys\_computer} = \text{yes}$ .

In general, associative classification consists of the following steps:

1. Mine the data for frequent itemsets, that is, find commonly occurring attribute–value pairs in the data.
2. Analyze the frequent itemsets to generate association rules per class, which satisfy confidence and support criteria.
3. Organize the rules to form a rule-based classifier.

Methods of associative classification differ primarily in the approach used for frequent itemset mining and in how the derived rules are analyzed and used for classification. We now look at some of the various methods for associative classification.

One of the earliest and simplest algorithms for associative classification is **CBA** (Classification Based on Associations). CBA uses an iterative approach to frequent itemset mining, similar to that described for Apriori in Section 4.2.1, where multiple passes are made over the data and the derived frequent itemsets are used to generate and test longer itemsets. In general, the number of passes made is equal to the length of the longest rule found. The complete set of rules satisfying minimum confidence and minimum support thresholds are found and then analyzed for inclusion in the classifier. CBA uses a heuristic method to construct the classifier, where the rules are ordered according to decreasing precedence based on their confidence and support. If a set of rules has the same antecedent, then the rule with the highest confidence is selected to represent the set. When classifying a new tuple, the first rule satisfying the tuple is used to classify it. The classifier also contains a default rule, having the lowest precedence, which specifies a default class for any new tuple that is not satisfied by any other rule in

the classifier. In this way, the set of rules making up the classifier form a *decision list*. In general, CBA was empirically found to be more accurate than C4.5 on a good number of data sets.

**CMAR** (Classification based on Multiple Association Rules) differs from CBA in its strategy for frequent itemset mining and its construction of the classifier. It also employs several rule pruning strategies with the help of a tree structure for efficient storage and retrieval of rules. CMAR adopts a variant of the *FP-growth* algorithm to find the complete set of rules satisfying the minimum confidence and minimum support thresholds. FP-growth was described in Section 4.2.4. FP-growth uses a tree structure, called an *FP-tree*, to register all the frequent itemset information contained in the given data set,  $D$ . This requires only two scans of  $D$ . The frequent itemsets are then mined from the FP-tree. CMAR uses an enhanced FP-tree that maintains the distribution of class labels among tuples satisfying each frequent itemset. In this way, it is able to combine rule generation together with frequent itemset mining in a single step.

CMAR employs another tree structure to store and retrieve rules efficiently and to prune rules based on confidence, correlation, and database coverage. Rule pruning strategies are triggered whenever a rule is inserted into the tree. For example, given two rules,  $R1$  and  $R2$ , if the antecedent of  $R1$  is more general than that of  $R2$  and  $\text{conf}(R1) \geq \text{conf}(R2)$ , then  $R2$  is pruned. The rationale is that highly specialized rules with low confidence can be pruned if a more generalized version with higher confidence exists. CMAR also prunes rules for which the rule antecedent and class are not positively correlated, based on an  $\chi^2$  test of statistical significance.

*“If more than one rule applies, which one do we use?”* As a classifier, CMAR operates differently than CBA. Suppose that we are given a tuple  $X$  to classify and that only one rule satisfies or matches  $X$ .<sup>6</sup> This case is trivial—we simply assign the rule’s class label. Suppose, instead, that more than one rule satisfies  $X$ . These rules form a set,  $S$ . Which rule would we use to determine the class label of  $X$ ? CBA would assign the class label of the most confident rule among the rule set,  $S$ . CMAR instead considers multiple rules when making its class prediction. It divides the rules into groups according to class labels. All rules within a group share the same class label and each group has a distinct class label.

CMAR uses a weighted  $\chi^2$  measure to find the “strongest” group of rules, based on the statistical correlation of rules within a group. It then assigns  $X$  the class label of the strongest group. In this way it considers multiple rules, rather than a single rule with highest confidence, when predicting the class label of a new tuple. In experiments, CMAR had slightly higher average accuracy in comparison with CBA. Its runtime, scalability, and use of memory were found to be more efficient.

*“Is there a way to cut down on the number of rules generated?”* CBA and CMAR adopt methods of frequent itemset mining to generate *candidate* association rules, which include all conjunctions of attribute–value pairs (items) satisfying minimum support. These rules are then examined, and a subset is chosen to represent the classifier. However, such methods generate quite a large number of rules. **CPAR** (Classification based on Predictive Association Rules) takes a different approach to rule generation, based on FOIL (a rule generation algorithm for classification). FOIL builds rules to distinguish positive tuples (e.g., *buys\_computer = yes*) from negative tuples (e.g., *buys\_computer = no*). For multiclass problems, FOIL is applied to each class. That is, for a class,  $C$ , all tuples of class  $C$  are considered positive tuples, while the rest are considered negative tuples. Rules are generated to distinguish  $C$  tuples from all others. Each time a rule is generated, the positive samples it satisfies (or *covers*) are

---

<sup>6</sup> If a rule’s antecedent satisfies or matches  $X$ , then we say that the rule satisfies  $X$ .



removed until all the positive tuples in the data set are covered. In this way, fewer rules are generated. CPAR relaxes this step by allowing the covered tuples to remain under consideration, but reducing their weight. The process is repeated for each class. The resulting rules are merged to form the classifier rule set.

During classification, CPAR employs a somewhat different multirule strategy than CMAR. If more than one rule satisfies a new tuple,  $X$ , the rules are divided into groups according to class, similar to CMAR. However, CPAR uses the best  $k$  rules of each group to predict the class label of  $X$ , based on expected accuracy. By considering the best  $k$  rules rather than all of a group's rules, it avoids the influence of lower-ranked rules. CPAR's accuracy on numerous data sets was shown to be close to that of CMAR. However, since CPAR generates far fewer rules than CMAR, it shows much better efficiency with large sets of training data.

In summary, associative classification offers an alternative classification scheme by building rules based on conjunctions of attribute–value pairs that occur frequently in data.

### 7.4.5 Discriminative frequent pattern–based classification

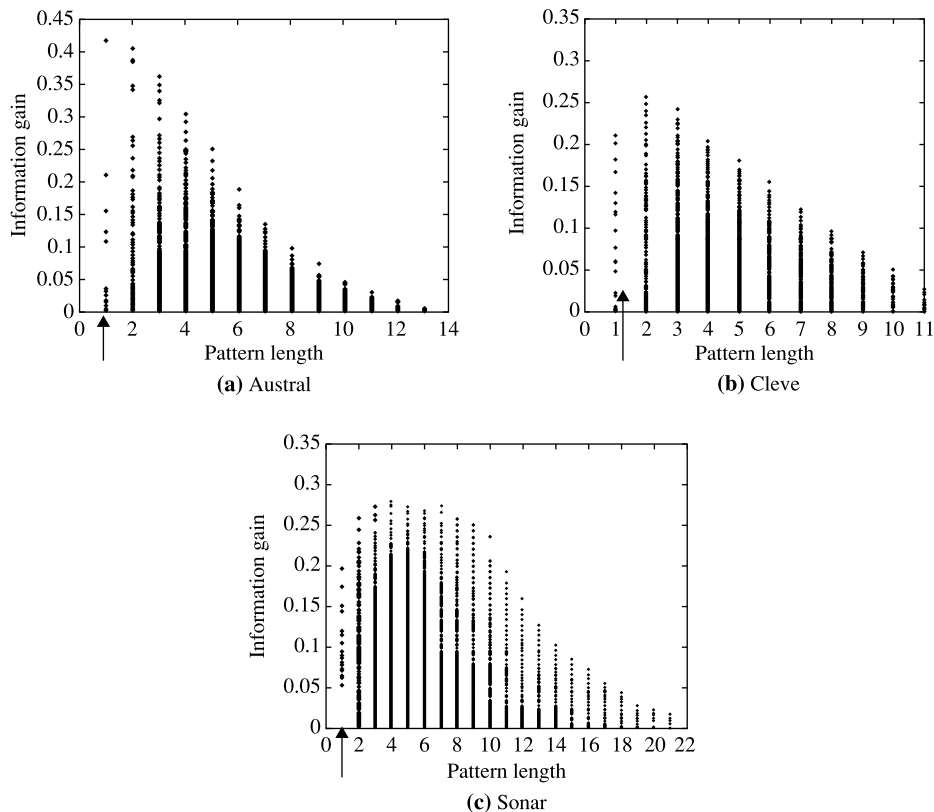
From work on associative classification, we see that frequent patterns reflect strong associations between attribute–value pairs (or items) in data and are useful for classification.

*“But just how discriminative are frequent patterns for classification?”* Frequent patterns represent feature combinations. Let's compare the discriminative power of frequent patterns and single features. Fig. 7.13 plots the information gain of frequent patterns and single features (i.e., of pattern length 1) for three UCI data sets.<sup>7</sup> The discrimination power of some frequent patterns is higher than that of single features. Frequent patterns map data to a higher-dimensional space. They capture more underlying semantics of the data and thus can hold greater expressive power than single features.

*“Why not consider frequent patterns as combined features, in addition to single features when building a classification model?”* This notion is the basis of **frequent pattern–based classification**—the learning of a classification model in the feature space of single attributes *as well as* frequent patterns. In this way, we transfer the original feature space to a larger space. This will likely increase the chance of including important features.

Let's get back to our earlier question: How discriminative are frequent patterns? Many of the frequent patterns generated in frequent itemset mining are indiscriminative because they are solely based on support, without considering predictive power. That is, by definition, a pattern must satisfy a user-specified minimum support threshold,  $min\_sup$ , to be considered frequent. For example, if  $min\_sup$  is 5%, a pattern is frequent if it occurs in 5% of the data tuples. Consider Fig. 7.14, which plots information gain vs. pattern frequency (support) for three UCI data sets. A theoretical upper bound on information gain, which was derived analytically, is also plotted. The figure shows that the discriminative power (assessed here as information gain) of low-frequency patterns is bounded by a small value. This is due to the patterns' limited coverage of the data set. Similarly, the discriminative power of very high-frequency patterns is also bounded by a small value, which is due to their commonness in the data. The upper bound of information gain is a function of pattern frequency. These observations can

<sup>7</sup> The University of California at Irvine (UCI) archives several large data sets at <http://kdd.ics.uci.edu/>. These are commonly used by researchers for the testing and comparison of machine learning and data mining algorithms.



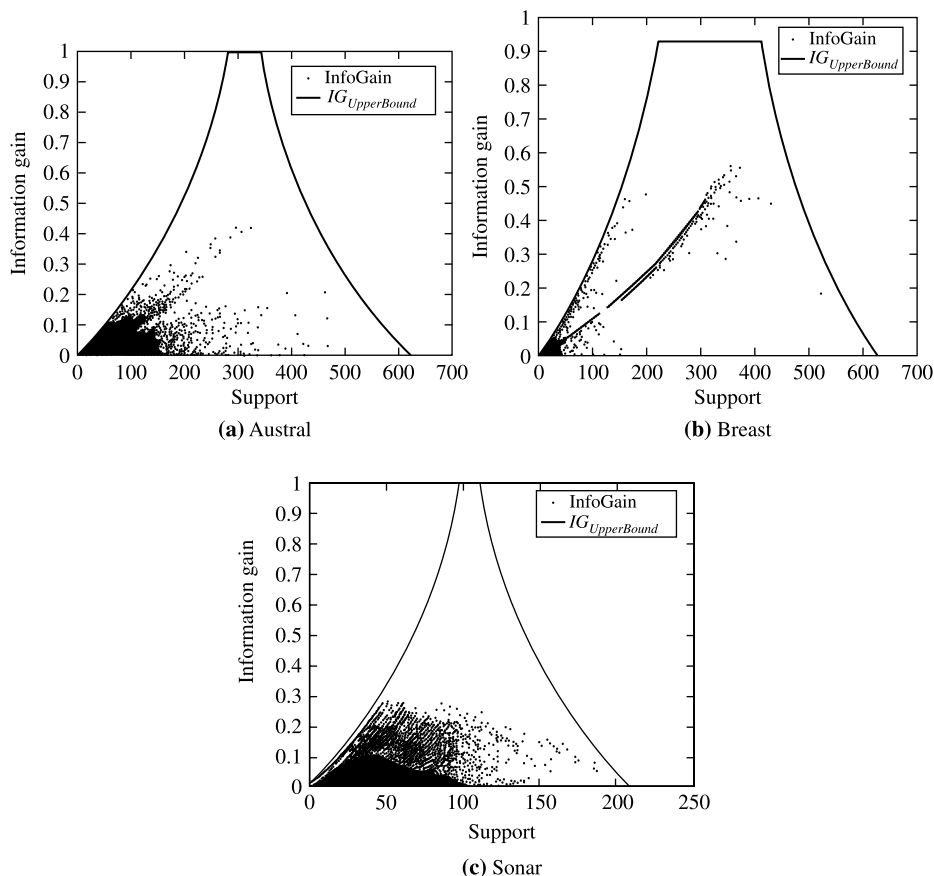
**FIGURE 7.13**

Single feature vs. frequent pattern: Information gain is plotted for single features (patterns of length 1, indicated by arrows) and frequent patterns (combined features) for three UCI data sets. *Source:* Adapted from Cheng, Yan, Han, and Hsu [CYHH07].

be confirmed analytically. Patterns with medium-large supports (e.g.,  $support = 300$  in Fig. 7.14(a)) may be discriminative or not. Thus not every frequent pattern is useful.

If we were to add all the frequent patterns to the feature space, the resulting feature space would be huge. This slows down the model learning process and may also lead to decreased accuracy due to a form of overfitting in which there are too many features. Many of the patterns may be also redundant. Therefore, it's a good idea to apply feature selection to eliminate the less discriminative and redundant frequent patterns as features. The *general framework for discriminative frequent pattern-based classification* is as follows.

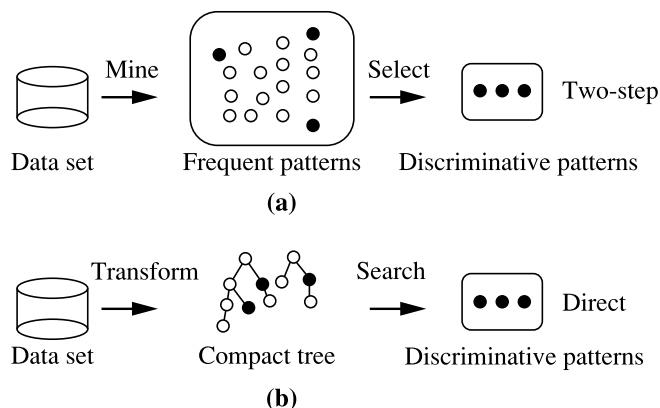
- 1. Feature generation:** The data,  $D$ , are partitioned according to class label. Use frequent itemset mining to discover frequent patterns in each partition, satisfying minimum support. The collection of frequent patterns,  $\mathcal{F}$ , makes up the feature candidates.

**FIGURE 7.14**

Information gain vs. pattern frequency (support) for three UCI data sets. A theoretical upper bound on information gain ( $IG_{UpperBound}$ ) is also shown. *Source:* Adapted from Cheng, Yan, Han, and Hsu [CYHH07].

- 2. Feature selection:** Apply feature selection to  $\mathcal{F}$ , resulting in  $\mathcal{F}_S$ , the set of selected (more discriminating) frequent patterns. Information gain, Fisher score, or other evaluation measures can be used for this step. Relevancy checking can also be incorporated into this step to weed out redundant patterns. The data set  $D$  is transformed to  $D'$ , where the feature space now includes the single features and the selected frequent patterns,  $\mathcal{F}_S$ . Commonly used feature selection methods were introduced in Section 7.1.
- 3. Learning of classification model:** A classifier is built on the data set  $D'$ . Any learning algorithm can be used as the classification model.

The general framework is summarized in Fig. 7.15(a), where the discriminative patterns are represented by dark circles. Although the approach is straightforward, we can encounter a computational

**FIGURE 7.15**

A framework for frequent pattern-based classification: (a) a two-step general approach vs. (b) the direct approach of DDPMine.

bottleneck by having to first find *all* the frequent patterns and then analyze *each one* for selection. The amount of frequent patterns found can be huge due to the explosive number of pattern combinations between items.

To improve the efficiency of the general framework, consider condensing steps 1 and 2 into just one step. That is, rather than generating the complete set of frequent patterns, it's possible to mine only the highly discriminative ones. This more direct approach is referred to as **DDPMine** (Direct Discriminative Pattern Mining). The DDPMine algorithm follows this approach, as illustrated in Fig. 7.15(b). It first transforms the training data into a compact tree structure known as a frequent pattern tree, or FP-tree (Chapter 4), which holds all of the attribute-value (itemset) association information. It then searches for discriminative patterns on the tree. The approach is direct in that it avoids generating a large number of indiscriminative patterns. It incrementally reduces the problem by eliminating training tuples, thereby progressively shrinking the FP-tree. This further speeds up the mining process.

By choosing to transform the original data to an FP-tree, DDPMine avoids generating redundant patterns because an FP-tree stores only the *closed* frequent patterns. By definition, any subpattern,  $\beta$ , of a closed pattern,  $\alpha$ , is redundant with respect to  $\alpha$  (Chapter 5). DDPMine directly mines the discriminative patterns and integrates feature selection into the mining framework. The theoretical upper bound on information gain is used to facilitate a branch-and-bound search, which prunes the search space significantly. Experimental results show that DDPMine achieves orders of magnitude speedup over the two-step approach without decline in classification accuracy. DDPMine also outperforms state-of-the-art associative classification methods in terms of both accuracy and efficiency.

Compared with associative classifiers, DDPMine is able to prune a huge number of nondiscriminative frequent patterns. However, DDPMine might still use hundreds or even thousands of frequent patterns in the classification model. *How can we further reduce the number of patterns to build a more compact classifier?* This will not only speed up the computation but also make the classifier more explainable to the end users. **DPClass** (Discriminative Pattern-based Classification) addresses this issue by combining the strength of two methods, including tree-based classifiers (e.g., decision tree clas-

sifier, random forest, etc., which were introduced in Section 6.2) and feature selection (e.g., forward selection, LASSO, etc., which were introduced in Section 7.1). DPClass works as follows. First, it uses random forest that contains multiple tree-based classifiers. Then, each prefix path from the root of a tree in random forest to its nonleaf node is treated as a discriminative pattern. Finally, it leverages feature selection, including forward feature selection and LASSO based method, to select a small subset of highly discriminative patterns to construct a linear classifier, such as logistic regression classifier or linear SVMs. The empirical evaluations on a good number of UCI data sets show that DPClass performs similarly as or better than DDPMine. On the other hand, DPClass uses a significantly less number of discriminative patterns than DDPMine. Therefore the classifier generated by DPClass is more compact, making itself faster in test and more explainable to end users.

---

## 7.5 Classification with weak supervision

The effectiveness of the classifiers we have introduced so far (e.g., SVMs, logistic regression,  $k$ -NN) largely depends on “strong supervision.” It means that in order to train a highly accurate classifier, we typically need a large number of high-quality training tuples, and the true class label for each training tuple is accurately annotated, say by the domain experts. However, what if there is only a small number of labeled training tuples? Document classification, speech recognition, computer vision, and information extraction are just a few examples of applications in which unlabeled data are abundant. Consider *document classification*, for example. Suppose we want to build a model to automatically classify text documents like articles or web pages. In particular, we want the model to distinguish between hockey and football documents. We have a vast amount of documents available, yet the documents are not class-labeled. Recall that supervised learning requires a training set, that is, a set of class-labeled data. To have a human examine and assign a class label to individual documents (to form a training set) is time consuming and expensive. *Speech recognition* requires the accurate labeling of speech utterances by trained linguists. It was reported that 1 minute of speech takes 10 minutes to label, and annotating phonemes (basic units of sound) can take 400 times as long. *Information extraction systems* are trained using labeled documents with detailed annotations. These are obtained by having human experts highlight items or relations of interest in text such as the names of companies or individuals. High-level expertise may be required for certain knowledge domains such as gene and disease mentions in biomedical information extraction. Clearly, the manual assignment of class labels to prepare a training set can be extremely costly, time consuming, and tedious. In computer vision, a fundamental task is to build a highly accurate classifier to automatically recognize various objects (i.e., class labels). However, some objects (e.g., a new type of dog) might appear only *after* the classifier has been built. In other words, there are no training tuples at all for the newly appeared class label. How can the classifier still recognize the test image of such a new type of dog?

We study five approaches for classification that are suitable for situations where there is only a limited number or no labeled training tuples. Section 7.5.1 introduces *semisupervised classification*, which builds a classifier using both labeled and unlabeled data. Section 7.5.2 presents *active learning*, where the learning algorithm carefully selects a few of the unlabeled data tuples and asks a human to label only those tuples. Section 7.5.3 presents *transfer learning*, which aims to extract the knowledge from one or more source classification tasks (e.g., classifying camera reviews) and apply the knowledge to a target classification task (e.g., classifying TV reviews). Section 7.5.4 studies *distant supervision*

whose key idea is to automatically obtain a large number of inexpensive, but potentially noisy labeled training tuples. Finally, Section 7.5.5 introduces *zero-shot learning*, which deals with the case there are no training tuples for certain class labels at all. Each of these strategies can reduce the need to annotate large amounts of data, resulting in cost and time savings. In comparison to the traditional setting that requires “strong supervision” (i.e., a large number of high-quality labeled tuples are available to train the classifier), we collectively refer to these approaches as **classification with weak supervision**.

Other forms of weak supervision exist. To name a few, *crowdsourcing learning* aims to train a classification model with a *noisy training set*. Here, the class labels are provided by workers on a crowdsourcing platform (e.g., Amazon Mechanical Turk), where we can often obtain a large amount of labeled training tuples with a relatively low cost. However, some (or many) labels provided by the crowdsourcing workers might be wrong. How to infer the true label (i.e., the ground truth) from the noisy labels is a major concern of crowdsourcing learning. Crowdsourcing learning can be viewed as a form of weakly supervised learning in that the supervision (i.e., labels) is noisy or inaccurate. In *multi-instance learning*, each training tuple (e.g., an image, a document) is called a *bag*, which consists of a set of *instances* (e.g., different regions of an image, different sentences of a document). A bag is labeled as a positive bag, as long as at least one of its instances is assigned with a positive class label. A bag is labeled as a negative bag if none of its instances has a positive class label. For example, an image is labeled as “beach” if at least one of its regions is about beach; and it is labeled as “nonbeach” if none of its regions is about beach. Given a set of labeled bags, the goal of multi-instance learning is to train a classifier to predict the label of a test (previously unseen) bag. Multi-instance learning can be viewed as a form of weakly supervised learning, in that the label (i.e., supervision) is provided at a coarse granularity (i.e., at the bag level instead of instance level). The label of a bag is also called *group-level* label (e.g., a group of regions of an image, a group of sentences of a document).

### 7.5.1 Semisupervised classification

**Semisupervised classification** uses both labeled data and unlabeled data to build a classifier. Let  $X_l = \{(x_1, y_1), \dots, (x_l, y_l)\}$  be the set of labeled data and  $X_u = \{x_{l+1}, \dots, x_n\}$  be the set of unlabeled data. Here we describe a few examples of this approach for learning.

**Self-training** is the simplest form of semisupervised classification. It first builds a classifier using the labeled data. The classifier then tries to label the unlabeled data. The tuple with the most confident label prediction is added to the set of labeled data, and the process repeats (Fig. 7.16). Although the method is easy to understand, a disadvantage is that it may reinforce errors.

**Cotraining** is another form of semisupervised classification, where two or more classifiers teach each other. Each learner uses a different and ideally independent set of features for each tuple. Consider web page data, for example, where attributes relating to the images on the page may be used as one set of features, whereas attributes relating to the corresponding text constitute another set of features for the same data. Each set of features (called “a view”) should be sufficient to train a good classifier. Suppose we split the feature set into two sets and train two classifiers,  $f_1$  and  $f_2$ , where each classifier is trained on a different set. Then,  $f_1$  and  $f_2$  are used to predict the class labels for the unlabeled data,  $X_u$ . Each classifier then teaches the other in that the tuple having the most confident prediction from  $f_1$  is added to the set of labeled data for  $f_2$  (along with its predicted label).

Similarly, the tuple having the most confident prediction from  $f_2$  is added to the set of labeled data for  $f_1$ . The method is summarized in Fig. 7.16. Cotraining is less sensitive to errors than self-training.

**Self-training**

1. Select a learning method such as Bayesian classification. Build the classifier using the labeled data,  $X_L$ .
2. Use the classifier to label the unlabeled data,  $X_U$ .
3. Select the tuple  $x \in X_U$  having the highest confidence (most confident prediction). Add it and its predicted label to  $X_L$ .
4. Repeat (i.e., retrain the classifier using the augmented set of labeled data).

**Cotraining**

1. Define two separate nonoverlapping feature sets for the labeled data,  $X_L$ .
2. Train two classifiers,  $f_1$  and  $f_2$ , on the labeled data, where  $f_1$  is trained using one of the feature sets and  $f_2$  is trained using the other.
3. Classify  $X_U$  with  $f_1$  and  $f_2$  separately.
4. Add the most confident  $(x, f_1(x))$  to the set of labeled data used by  $f_2$ , where  $x \in X_U$ . Similarly, add the most confident  $(x, f_2(x))$  to the set of labeled data used by  $f_1$ .
5. Repeat.

**FIGURE 7.16**


---

Self-training and cotraining methods of semisupervised classification.

A difficulty is that the assumptions for its usage may not hold true, that is, it may not be possible to split the features into mutually exclusive and class-conditionally independent sets.

Alternate approaches to semisupervised learning exist. For example, we can model the joint probability distribution of the features and the labels. For the unlabeled data, the labels can then be treated as missing data. The EM algorithm (Chapter 9) can be used to maximize the likelihood of the model. Semisupervised classification methods using support vector machines have also been proposed.

“*When does semisupervised classification work?*” Generally speaking, there are two commonly used assumptions behind semisupervised learning. The first assumption is *clustering assumption*, which means that data tuples from the same cluster are likely to share the same class label. The clustering algorithms will be introduced in Chapters 8 and 9. A representative example that utilizes the clustering assumption is semisupervised support vector machines (S3VMs). Recall that in the standard SVMs (Section 7.3), we seek a max-margin hyperplane that correctly separates the positive training tuples from negative tuples with a large margin. In S3VMs, it considers two design objectives, including (1) seeking a max-margin hyperplane to separate positive tuples from negative ones (which is the same as standard SVMs) and (2) avoiding to disrupt the clustering structure of unlabeled tuples. For the latter, this means that we favor a classifier (e.g., a hyperplane) that goes through the low-density region of the unlabeled tuples. The second commonly used assumption behind semisupervised learning is manifold assumption. We will not go into the technical details of manifold.<sup>8</sup> Simply put, the manifold assumption in the contexts of classification means that a pair of close tuples are likely to share the same class label. A representative example that utilizes the manifold assumption is graph-based semisupervised classification. It works as follows. First, we construct a graph whose nodes are input tuples, including both labeled and unlabeled tuples, and the edges indicate the local proximity. For example, we can link each data tuple to its  $k$ -nearest neighbors. In the constructed graph, only a small handful of nodes are

---

<sup>8</sup> In mathematical terms, a manifold is a topological space that approximates the Euclidean space in the vicinity of each data point.

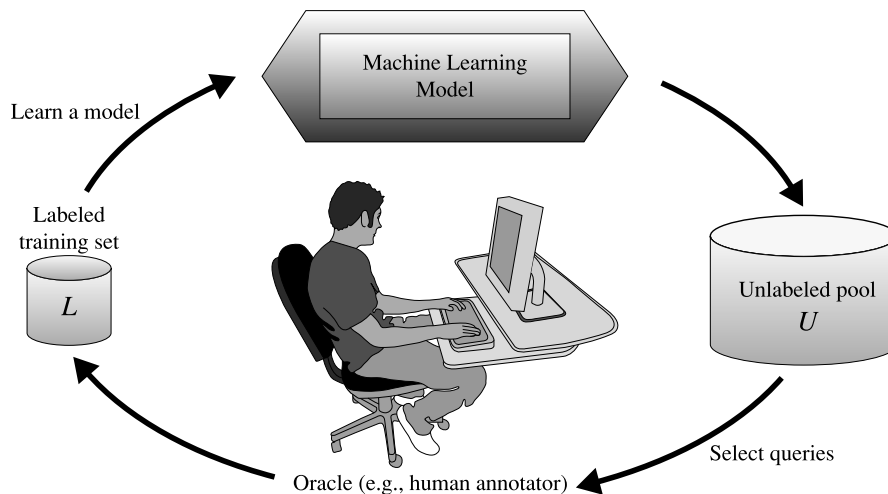
labeled and the vast majority are unlabeled. The classification method propagates the labels of these labeled nodes (i.e., tuples) to the unlabeled nodes.

### 7.5.2 Active learning

**Active learning** is an iterative type of supervised learning that is suitable for situations where data are abundant, yet the class labels are scarce or expensive to obtain. The learning algorithm is active in that it can purposefully query a user (e.g., a human annotator) for labels. The number of tuples used to learn a concept this way is often much smaller than the number required in typical supervised learning.

“How does active learning work to overcome the labeling bottleneck?” To keep costs down, the active learner aims to achieve high accuracy using as few labeled instances as possible. Let  $D$  be all of data under consideration. Various strategies exist for active learning on  $D$ . Fig. 7.17 illustrates a *pool-based approach* to active learning. Suppose that a small subset of  $D$  is class-labeled. This set is denoted  $L$ .  $U$  is the set of unlabeled data in  $D$ . It is also referred to as a pool of unlabeled data. An active learner begins with  $L$  as the initial training set. It then uses a *querying function* to carefully select one or more data samples from  $U$  and requests labels for them from an oracle (e.g., a human annotator). The newly labeled samples are added to  $L$ , which the learner then uses in a standard supervised way. The process repeats. The goal of active learning is to achieve high accuracy using as few labeled tuples as possible. Active learning algorithms are typically evaluated with the use of learning curves, which plot accuracy as a function of the number of instances queried.

Most of the active learning research focuses on how to *choose* the data tuples to be queried. Several frameworks have been proposed. *Uncertainty sampling* is the most common strategy, where the active learner chooses to query the tuples that it is the least certain how to label. *Query-by-committee* is



**FIGURE 7.17**

The pool-based active learning cycle. *Source:* From Settles [Set10], Burr Settles Computer Sciences Technical Report 1648, University of Wisconsin–Madison; used with permission.



another commonly used active learning strategy. In this method, it constructs multiple (say five) classification models and then selects the unlabeled tuple that constructed classification models have most disagreement in terms of its predicted class labels (say three classifiers predict that it belongs to positive class, whereas two classifiers predict it a negative tuple). Other strategies work to reduce the *version space*, that is, the subset of all hypotheses (i.e., classifiers) that are consistent with the observed training tuples. Alternatively, we may follow a decision-theoretic approach that estimates expected error reduction. This selects tuples that would result in the greatest reduction in the total number of incorrect predictions such as by reducing the expected entropy over  $U$ . This latter approach tends to be more computationally expensive.

### 7.5.3 Transfer learning

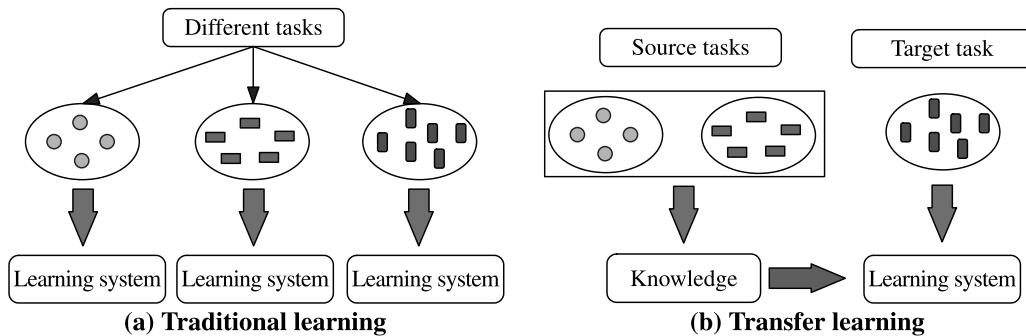
Suppose that an electronics store has collected a number of customer reviews on a product such as a brand of camera. The classification task is to automatically label the reviews as either positive or negative. This task is known as **sentiment classification**. We could examine each review and annotate it by adding a *positive* or *negative* class label. The labeled reviews can then be used to train and test a classifier to label future reviews of the product as either positive or negative. The manual effort involved in annotating the review data can be expensive and time consuming.

Now, suppose that the same store has customer reviews for other products as well, such as TVs. The distributions of review data for different types of products can vary greatly. We cannot assume that the TV-review data will have the same distribution as the camera-review data; thus we must build a separate classification model for the TV-review data. Examining and labeling the TV-review data to form a training set will require a lot of effort. In fact, we would need to label a large amount of data to train the review-classification models for each product. It would be nice if we could adapt an existing classification model (e.g., the one we built for cameras) to help learn a classification model for TVs. Such *knowledge transfer* would reduce the need to annotate a large amount of data, resulting in cost and time savings. This is the essence behind *transfer learning*.

**Transfer learning** aims to extract the knowledge from one or more *source tasks* and apply the knowledge to a *target task*. In our example, the source task is the classification of camera reviews, and the target task is the classification of TV reviews. Fig. 7.18 illustrates a comparison between traditional learning methods and transfer learning. Traditional learning methods build a new classifier for each new classification task, based on available class-labeled training and test data. Transfer learning algorithms apply knowledge about source tasks when building a classifier for a new (target) task. Construction of the resulting classifier requires fewer training data and less training time. Traditional learning algorithms assume that the training data and test data are drawn from the same distribution and the same feature space. Thus if the distribution changes, such methods need to rebuild the models from scratch.

Transfer learning allows the distributions, tasks, and even the data domains used in training and testing to be different. Transfer learning is analogous to the way humans may apply their knowledge of a task to facilitate the learning of another task. For example, if we know how to play the recorder, we may apply our knowledge of note reading and music to simplify the task of learning to play the piano. Similarly, knowing Spanish may make it easier to learn Italian.

Transfer learning is useful for common applications where the data becomes outdated or the distribution changes. Here we give two more examples. Consider *web-document classification*, where we may have trained a classifier to label, say, articles from various newsgroups according to predefined



**FIGURE 7.18**

Transfer learning vs. traditional learning. (a) Traditional learning methods build a new classifier from scratch for each classification task. (b) Transfer learning applies knowledge from a source classification task to simplify the construction of a classifier for a new, target classification task. *Source:* From Pan and Yang [PY10]; used with permission.

categories. The web data that were used to train the classifier can easily become outdated because the topics on the Web change frequently. Another application area for transfer learning is *email spam filtering*. We could train a classifier to label email as either “spam” or “not spam,” using email from a group of users. If new users come along, the distribution of their email can be different from the original group, hence the need to adapt the learned model to incorporate the new data.

There are various approaches to transfer learning, the most common of which is the *instance-based transfer learning* approach. This approach reweights some of the data from the source task and uses it to learn the target task. The **TrAdaBoost** (Transfer AdaBoost) algorithm exemplifies this approach. Consider our previous example of web-document classification, where the distribution of the old data on which the classifier was trained (the source data) is different from the newer data (the target data). TrAdaBoost assumes that the source and target domain data are each described by the same set of attributes (i.e., they have the same “feature space”) and the same set of class labels, but that the distributions of the data in the two domains are very different. It extends the AdaBoost ensemble method described in Section 6.7.3. TrAdaBoost requires the labeling of only a small amount of the target data. Rather than throwing out all the old source data, TrAdaBoost assumes that a large amount of it can be useful in training the new classification model. The idea is to filter out the influence of any old data that are very different from the new data by automatically adjusting weights assigned to the training tuples.

Recall that in boosting, an ensemble is created by learning a series of classifiers. To begin, each tuple is assigned a weight. After a classifier  $M_i$  is learned, the weights are updated to allow the subsequent classifier,  $M_{i+1}$ , to “pay more attention” to the training tuples that were misclassified by  $M_i$ . TrAdaBoost follows this strategy for the target data. However, if a source data tuple is misclassified, TrAdaBoost reasons that the tuple is probably very different from the target data. It therefore *reduces* the weight of such tuples so that they will have less effect on the subsequent classifier. As a result, TrAdaBoost can learn an accurate classification model using only a small amount of new data and a large amount of old data, even when the new data alone are insufficient to train the model. Hence in this way TrAdaBoost allows knowledge to be transferred from the old classifier to the new one.

A major challenge with transfer learning is **negative transfer**, which occurs when the new classifier performs worse than if there had been no transfer at all. Work on how to avoid negative transfer is an area of active research, where the key is to quantify the difference between the source task and the target task. *Heterogeneous transfer learning*, which involves transferring knowledge from different feature spaces and multiple source domains, is another active research topic. Traditionally, transfer learning has been used on small-scale applications. The use of transfer learning on larger applications, such as social network analysis and video classification, is often built upon the deep learning models with a “pretraining” plus “fine-tuning” strategy, which will be introduced in Chapter 10.

Transfer learning is closely related to another powerful weakly supervised learning method, namely *multitask learning*.<sup>9</sup> Let us use the sentiment classification example to illustrate the difference between transfer learning and multitask learning. In the transfer learning setting, we assume that we have a large number of manually labeled camera review data (i.e., the source task), but a very limited number of manually labeled TV review data (i.e., the target task). The goal of transfer learning is to transfer the knowledge about the source task (camera review sentiment classification) to help build a better classifier for TV review sentiment classification (i.e., the target task). Now, suppose for both TV review and camera review, we only have a small amount of manually labeled data. How can we accurately build both classifiers—one for TV review sentiment and the other for camera review sentiment? Multitask learning addresses this challenge by training both classifiers simultaneously so that the knowledge from one learning task (e.g., TV review sentiment) can be transferred to the other learning task (e.g., camera review sentiment), and vice versa.

### 7.5.4 Distant supervision

Let us take another look at the sentiment classification example. Suppose that an electronics store launches a new holiday sales campaign on social media platforms (e.g., Twitter), which goes viral with hundreds of thousands tweets. The store manager wants to figure out the sentiment of these Tweets, so that she can adjust the campaign strategy accordingly. We could manually label a large number of tweets regarding their sentiment and then train a classifier to predict the sentiment (positive vs. negative) of the remaining tweets. However, that would be time consuming. The manager wonders: “*Can we train a sentiment classifier about the tweets without any manual labels?*” **Distant supervision** aims to answer this question by automatically generate a large number of labeled tuples. In particular, the manager notices that for a large subset of the tweets, its text content contains a “:)” sign or a “:(” sign, which are often associated with positive and negative sentiments, respectively. Therefore we could treat all the tweets with a “:)” sign as positive tuples and those with a “:(” sign negative tuples and use them to train a sentiment classifier. Once the classifier is trained, we can use it to predict the sentiment for any future tweet even if it does not contain a “:)” or “:(” sign. Notice that in this case, we do not need to manually label *any* tweet in terms of its sentiment, and such labels (regarding positive or negative sentiment) are automatically generated.

In the tweet sentiment classification example above, we exploit the specific information (i.e., a “:)” or “:(” sign) in the input data to automatically generate labeled training tuples. An alternative strategy for classification with distant supervision often leverages the external knowledge base to automatically

---

<sup>9</sup> In some machine learning literature, multitask learning is viewed as a special case of transfer learning, namely inductive transfer learning where the source and target domains share the same feature (i.e., attribute) space.

generate labels for the training tuples. For example, in order to classify tweets into different categories (e.g., news, health, science, games, etc.), we could explore the Open Directory Project (ODP, <http://odp.org>), which maintain a directory for web links by volunteers. Thus if a tweet contains a url (e.g., <http://nytimes.com>), we can automatically find its ODP category (e.g., news), which is treated as the label of the corresponding tweet. In this way, we will be able to automatically generate a large labeled training set. Once the classifier is trained, we can use it to predict the class label (i.e., the category) of a test tweet, even if it does not contain a url. Another way to automatically generate labeled training examples is to leverage YouTube video that is linked to the tweet. The method is based on the following two observations. First, there are a large number of tweets, each of which contains a link to a YouTube video. Second, for each YouTube video, it is always associated with one of 18 predefined class labels. Therefore we can treat the label of YouTube video as the label of the associated tweet.

In addition to social media post classification tasks, distant supervision is also found useful for relation extraction for natural language processing. An active research direction in distant supervision is how to effectively ask users to write a *labeling function*, instead of manually label training tuples, to automatically generate labels for a large number of unlabeled data. A major limitation of distant supervision is that the automatically generated labels are often very noisy. For example, some tweets with a “:)” sign could have neutral or even negative sentiment; the class labels of a tweet does not always align with the label or category of the url (either a web page or a YouTube video) it contains.

### 7.5.5 Zero-shot learning

Suppose that we have a collection of animal images, each of which has a unique label, including “owl,” “dog,” or “fish.” Using this training data set, we can build a classifier, say SVMs or logistic regression classifier.<sup>10</sup> Then, given a test image, we can use the trained classifier to predict its class label, that is, which one of the three possible animals (owl, dog, or fish) this image is about. *But, what if the test image is actually about a cat?* In other words, the class label of the test data *never* appears in the training data. This is what *zero-shot learning* aims to address, where the classifier needs to predict a test tuple whose class label was never observed during the training stage. In other words, there is *zero* training tuples for the novel class label (e.g., cat in our example). The term “shot” here refers to data tuple.

At the first glance, this seems to be an impossible mission. You might wonder: “*If there is zero training tuples about the cat, how can I build a classifier to recognize an image about the cat?*” However, we might have some high-level description about the novel classes. For example, for “cat,” we can learn from the Wikipedia that a cat has retractable claws and super night vision. Zero-shot learning tries to leverage such external knowledge or side-information to build a classifier that can recognize such novel class labels.

Let us use the animal classification example (Fig. 7.19) to explain how zero-shot learning works. Formally, there are  $n$  training images each of which is represented by a  $d$ -D feature vector and a 3-D label vector. The label vector indicates which of the three known classes the training image belongs to. For example, for an image about a “dog,” its label vector is  $[1, 0, 0]$ . In addition, we have the external

---

<sup>10</sup> Different from the classification tasks we have seen so far which typically involve two possible class labels (e.g., positive vs. negative sentiment), in this setting, we have a multiclass classification problem since there are three possible class labels. The techniques for multiclass classification will be introduced in Section 7.7.1.

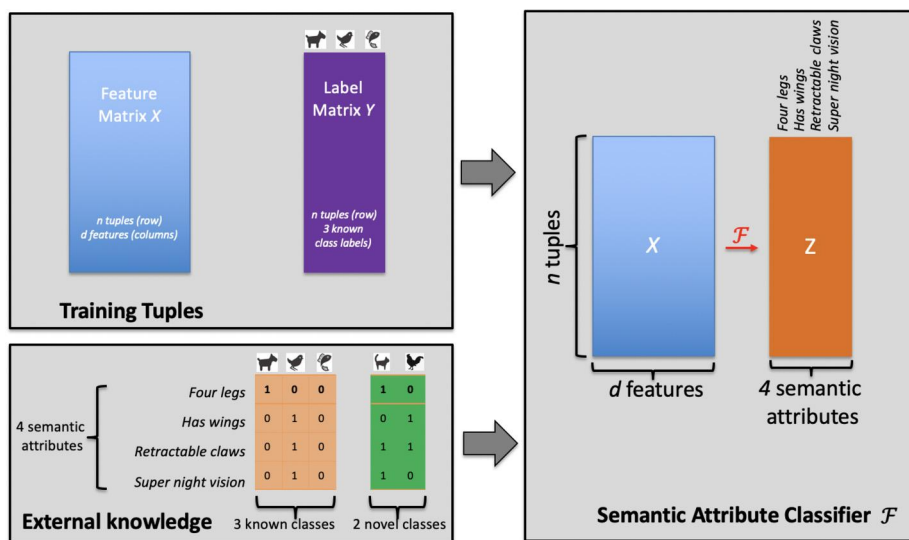


FIGURE 7.19

Top left: input  $n$  training tuples in  $d$ -dimensional feature space, each of which is labeled by one of the three known classes (i.e., “dog,” “owl,” and “fish”). Bottom left: external knowledge where each known and novel class is described by four semantic attributes. Right: the trained semantic attribute classifier  $\mathcal{F}$ .

knowledge about the class label, where each class label (animal) can be described by four *semantic attributes*,<sup>11</sup> including whether the animal “has four legs,” “has wings,” “has retractable claws,” and “has super night vision.” For example, since a dog has four legs, but no wings or retractable claws or super night vision, the class label “dog” can be described by a 4-D semantic attribute vector  $[1, 0, 0, 0]$ . Likewise, the class label “cat” can be described by a 4-D semantic attribute vector  $[1, 0, 1, 1]$ , meaning that a cat has four legs, retractable claws and super night vision but no wings. Notice that such external knowledge is available for both known class labels (e.g., “owl,” “dog,” and “fish”) and novel class labels (e.g., “cat” and “rooster”).

Then, using the input training tuples (i.e., the  $n \times d$  feature matrix  $X$  and the  $n \times 3$  label matrix  $Y$  in the upper left corner of Fig. 7.19) and the external knowledge about the three known class labels (i.e., the information about four semantic attributes for the three known class labels in the bottom left corner of Fig. 7.19), we train a *semantic attribute classifier*  $\mathcal{F}$ , which predicts a 4-D semantic attribute vector for an input image represented by a  $d$ -dimensional feature vector. In our example, the output of the semantic attribute classifier  $\mathcal{F}$  tells whether the given image has “four legs,” “wings,” “retractable claws” and “super night vision,” respectively. We can use a two-layer neural network to train such a

<sup>11</sup> In the literature, the semantic attribute is also referred to as semantic feature or semantic property or just attribute.

semantic attribute classifier, which will be introduced in Chapter 10.<sup>12</sup> Then, given a test image, we predict which of the two novel classes (i.e., “cat” and “rooster”) it belongs to based on the following two steps. First, given the  $d$ -D feature vector of the test image, we use the semantic attribute classifier  $\mathcal{F}$  to output a 4-D semantic attribute vector, whose elements indicate whether or not the test image has the corresponding semantic attributes. For example, if the semantic attribute classifier output a vector  $[1, 0, 0, 1]$ , it means that the classifier predicts that the test image (1) has four legs, (2) has no wings, (3) has no retractable claws, and (4) has super night vision. Second, we compare the predicted semantic attribute vector with the external knowledge about the two novel classes, respectively (i.e., the  $4 \times 2$  green (dark gray in print version) table in the middle bottom of Fig. 7.19). We predict that the test image belongs to the novel class whose semantic attribute vector is most similar to that of the test image. In our example, since the predicted semantic attribute vector  $[1, 0, 0, 1]$  is more similar to that of “cat”  $([1, 0, 1, 1])$  than that of “rooster”  $([0, 1, 1, 0])$ , we predict that it is an image about “cat.”

The key of the method described above is that we leverage the semantic attributes as a bridge to transfer the output of the semantic classifier that was trained on the known class labels to predict the novel class labels. From this perspective, we can also view zero-shot learning as a special form of transfer learning (i.e., to transfer the knowledge about the known class labels to novel classes). In addition to the semantic attribute, there are other forms of external knowledge that can be harnessed for zero-shot learning. An example is the class-class similarity between known and novel classes. In the animal image classification application mentioned above, we can train a multiclass classifier to predict which of the three known classes an image belongs to. Now, given a test image that comes from the novel class (either “cat” or “rooster”), the trained classifier predicts it belongs to “dog,” and if we know that “dog” is more similar to “cat” than “rooster,” it is safe to predict the test image is indeed a “cat,” rather than a “rooster.” In the standard zero-shot learning setting, we always assume that the test image must come from one of the novel classes. This assumption might be too strong in reality. For example, the test image might come from either known classes (dog, owl, or fish) or novel classes (cat or rooster). There have been research on *generalized zero-shot learning* to address such a more complicated setting. Other applications of zero-shot learning include neural activity recognition, where the classifier needs to recognize the word that a person is thinking about based her neural activity reflected on the fMRI image. In this application, the class labels are words. It is impossible to construct a training data set that covers all possible words that a human can think of. Zero-shot learning can effectively help extrapolate the classifier trained on a limited number of words (known class labels) to the unseen words during the training stage (i.e., the novel classes).

---

## 7.6 Classification with rich data type

The classification techniques we have seen so far assume the following setting. That is, given a training set, where each training tuple is represented by a feature (or attribute) vector and a class label, we build

---

<sup>12</sup> The input of this two-layer neural network is the  $d$ -D feature, the hidden layer corresponds to the four semantic attributes, and output layer corresponds to the three known class labels. Unlike a typical neural network, the model parameter for the second layer (from semantic attribute to the known class labels) can be directly obtained based on the external knowledge about four semantic attributes for the three known class labels.

a classifier that predicts the label of a test (unseen) tuple. Since each training tuple is represented as a feature vector, it can be viewed as a data point in the feature space (i.e., *spatial* data). Beyond spatial data, there are rich types of data from real-world applications, such as stream data, sequence (e.g., text), graph data, grid data (e.g., image), and spatial-temporal data (e.g., video). Many classification techniques have been developed for various types of data. In this section, we will see three case studies, including stream data classification (Section 7.6.1), sequence classification (Section 7.6.2), and graph data classification (Section 7.6.3). Deep learning techniques that will be introduced in Chapter 10 provide another powerful way for classification with rich data types, by automatically learning a feature representation of the input data (e.g., image, text, graph).

### 7.6.1 Stream data classification

Suppose a bank wants to develop a data mining tool to automatically detect fraudulent transactions. To start with, we could collect a large set of historical transactions that contain both legitimate and fraudulent transactions and use them as the training tuples to construct a classifier (e.g., SVMs, logistic regression, etc.). If the performance of the trained classifier is acceptable, we integrate it into the bank's IT system to classify future transactions as fraudulent vs. legitimate. When the classifier flags a fraudulent transaction, we will ask a bank expert to manually check if it is indeed a fraudulent transaction or a false positive (i.e., the transaction is actually a legitimate one). Once we receive such manual annotation from the bank expert, naturally, we might want to use it as the new training tuples to improve the classification accuracy. This is a new classification setting, namely *stream data classification*, where the transactions arrive sequentially at different times in a stream fashion,<sup>13</sup> and the classifier needs to (1) classify the new transactions upon their arrival and (2) be updated (i.e., re-trained) with the newly available labeled tuples (e.g., the new fraudulent transactions confirmed by the bank expert).

There are a number of challenges facing stream data classification. First, the new data tuples often arrive at high speed. Therefore simply using the newly obtained labeled tuples (e.g., newly found fraudulent transactions by the bank expert) together with the historical training tuples to re-train the underlying classifier might not be affordable, since the speed to re-train the classifier from scratch might be slower than the arrival rate of the new data tuples. Second, the length of the stream data (i.e., the total number of the transactions) is often very large and in theory could be *infinite*. For example, a bank could constantly receive new transactions over many years. Therefore it is impossible to store all the data tuples, and as such in stream data classification, it is often assumed that each data tuple can only be accessed once or a limited number of times. This is often referred to as *one-pass constraint*. Third, the characteristic of the underlying classes might change over time. For example, some fraudulent users might change their behaviors in order to bypass the current fraud detection tool. As such, some historical training tuples might become less relevant or even noisy for building an effective classifier to detect new fraudulent transactions. This is often referred to as *concept drifting*, that is, the concept (i.e., the class label that the classifier aims to learn) is evolving over time.

An effective method for stream data classification is based on ensemble, which works as follows (see Fig. 7.20 for an illustration). We partition the data stream into equal-sized chunks. Each chunk contains a training set  $(X_i, y_i)$  ( $i = 0, 1, \dots, k$ ), where  $X_i$  and  $y_i$  are the feature matrix and the label vector for

<sup>13</sup> Formally, a data stream is an ordered sequence of data tuples.

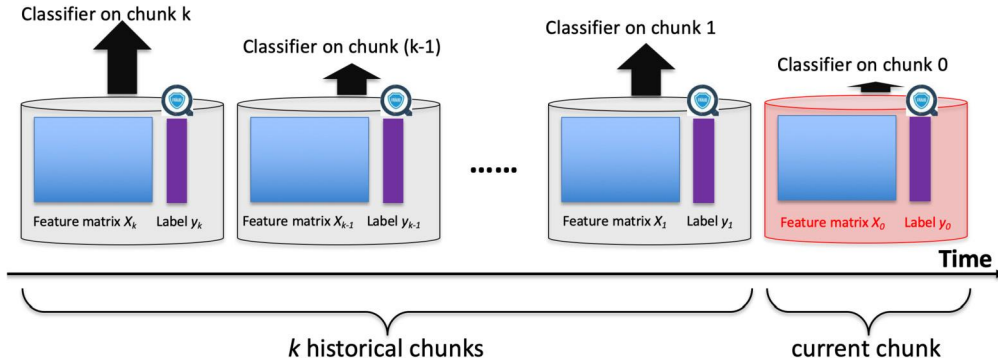


FIGURE 7.20

Ensemble based method for stream data classification. Each chunk trains an individual classifier and uses the most recent chunk (chunk 0) to estimate the classification error. The higher the classification error, the lower the weight in the ensemble. In the figure, the height of the black arrow indicates the classification error. The individual classifier on chunk  $k$  will be discarded since it has the highest error rate. The individual classifier on chunk 1 has a lower weight in the ensemble, compared with chunk  $(k - 1)$  or chunk 0.

the  $i$ th chunk; and  $i = 0, 1, \dots, k$  is the index of the chunks, with  $i = 0$  being the current (i.e., the most recent) chunk and  $i = k$  being the oldest chunk. For each of the  $k$  historical chunks (i.e.,  $i = 1, \dots, k$ ), we train a classifier  $f_i$  (e.g., Naive Bayes classifier), which outputs the posterior probability that the given tuple  $x$  belongs to the target class  $c$  (e.g., fraudulent transaction)  $f_i(x) = p_i(c|x)$ . Each classifier  $f_i$  is also associated with a weight  $w_i$ . Therefore the ensemble output of a test tuple  $x$  (i.e., the overall predicted probability that the given transaction  $x$  is fraudulent) is the weighted sum of the outputs of these  $k$  classifiers, that is,  $p(c|x) = \sum_{i=1}^k f_i(x)w_i$ . When a new chunk  $(X_0, y_0)$  arrives, we train a new classifier  $f_0$ . In the meanwhile, we use the newly arrived chunk as the test set to estimate the classification error of all  $(k + 1)$  classifiers  $f_i (i = 0, \dots, k)$ .<sup>14</sup> Among all  $(k + 1)$  classifiers, we select  $k$  classifiers with the lowest classification errors as the members of the ensemble. In other words, the classifier with the highest classification error rate will be discarded. In the meanwhile, for the  $k$  survived classifiers, we update their weights. The lower the classification error rate, the higher the weight. We can see that this method naturally addresses the three main challenges for stream data classification mentioned before. First, we only use the most recent chunk to train the new classifier  $f_0$ , which handles the high arrival rate of the stream data. Second, each incoming data tuple is accessed once to train the current classifier and to update the weights of individual classifiers. Third, by adjusting the weights of the individual classifiers, the ensemble pays more attention to the most relevant chunks (i.e., those individual classifiers with lower classification errors on the most recent chunk), so that it naturally captures the drifted class concept over time.

<sup>14</sup> For the newly constructed classifier  $f_0$ , we cannot directly use the new chunk  $(X_0, y_0)$  as the test set, since it is also used to train the classifier  $f_0$ . Otherwise, it will lead to an overoptimistic error rate. Instead, we can use the cross-validation technique on chunk  $(X_0, y_0)$  to estimate the error rate of the classifier  $f_0$ .



In addition to finance, stream data classification has also been applied to other application domains, such as marketing, network monitoring, and sensor networks. Many alternative learning strategies for stream data classification exist. For example, the very fast decision tree (VFDT) is built upon (a) the so-called Hoeffding trees, which build the decision tree (e.g., splitting an attribute on tree nodes) by using a sampled subset of training tuples and (b) a sliding window mechanism to obtain classifier that focuses on the most recent stream data. Another characteristic of stream data lies in its semisupervised nature. This means that the vast majority of the newly arrived data are unlabeled and only a handful of them are labeled. For example, in the fraud transaction examples, the bank expert might only be able to manually label a very small percentage (say 1%) of them. There has been research that develops semisupervised stream data classification based on ensemble methods.

### 7.6.2 Sequence classification

A **sequence** (i.e. sequential data) is an *ordered* list of values  $(x^1, x^2, \dots, x^T)$ , where  $x^t$  ( $t = 1, \dots, T$ ) is the value at a particular position or time stamp  $t$ , and  $T$  is the length of the sequence. The value  $x^t$  could be a categorical value (i.e., a symbol), or a numerical value or even a vector, or an itemset. Sequence naturally appears in many real applications. To name a few, in natural language processing, the sequence could be a sentence, where  $x^t$  is the  $t$ th word of the input sentence and  $T$  is the length (number of words) of the sentence; in genomic analysis, a sequence could be a DNA segment, where each value  $x^t$  is one of the four amino acid A, C, G, and T; in time series, the sequence could be a sequence of measurements at different time stamps, where  $x^t$  is one or more measurements (e.g., temperature, humidity) at the  $t$ th time stamp, and  $T$  is the total time stamps; in frequent pattern mining for market analysis, the sequence could be a list of transactions of a customer over time (i.e., each value  $x^t$  is the itemset the customer purchased at the corresponding time stamp  $t$ ). The goal of sequence classification is to build a classifier that predicts the label of a given sequence. The label could be the positive vs. negative sentiment of the sentence in natural language processing, the gene coding area vs. noncoding area in genomic analysis, or high-value vs. ordinary customer in market analysis.

One approach for sequence classification is through *feature engineering*. That is, we first convert the input sequence to a vector of features, which is in turn fed into a conventional classifier (e.g., decision tree, support vector machine). For symbolic sequence where each value  $x^t$  is a categorical value, we can use *n-gram* to construct the candidate features. Formally, an *n-gram* is a segment of sequence with  $n$  consecutive symbols. In our genomic analysis example, each 1-gram (also called unigram) is just one of the four amino acid (A, C, G, and T); and each 2-gram (also called bigram) is a sequence segment of two consecutive amino acid (e.g., AC, AG, GT, etc.). In natural language processing, each unigram could be a single word and a bigram is two consecutive words. For frequent sequence pattern mining, we can use pattern-based approach where each candidate feature is a frequent sequence pattern. Given a set of candidate features, a sequence can be converted to a feature vector, whose elements indicate the presence or absence of the corresponding candidate features in the input sequence. Alternatively, the elements of the feature vector could indicate the frequency of the corresponding features (i.e., how many times the given candidate feature appears in the input sequence). See Fig. 7.21 for an example. For a numerical sequence, we can first use discretization to convert it into a symbolic sequence, which might cause information loss. One potential issue with the feature engineering based approach is that the number of candidate features might be large, many of which are irrelevant to the classification task. Hence, feature engineering often works hand-in-hand with feature selection for sequence classification. Feature selection was introduced in Section 7.1.

	Unigrams				Bigrams															
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Candidate features	A	C	G	T	AC	AG	AT	CG	CT	CA	GA	GC	GT	TA	TC	TG	AA	CC	GG	TT
Feature vector (binary)	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
Feature vector (frequency)	1	5	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0	4	0	0

FIGURE 7.21

An example of using  $n$ -gram for sequence data feature engineering. Given a DNA segment “ACCCCGT,” we want to convert it to a feature vector, using  $n$ -grams. There are 20 candidate features in total, including 4 unigrams and 16 bigrams. For the binary feature vector (the third row), an element indicates whether the corresponding feature appears (1) or not (0) in the input sequence. For the weighted feature vector (the fourth row), an element indicates how many times (the frequency) that the corresponding feature appears in the input sequence.

Some classifiers we have introduced before rely on certain distance measures (e.g.,  $k$ -nearest neighbor classifier) or a kernel function (e.g., support vector machines) between different data tuples. Therefore, an alternative way for sequence classification is to define appropriate distance measures or a kernel function between different sequences.  $k$ -nearest neighbor classifier with commonly used distance measures (e.g., Euclidean distance) often leads to a competitive performance for sequence classification. There also exist distance measures and kernel functions that are specially designed for sequence data. For example, *dynamic time warping* distance (DTW) is a more robust distance measure than Euclidean distance with respect to the distortion in time; a commonly used kernel function for sequence data is called *string kernel*, which can be in turn fed into support vector machine for sequence classification.

In addition to predicting a class label for the entire sequence, some sequence classification task seeks to predict a label for each time stamp. For example, in natural language processing, we might want to predict whether each word of a given sentence is a specific type of named entity (e.g., location, person); in part-of-speech tagging, we need to predict whether each word is a pronoun or verb or noun. The key to sequence classification is to accurately model the *sequential dependence* among different values  $\mathbf{x}^t$  ( $t = 1, \dots, T$ ). A traditional method for modeling sequential dependence is called *Hidden Markov Model* (or HMM for short), which has a fundamental limitation, in that it assumes the future values ( $\mathbf{x}^{t+1}$ ,  $\mathbf{x}^{t+2}$ , ...) are independent of the past values ( $\mathbf{x}^1$ , ...,  $\mathbf{x}^{t-1}$ ) given the current value  $\mathbf{x}^t$  (known as the Markov assumption). A more powerful method to handle the sequential dependence is a specific type of deep learning technique called *Recurrent Neural Networks* which will be introduced in Chapter 10.

### 7.6.3 Graph data classification

Graph data (also referred to as network data), which is essentially a collection of *nodes* (or vertices) linked with each other by *edges*, is a ubiquitous data type arising in many applications, including social networks, power-grid, transaction network, biological networks, and more. The goal of graph data classification is to build a classifier to predict the label of either nodes (i.e., node-level classification) or the entire graphs (i.e., graph-level classification). An example of node-level classification is webpage classification. Given a web graph whose nodes are webpages and edges are hyperlinks from one

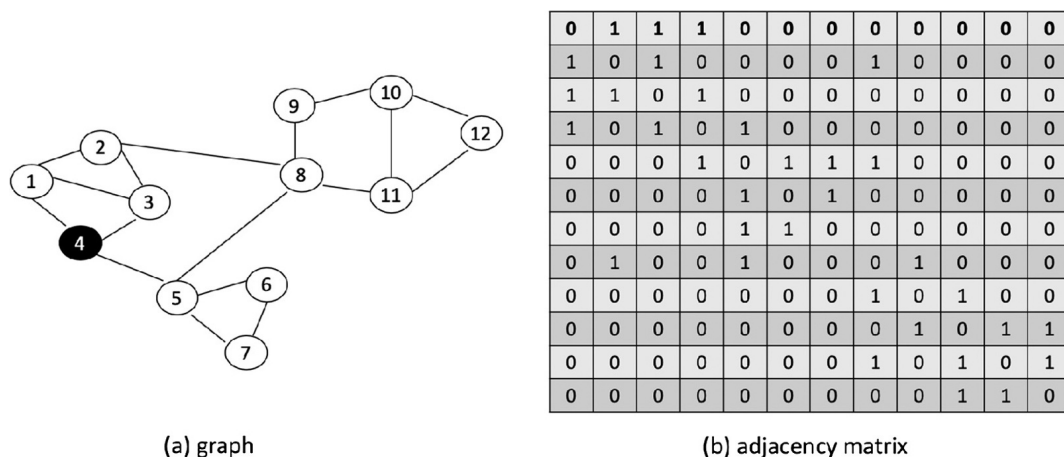


FIGURE 7.22

An example of a graph (a) and its adjacency matrix (b).

webpage to another, we want to determine the category (i.e., the label) of each webpage (i.e., a node in the webgraph). An example of graph-level classification is toxicity prediction. Given a collection of molecule graphs, we want to build a classifier to predict whether a given molecule graph is toxic (i.e., positive class label) or not (i.e., negative class label).

In a similar spirit of sequence classification, graph data classification can be done through feature engineering or proximity measures. For feature engineering based graph classification, we first extract a set of features describing each node or each graph, which are in turn fed into a conventional classifier (e.g., decision tree, logistic regression) to build a node-level or graph-level classifier. For node-level classification, commonly used node features include the number of neighboring nodes linked to the given node (i.e., node degree), the number of triangles the given node participates, the total edge weights of neighboring nodes (i.e., the weighted node degree), the node importance measure (e.g., eigen-centrality score, the PageRank score), the local clustering coefficient that measures to what extent the neighborhood of the given node looks like a full clique. For graph-level classification, commonly used graph features include the size of the graph (e.g., the number of nodes, edges, the total edge weights), the diameter of the graph, the total number of triangles in the graph, etc. More recent approaches often rely on a specific deep learning technique called *Graph Neural Networks* (which will be introduced in Chapter 10) to automatically extract node-level or graph-level features.

“How can we measure the proximity between two nodes or two graphs?” Let us first introduce the notation of *adjacency matrix*, which is an  $n \times n$  matrix for a graph with  $n$  nodes. The rows and columns of the adjacency matrix  $A$  represent the nodes. Given two nodes  $i$  and  $j$ , if there is a connection between them, we set the corresponding entries of the adjacency matrix as 1 (i.e.,  $A(i, j) = A(j, i) = 1$ ); otherwise, we set  $A(i, j) = A(j, i) = 0$ . We can also set entry  $A(i, j)$  as a numerical value to indicate the weight of the corresponding edge. Fig. 7.22 presents an example. An effective way to measure the node proximity is called **random walk with restart**. The algorithm is summarized in Fig. 7.23 and the steps are described next.

**Algorithm: Random walk with restart** for measuring node proximity.

**Input:**

- $A$ , adjacency matrix of the input graph of size  $n \times n$ ;
- $i$ , the query node;
- $0 < c < 1$ , a damping factor.

**Output:** The node proximity vector  $\mathbf{r}$  of size  $n \times 1$ .

**Method:**

- ```
//Preprocessing
(1) Set the query vector  $\mathbf{e}$  as an  $n \times 1$  vector, where  $e(i) = 1$  and  $e(j) = 0$  ( $j \neq i$ );
(2) Initialize proximity vector  $\mathbf{r} = \mathbf{e}$ ;
(3) Calculate the degree matrix  $\mathbf{D}$  of  $A$ ,
    where  $\mathbf{D}(i, i) = \sum_{j=1}^n A(i, j)$ ,  $\mathbf{D}(i, j) = 0$  ( $i \neq j, i, j = 1, \dots, n$ );
(4) Normalize  $\hat{A} = \mathbf{A}\mathbf{D}^{-1}$ ;
(5) while (termination condition is not satisfied){ // for each iteration
(6)     Update  $\mathbf{r} \leftarrow c\hat{A}\mathbf{r} + (1 - c)\mathbf{e}$ ;
(7) }
```

**FIGURE 7.23**

Random walk with restart for measuring the proximity between nodes on a graph.

Given a query node  $i$ , Fig. 7.23 produces a node proximity vector  $\mathbf{r}$  of length  $n$ , which contains the proximity measures from node  $i$  to other nodes in the graph. First (Step 1), we introduce a query vector  $\mathbf{e}$  that is an  $n \times 1$  vector, whose  $i$ th entry is 1 (i.e.,  $e(i) = 1$ ) and all other entries are zeros. We (Step 2) initialize the node proximity vector  $\mathbf{r}$  as the query vector  $\mathbf{e}$ . Then (Steps 3-4), we normalize the adjacency matrix so that each column of  $\hat{A}$  sums up to 1.<sup>15</sup> After that, we iteratively update the node proximity vector  $\mathbf{r}$  (Step 6) until a termination condition is met. At each iteration, we update the node proximity score for each node as follows. For the query node itself, its proximity score is updated as  $r(i) \leftarrow c \sum_{t=1}^n \hat{A}(t, i)r(t) + (1 - c)$ . That is, the updated proximity score for node  $i$  is a weighted sum of the proximity scores of its neighboring nodes, damped by the parameter  $c$  and then increased by an amount of  $(1 - c)$ . For each other node  $j$  ( $j \neq i$ ), its proximity score is updated as  $r(j) \leftarrow c \sum_{t=1}^n \hat{A}(t, j)r(t)$ . That is, the updated proximity score for node  $j$  is a weighted sum of the proximity scores of its neighboring nodes, damped by the parameter  $c$ . We repeat this process until a termination condition is met (e.g., a maximum iteration number is reached, or the difference between the node proximity vectors in two consecutive iterations is small enough).

**Example 7.7.** Let us apply random walk with restart algorithm in Fig. 7.23 to the graph in Fig. 7.22, and we aim to compute the proximity scores from the query node 4 to all other nodes. To this end, we set the query node  $\mathbf{e}$  as a vector of length 12, since there are 12 nodes in total, and the 4th entry of the query vector is 1 and all others are 0s. We initialize the proximity vector the same as the query vector  $\mathbf{r} = \mathbf{e} = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)'$  and normalize the adjacency matrix (Fig. 7.24(a)). In order to update the node proximity vector, we iteratively apply Step 7 of Fig. 7.23. Fig. 7.24(a) demonstrates the computation results of the first iteration. The final proximity scores and proximity vector are shown in

<sup>15</sup> Alternative normalization approaches exist. For example, for a directed graph, we can set  $\hat{A} = (\mathbf{D}^{-1}\mathbf{A})'$ , where the prime denotes matrix transpose. We can also use a symmetric normalization  $\hat{A} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ .

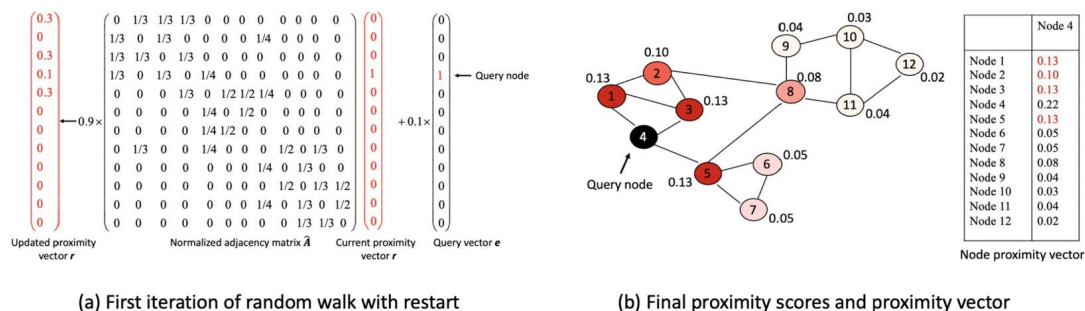


FIGURE 7.24

An example of applying random walk with restart to compute the proximity vector of query node 4. (a) The first iteration of the algorithm, where the damping factor  $c = 0.9$ . (b) The final proximity scores from node 4 to other nodes (more red (gray in print version) mean higher proximity scores) and proximity vector.

Fig. 7.24(b). We can see that the results are more or less consistent with our intuition, in that, nodes that are closer to the query node 4 (e.g., nodes 1, 2, 3, 5) receive higher proximity scores than others.  $\square$

“Why is random walk with restart a good node proximity measure?” Algorithm described in Fig. 7.23 is equivalent to the following random walk process. At the beginning, there is a random particle at the query node (e.g., node 4 in our example). At each iteration, the random particle does one of the following two things. First, it randomly surfs the graph. That is, it will randomly jump to one of its neighbors with the probability that is proportional to the edge weights between them. Second, it returns to the starting (i.e., query) node (hence the name “with restart”). It turns out the proximity score computed from Fig. 7.23 is equivalent to the *steady state probability* that the random particle will eventually stay at the corresponding node. Therefore if a node is closer to the query node, it will have a higher chance to attract the random particle to eventually stay there and hence receive a higher proximity score. Third, random walk with restart provides a good node proximity lies in its ability to summarize *multiple, weighted relationships* between nodes on graph. For the example in Fig. 7.24, the proximity between node 4 and node 8 by random walk with restart summarizes all the paths between these two nodes, with higher weights to those shorter, heavily weighted paths. Therefore if there exist multiple paths between two nodes and each path is short and heavily weighted, random walk with restart will assign a higher proximity score between.

Once we have a node proximity measure, we can apply it to *semisupervised node classification*. Given a graph and some (often a very limited number) labeled nodes, we can predict the labels of the remaining nodes as follows. If the average node proximity between a test node and all positively labeled nodes is higher than the average node proximity between the test node and all negatively labeled nodes, we predict it a positive node. Otherwise, we predict it a negative node.

Many variants of random walk with restart exist. For example, some methods measure the node proximity based on *commute time* or *hitting time*<sup>16</sup> from random walk perspective; some methods view

<sup>16</sup> Hitting time is the expected time for a random particle to start from the source node and reach the target node. Commute time is the expected time for a random particle to start from the source node, reach the target node, and then return to the source node.

the graph as an electric network and measure the node proximity based on *effective conductance*. Most of these methods share the similar idea as random walk with restart in that these methods all aim to summarize multiple, weighted relationship between nodes as the proximity measure. There also exist methods that generalize random walk with restart to attributed graphs, so that the node proximity considers both the topology of the graph and the attributes of nodes and edges. Random walk based approaches can also be generalized to measure the proximity between different graphs. For example, one method uses a similar mathematical formulation as random walk with restart, but defined over the *Kronecker graph* of the input graphs, to measure the similarity between two graphs. Such a similarity measure turns out to be a valid kernel function (i.e., random walk based graph kernel). Therefore we can feed such proximity measures into support vector machines for graph level classification.

---

## 7.7 Potpourri: other related techniques

Classification plays a very important role in data mining and has made tremendous progress in the past decades. As such, many techniques have been developed to address various aspects of classification. Classification is also closely related to other data mining tasks. In this section, we will learn some of these techniques. Most classification algorithms we have studied handle multiple classes, but some, such as support vector machines and logistic regression, are commonly used when only two classes exist in the data. What adaptations can be made to allow more than two classes? This question is addressed in Section 7.7.1 on *multiclass classification*. Some classifiers we have learned so far (e.g.,  $k$ -NN) rely on a distance (or similarity) measure between different data tuples. Section 7.7.2 introduces how to automatically learn a good distance metric for the classification task. In addition to classification accuracy, an increasingly important aspect is the interpretability of classification. It is highly desirable that the trained classifier not only predicts the class label of a test tuple but also helps the user understand why the classifier “thinks” the test tuple should have the predicted label. Section 7.7.3 introduces techniques to render interpretability to classification. As we have seen, many classification problems can be formulated from the optimization perspective, some of which are not easy to solve due to their combinatorial, nonconvex nature. Genetic algorithm is a powerful technique to handle combinatorial optimization problem, which will be introduced in Section 7.7.4. Finally, classification belongs to a specific type of supervised learning, where the classifier receives the *instructive feedback* (i.e., the true labels for the training tuples) in order to construct the best classifier. *Reinforcement learning* represents another type of supervised learning, where the learning agent receives *evaluative feedback* (e.g., the reward of an action taken at a specific time step instead of the true value of that action). Reinforcement learning is introduced in Section 7.7.5.

### 7.7.1 Multiclass classification

Some classification algorithms, such as support vector machines and logistic regression, are typically designed for binary classification. How can we extend these algorithms to allow for **multiclass classification** (i.e., classification involving more than two classes)?

A simple approach is **one-vs.-all** (OVA). Given  $m$  classes, we train  $m$  binary classifiers, one for each class. Classifier  $j$  is trained using tuples of class  $j$  as the positive class, and the remaining tuples as the negative class. It learns to return a positive value for class  $j$  and a negative value for the rest. To classify

an unknown tuple,  $\mathbf{X}$ , the set of classifiers vote as an ensemble. For example, if classifier  $j$  predicts the positive class for  $\mathbf{X}$ , then class  $j$  gets one vote. If it predicts the negative class for  $\mathbf{X}$ , then each of the classes except  $j$  gets one vote. The class with the most votes is assigned to  $\mathbf{X}$ .

**All-vs.-all** (AVA) is an alternative approach that learns a classifier for each pair of classes. Given  $m$  classes, we construct  $\frac{m(m-1)}{2}$  binary classifiers. A classifier is trained using tuples of the two classes it should discriminate. To classify an unknown tuple, each classifier votes. The tuple is assigned the class with the maximum number of votes. All-vs.-all tends to be superior to one-vs.-all.

A problem with the previous schemes is that binary classifiers are sensitive to errors. If any classifier makes an error, it can affect the vote count.

**Error-correcting codes** can be used to improve the accuracy of multiclass classification, not just in the previous situations, but for classification in general. Error-correcting codes were originally designed to correct errors during data transmission for communication tasks. For such tasks, the codes are used to add redundancy to the data being transmitted so that, even if some errors occur due to noise in the channel, the data can be correctly received at the other end. For multiclass classification, even if some of the individual binary classifiers make a prediction error for a given unknown tuple, we may still be able to correctly label the tuple.

An error-correcting code is assigned to each class, where each code is a bit vector. Fig. 7.25 shows an example of 7-bit codewords assigned to classes  $C_1$ ,  $C_2$ ,  $C_3$ , and  $C_4$ . We train one classifier for each bit position.<sup>17</sup> Therefore in our example we train seven classifiers. If a classifier makes an error, there is a better chance that we may still be able to predict the right class for a given unknown tuple because of the redundancy gained by having additional bits. The technique uses a distance measurement called the Hamming distance to guess the “closest” class in case of errors, and is illustrated in Example 7.8.

**Example 7.8. Multiclass classification with error-correcting codes.** Consider the 7-bit codewords associated with classes  $C_1$  to  $C_4$  in Fig. 7.25. Suppose that, given an unknown tuple to label, the seven trained binary classifiers collectively output the codeword 0001010, which does not match a codeword for any of the four classes. A classification error has obviously occurred, but can we figure out what the classification most likely should be? We can try by using the **Hamming distance**, which is the number of different bits between two codewords. The Hamming distance between the output codeword and the codeword for  $C_1$  is 5 because five bits—namely, the first, second, third, fifth, and seventh—differ. Similarly, the Hamming distance between the output code and the codewords for  $C_2$  through  $C_4$  are 3, 3, and 1, respectively. Note that the output codeword is closest to the codeword for  $C_4$ . That is, the

| Class | Error-correcting codeword |
|-------|---------------------------|
| $C_1$ | 1 1 1 1 1 1 1             |
| $C_2$ | 0 0 0 0 1 1 1             |
| $C_3$ | 0 0 1 1 0 0 1             |
| $C_4$ | 0 1 0 1 0 1 0             |

**FIGURE 7.25**

Error-correcting codes for a multiclass classification problem involving four classes.

<sup>17</sup> Conceptually, we can think of each bit as a semantic attribute in the zero-shot learning setting which was introduced in Section 7.5.5.

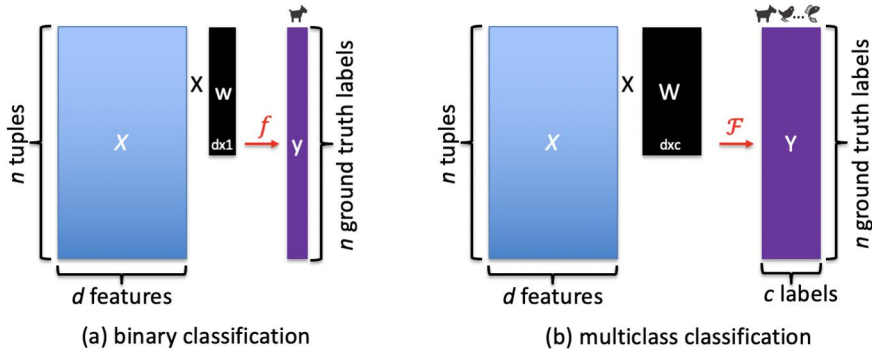


FIGURE 7.26

Comparison between binary vs. multiclass classification.

smallest Hamming distance between the output and a class codeword is for class  $C_4$ . Therefore, we assign  $C_4$  as the class label of the given tuple.  $\square$

Error-correcting codes can correct up to  $\frac{h-1}{2}$  1-bit errors, where  $h$  is the minimum Hamming distance between any two codewords. If we use one bit per class, such as for 4-bit codewords for classes  $C_1$  through  $C_4$ , then this is equivalent to the one-vs.-all approach, and the codes are not sufficient to self-correct. (Try it as an exercise.) When selecting error-correcting codes for multiclass classification, there must be good row-wise and column-wise separation between the codewords. The greater the distance, the more likely that errors will be corrected.

**Binary vs. Multiclass classification.** Let us use the example in Fig. 7.26 to explain the relationship between binary classification and multiclass classification. For clarity, we use linear classifiers, such as logistic regression or linear SVMs. Suppose that we are given an  $n \times d$  feature matrix  $X$  that represents  $n$  training images in  $d$ -D feature space. For the binary classification setting (Fig. 7.26(a)), we want to predict if a given image is a “dog” or not. Therefore, the class label for each training image can be represented as a binary scalar (e.g., 1 means the image is a dog, and  $-1$  means it is not). For the class labels for all  $n$  training tuples, we have an  $n \times 1$  label vector  $y$ . In order to train a linear classifier, we seek an optimal  $d$ -D weight vector  $w$  which minimizes the following loss function,  $w = \operatorname{argmin}_w \mathcal{L}(Xw, y) + \lambda \Omega(w)$ , where  $\mathcal{L}$  is a loss function that depends on the specific classifier,<sup>18</sup>  $\lambda > 0$  is a regularization parameter and  $\Omega(\cdot)$  is a regularization term of the weight vector  $w$ . A default choice for  $\Omega(w)$  could be the squared  $l_2$  norm of the weight vector  $w$ . Now, for the multiclass classification setting (Fig. 7.26(b)), we want to predict which of  $c$  classes (e.g., “dog,” “owl,” “fish,” etc.) a given image belongs to. Therefore, the class label for each training image can be represented as a  $c$ -dimensional label vector (e.g.,  $[-1, 1, \dots, -1]$  means that the image is an owl). For the class labels for all  $n$  training tuples, we have an  $n \times c$  label matrix  $Y$ . In order to train a linear classifier, we seek an optimal  $d \times c$  weight matrix  $W$  (instead of a vector) which minimizes the following loss function,

<sup>18</sup> For example,  $\mathcal{L}$  is the hinge loss function for linear SVMs and it is negative log likelihood function for logistic regression classifier.



$W = \operatorname{argmin}_W \mathcal{L}(XW, Y) + \lambda \Omega(W)$ , where the loss function  $\mathcal{L}$ , the regularization parameter  $\lambda$  and the regularization term  $\Omega(\cdot)$  have similar meanings as in the binary setting. Notice that in the multiclass setting, the classifier is expressed in the form of a  $d \times c$  weight matrix  $W$ . The predicted class label  $\tilde{y}$  for a test tuple  $\tilde{x}$  can be set as  $\tilde{y} = \operatorname{argmax}_i \tilde{x} \cdot W_i$ , where  $\cdot$  is the dot product between two vectors,  $W_i$  is the  $i$ th column of  $W$  and its elements measure the weights of the corresponding features for class label  $i$ . A default choice for the regularization term  $\Omega(\cdot)$  could be the squared Frobenius norm of the weight matrix  $W$ .

Multiclass classification is closely related to **multilabel classification** problem, where each data tuple could belong to one or more classes. For example, in document classification, each document can have multiple labels, each corresponding to a specific category or tag the document belongs to. Let  $L$  be the total number of classes. A natural way to handle multilabel classification is to train  $L$  independent binary classifiers, one for each class label. That is, the  $l$ th classifier predicts whether or not the given data tuple has the class label  $l$  ( $l = 1, \dots, L$ ). Note that this method bears subtle difference from the one-vs.-all or all-vs.-all methods for multiclass classification problem introduced before. For the former (multilabel classification), a given data tuple can belong to more than one class since  $L$  classifiers are independently trained and applied. For the latter (multiclass classification), we always assign a single label (out of  $L$  possible labels) to a data tuple, by voting. Another method for multilabel classification with  $L$  labels is to convert it to a multiclass classification problem with  $(2^L - 1)$  labels. This process is called *label powerset transformation*. For example, for a multilabel classification problem with three possible labels, namely  $A$ ,  $B$  and  $C$ , we can convert it to a multiclass classification problem with the following seven possible labels, namely  $A$ ,  $B$ ,  $C$ ,  $AB$ ,  $AC$ ,  $BC$ , and  $ABC$ . In other words, each of the seven newly constructed labels corresponds to a subset of the original three labels that is assigned to a data tuple.

In Chapter 10, we will introduce deep learning techniques, which can naturally handle multiclass classification problem by introducing one node for each class label in the output layer.

### 7.7.2 Distance metric learning

Some classifiers (e.g.,  $k$ -nearest-neighbor classifiers) rely on a distance measure. In Section 6.4, we have learned that even with the same training tuples and the same choice of  $k$ , different distance metrics (e.g.,  $L_1$  vs.  $L_2$ ) might lead to quite different decision boundaries. *So, is there a way that we can automatically learn the best distance metrics for a given classification task?* Distance metric learning (or metric learning) aims to answer this question.

A commonly used distance metric is Euclidean distance (also referred to as  $L_2$  distance), which is often the default choice for many classifiers. Given two data tuples in  $p$ -dimensional space  $X_1 = (X_{1,1}, X_{1,2}, \dots, X_{1,p})'$  and  $X_2 = (X_{2,1}, X_{2,2}, \dots, X_{2,p})'$ , the Euclidean distance between them is defined as  $d(X_1, X_2) = \sqrt{\sum_{i=1}^p (X_{1,i} - X_{2,i})^2} = \sqrt{(X_1 - X_2)'(X_1 - X_2)}$ , where  $'$  denotes the transpose of a vector. We compare the difference  $(X_{1,i} - X_{2,i})$  between the two input data tuples along each of the  $p$  dimensions, sum the squared difference over all the  $p$  dimensions, and take the square root of such a summation as the Euclidean distance. In other words, the different dimensions have the *equal* weights on the overall distance between two data tuples, and their effects are considered *independently*. Therefore if certain dimension has a larger value range than others, or if different dimensions are correlated with each other, Euclidean distance might lead to suboptimal classification performance. To overcome the limitations of Euclidean distance, a more flexible and powerful distance is called Mahalanobis dis-

tance, which is defined as follows:

$$d_M(X_1, X_2) = \sqrt{(X_1 - X_2)'M(X_1 - X_2)} = \sqrt{\sum_{i,j=1}^p (X_{1,i} - X_{2,i})M(i, j)(X_{1,j} - X_{2,j})}, \quad (7.24)$$

where the prime symbol  $'$  denotes the matrix transpose and  $M$  is a  $p \times p$  symmetric positive semidefinite matrix.<sup>19</sup> Compared with Euclidean distance, Mahalanobis distance naturally (1) assigns different weights (through the diagonal elements of matrix  $M$ ) to different dimensions and (2) incorporates the interaction effect of different dimensions (through the off-diagonal elements of matrix  $M$ ) in measuring the distance between the two input tuples.

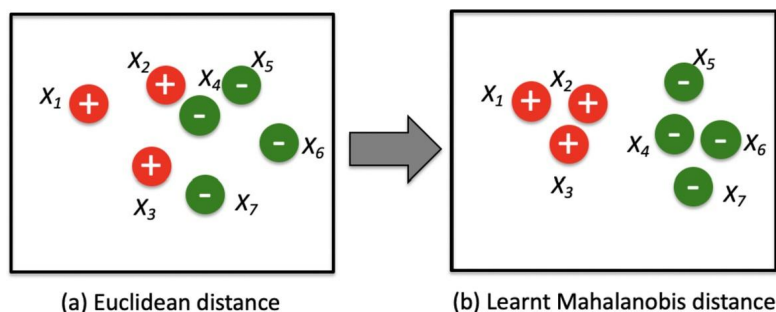
Depending on the specific choice of the  $M$  matrix, the Mahalanobis distance between two data tuples will vary. Therefore the goal of distance metric learning becomes learning the optimal  $M$  matrix for a given classification task. Suppose there are  $n$  labeled training tuples in  $p$ -D space. Let  $\mathcal{S}$  contain all the training *tuple pairs* which are similar with each other (e.g., each member in  $\mathcal{S}$  is a pair of training tuples which share the same class label). Let  $\mathcal{D}$  contain all the training *tuple pairs* which are dissimilar with each other (e.g., each member in  $\mathcal{D}$  is a pair of training tuples with different class labels). The basic idea of distance metric learning is that we want to find the optimal  $M$  matrix (hence the optimal Mahalanobis distance) so that (1) the distance between any pair of *similar* tuples in  $\mathcal{S}$  is small and (2) the distance between any pair of *dissimilar* tuples in  $\mathcal{D}$  is large (see Fig. 7.27 for an illustration). Mathematically, we can formulate it as the following optimization problem.

$$\begin{aligned} & \max_{M \in R^{p \times p}} \sum_{(X_i, X_j) \in \mathcal{D}} d_M^2(X_i, X_j) \\ \text{s.t.} \quad & \sum_{(X_i, X_j) \in \mathcal{S}} d_M^2(X_i, X_j) \leq 1, \quad M \geq 0 \end{aligned} \quad (7.25)$$

$M \geq 0$  means that matrix  $M$  is positive semidefinite. The optimization problem defined in Eq. (7.25) is convex. Therefore we can resort to the off-the-shelf optimization software to solve it. We will not go into such details, which are out of the scope of this book.

The objective function as well as constraints in Eq. (7.25) are in the form of a pair of training tuples. In fact, the above formulation can be viewed as a binary classification problem, where the input is a *pair of training tuples* whose class label is  $+1$  if the pair comes from  $\mathcal{S}$  (i.e., they share the same class label), and  $-1$  if the pair comes from  $\mathcal{D}$  (i.e., they have different class labels). Alternative formulations for distance metric learning exist. For example, in the ranking task, the supervision might be in the form that certain tuples should be ranked higher than other tuples for a given query. The distance metric learning with such kind of supervision can be formulated with respect to the *triples* of training tuples (e.g., the query tuple, a high-ranked tuple, and a low-ranked tuple). In Eq. (7.25), we work

<sup>19</sup> Mathematically, a  $p \times p$  matrix  $M$  is positive semidefinite if  $X'MX \geq 0$  for any  $X \in R^p$ . This is to ensure that  $d_M(X_1, X_2)$  defined in Eq. (7.24) is a valid distance metric. Since  $M$  is symmetric and positive semidefinite, we can decompose  $M$  as  $M = L'L$ , where  $L$  is a  $r \times p$  matrix. In this way, we can re-write the Mahalanobis distance as  $d_M(X_1, X_2) = \sqrt{(LX_1 - LX_2)'(LX_1 - LX_2)}$ . Therefore the Mahalanobis distance can be interpreted as the following process. That is, we first perform a linear feature transformation of the input tuples through the  $r \times p$  matrix  $L$  (i.e.,  $X_j \rightarrow LX_j$  ( $j = 1, 2$ )); and then, we calculate the Euclidean distance between the two transformed data tuples in the  $r$ -D space.



**FIGURE 7.27**

An illustrative example of distance metric learning. Given three positive training tuples ( $X_1$ ,  $X_2$ ,  $X_3$ ) and four negative training tuples ( $X_4$ ,  $X_5$ ,  $X_6$ ,  $X_7$ ). Left: Euclidean distance. Right: Learnt Mahalanobis distance, where the distance between tuple pairs of the same class label is smaller than that of different class labels.  $\mathcal{S} = \{(X_1, X_2), (X_1, X_3), (X_3, X_2), (X_4, X_5), (X_4, X_6), (X_4, X_7), (X_5, X_6), (X_5, X_7), (X_6, X_7)\}$ , and  $\mathcal{D} = \{(X_1, X_4), (X_1, X_5), (X_1, X_6), (X_1, X_7), (X_2, X_4), (X_2, X_5), (X_2, X_6), (X_2, X_7), (X_3, X_4), (X_3, X_5), (X_3, X_6), (X_3, X_7)\}$ .

with the squared Mahalanobis distance, which makes the objective function as well as its constraints be linear with respect to matrix  $M$ . This type of methods is often referred to as linear distance metric learning. Nonlinear distance metric learning methods exist. For example, we can *kernelize* a linear distance metric learning method in the similar way as how we make a linear SVM classifier be a nonlinear in Section 7.3. Alternatively, some nonlinear distance metric methods learn *multiple* local linear metrics (e.g., one linear Mahalanobis distance for tuples in a given cluster). In terms of computation of Eq. (7.25), a major bottleneck lies in the constraint that the matrix  $M$  must be positive semidefinite (i.e.,  $M \succeq 0$ ). If we drop such a constraint, Eq. (7.25) becomes *similarity learning* problem, which can often be solved faster than distance metric learning. Beyond the standard classification, distance metric learning has also been applied to classification with weak supervision (e.g., semisupervised learning, transfer learning) and other data mining tasks (e.g., ranking, clustering).

### 7.7.3 Interpretability of classification

So far, we have primarily focused on the accuracy of the classification models. Indeed, the field of data mining, machine learning and AI has witnessed tremendous progress on improving classification accuracy. For some application domains (e.g., computer visions, natural language processing), sophisticated classification models (such as deep learning techniques that will be introduced in Chapter 10) can now achieve an accuracy that is comparable or even surpasses humans on a variety of classification tasks. That said, the accuracy alone is often not sufficient for many application scenarios. For examples, how can the end user (e.g., the manager of an electronics store) understand the classification results by an SVM classifier? If a newly developed classification model improve the sentiment classification accuracy by 10% over the existing classifier, can we *trust* the results, that is, is the 10% improvement due to the new model's capability to capture the hidden feature in relation to the sentiment that was ignored by the existing classifier, or it is just due to the random noise? The answers to such questions lie in the

*interpretability*, which describes the models' ability to explain the classification results or process in a user understandable way.

Interpretability naturally comes with some of the classification models we have learned so far. To name a few, for a decision tree classifier, the path from the root to the leaf node that the test tuple belongs to provide interpretation in terms of why the classifier predicts certain class label for the given test tuple. For the example in Section 6.1 (Table 6.1 and Fig. 6.5), such an interpretation could be that “since the individual is *young*, with *medium income* and an *excellent credit rating*, the classifier predicts she will purchase a computer.” Likewise, the rule antecedent in rule-based classification models could provide similar interpretation on why the classifier predicts a certain class label for a tuple. However, when the decision tree becomes deeper or the rule antecedent becomes longer, this type of interpretation becomes less effective. For linear classifiers (e.g., perceptron, logistic regression), the decision boundary of the classifier is in the form of  $\sum_{i=1}^p w_i x_i = 0$ , where  $w_i$  is the weight of the  $i$ th attribute. Therefore both the magnitude and the sign of the weight  $w_i$  provides an interpretation on the impact or contribution of the corresponding attribute in making the class prediction. For high-dimensional data with a large number of attributes, it often leads to more effective interpretation if the linear classifier is combined with the feature selection (e.g., LASSO introduced in Section 7.1), so that we can focus on a few most important attributes to interpret the classification results. However, we cannot directly use this strategy to interpret nonlinear classification models, such as Bayesian belief networks, nonlinear SVMs, deep neural networks.

The interpretation methods mentioned above (e.g., the path in a decision tree, the rule antecedent, the weights in linear classifiers) are *model-specific*, in the sense that the interpretation is designed for a specific classification model and we have the full access to the details of that model. But in some cases, the end user might only have the access to a black-box classification model  $f$ , which predicts a class label  $y$  for a given test tuple  $x$ . However, the details (e.g., what kind of classification model, or its parameters) are unknown to the end user. How can we provide *model-agnostic* interpretation for such a black-box model? An effective way is to use a *proxy* or *surrogate* model  $g$ , which itself is interpretable (such as a shallow decision tree or a sparse linear classifier), to approximate the black-box classification model classification  $f$  in the local vicinity of a given test tuple which we wish to interpret.<sup>20</sup> LIME (Local Interpretable Model-agnostic Explanation) is such a model-agnostic method. Let us introduce its key idea using the example in Fig. 7.28. In Fig. 7.28(a), there is a black-box binary classification model  $f$  with a complicated decision boundary, which classifies the shaded area as positive labels and the remaining as the negative labels. We have a test tuple (the black dot) to interpret, that is, to help the end-user understand why the black-box model  $f$  predicts its class label as positive. To this end, LIME samples a few data tuples in the local vicinity of the test tuple (i.e., the purple (mid gray in print version) circle in Fig. 7.28(b)). For each sampled data tuple, it is assigned a binary class label, which is the prediction of the black-box model  $f$ . Each sampled tuple is also assigned a weight, which is in reverse proportion to its distance to the test tuple, the closer to the test tuple, the higher the weight (indicated by the size of sampled tuples in Fig. 7.28(b)). Then, LIME trains a surrogate model  $g$  using the weighted sample tuples (e.g., a linear classifier in Fig. 7.28(c)). LIME uses the surrogate model  $g$  to interpret the classification of the test tuple by the black-box model  $f$ . In this example, the test

<sup>20</sup> There exist methods to use surrogate models to approximate  $f$  in the entire feature space (i.e., regardless of which test tuple we wish to interpret). However, these methods are less common, since it is very hard to find an interpretable surrogate model which can approximate the black-box classification model *globally*.

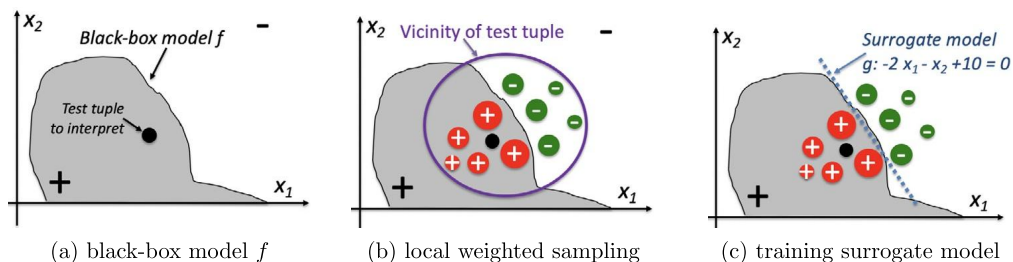


FIGURE 7.28

An example of LIME: Local Interpretable Model-agnostic Explanation.

tuple is classified by the black-box model as a positive tuple and the decision boundary of the surrogate model  $g$  is  $-2x_1 - x_2 + 10 = 0$ . Therefore we have the following interpretation: “the tuple is classified as positive by the black-box model  $f$ . This is because in its local vicinity, (1) both attributes ( $x_1$  and  $x_2$ ) have a negative impact on the positive class label (the larger  $x_1$  and  $x_2$ , the less likely the test tuple belongs to the positive class; and (2) the effect of the first attribute  $x_1$  is twice that of the second attribute  $x_1$ .”

When we train the local surrogate model  $g$ , we use the *predicted label* by the black-box model  $f$  as the labels of the sampled tuples. This is because we want to ensure the *local fidelity* of the surrogate model, which measures how well the surrogate model  $g$  approximates the black-box model in the local vicinity of the test tuple  $x$ . Meanwhile, we want to keep the surrogate model  $g$  to be “simple” so that it is interpretable to the end users. To this end, the surrogate model  $g$  often uses a set of interpretable features (e.g., the actual words in text classification, the region or superpixel in image classification) which are different from the features used by the black-box model. The key in LIME is to strike a good trade-off between the *local fidelity* and the *model complexity*. The model complexity (e.g., the depth of the decision tree model, the number of selected features in the sparse linear classifiers) is in reverse proportion to the interpretability of the surrogate model  $g$ : a more complex model is less interpretable.

Alternative approaches to interpret the classification models exist. For example, *counterfactual explanation* interprets the prediction on a test tuple by identifying the optimal changes to its attributes (e.g., deleting certain words for text classification, changing the superpixel for image classification, perturbing the continuous attribute values), which would alter the prediction results of the classification model. Another powerful interpretation technique is to use *influence function*, which is originated from the robust statistics. By influence function analysis, one could identify the most *influential* or important training tuples, which, if perturbed, would significantly alter the classification model (e.g., the weight vector in a linear classifier).

Beyond classification, interpretability also plays an important role in other data mining tasks, including clustering, outlier detection, ranking, and recommendation. In addition to making the data mining models transparent so as to gain the users’ trust of the model, interpretability is also intimately related to other important aspects of data mining, such as fairness, robustness, causality, privacy.

### 7.7.4 Genetic algorithms

**Genetic algorithms** attempt to incorporate ideas of natural evolution. In general, genetic learning starts as follows. An initial **population** is created consisting of randomly generated rules. Each rule can be represented by a string of bits. As a simple example, suppose that samples in a given training set are described by two Boolean attributes,  $A_1$  and  $A_2$ , and that there are two classes,  $C_1$  and  $C_2$ . The rule “*IF  $A_1$  AND NOT  $A_2$  THEN  $C_2$* ” can be encoded as the bit string “100,” where the two leftmost bits represent attributes  $A_1$  and  $A_2$ , respectively, and the rightmost bit represents the class. Similarly, the rule “*IF NOT  $A_1$  AND NOT  $A_2$  THEN  $C_1$* ” can be encoded as “001.” If an attribute has  $k$  values, where  $k > 2$ , then  $k$  bits may be used to encode the attribute’s values. Classes can be encoded in a similar fashion.

Based on the notion of survival of the fittest, a new population is formed to consist of the *fittest* rules in the current population, as well as *offspring* of these rules. Typically, the **fitness** of a rule is assessed by its classification accuracy on a set of training samples. Offspring are created by applying genetic operators such as crossover and mutation. In **crossover**, substrings from pairs of rules are swapped to form new pairs of rules. In **mutation**, randomly selected bits in a rule’s string are inverted. The process of generating new populations based on prior populations of rules continues until a population,  $P$ , evolves where each rule in  $P$  satisfies a prespecified fitness threshold. Furthermore, genetic algorithms are easily parallelizable and have been used for classification, feature selection as well as other optimization problems. In data mining, they may be used to evaluate the fitness of other algorithms.

### 7.7.5 Reinforcement learning

Suppose that the advertisement department at an electronics store has a daily budget to do a single advertisement each day for one of the three products: TV, camera, or printer. Which product shall you choose to advertise on each day?

Let us first introduce some notation. Here, we have three possible *actions*  $a$ :  $a \in \{TV, camera, printer\}$ , representing which product will be chosen for advertisement on a particular day. Let us assume that each action has a fixed *value*  $q(a)$ . The value  $q(a)$  measures the *expected reward* if the action  $a$  is taken:  $q(a) = E(R|a)$ , where  $R|a$  is the reward (such as the increased revenue) given that an action (an advertisement on the corresponding product) is taken. Notice that the reward itself is a random variable since the actual increased revenue on different days might vary, even with the same advertisement. Therefore we use its expectation to measure the value of the corresponding action, that is, the increased revenue *on average* that the corresponding action (advertisement) brings. If we know the value  $q(a)$ , we can just choose the action (advertisement) with the highest value, since it brings the highest (expected) rewards. However, in reality, such values are unknown. Now, what shall we do? It turns out we can resort to a powerful computational method called **reinforcement learning**, which learns through interaction.

In this example, we might decide to try advertisements with different products on different days, observe the *actual* rewards on these days, and then adjust the advertisement strategy for the future days accordingly. Suppose we try the camera advertisement for 4 days and TV advertisement for 1 day, and we observe the rewards in these 5 days, which are summarized in Fig. 7.29(a). Based on these observed rewards, we can calculate an *estimate value*  $Q(a)$  for each action (summarized Fig. 7.29(b)). Therefore one possible strategy is that we just take the action with the highest estimated value  $Q(a)$  for the next day (i.e., TV in this case). This strategy is *greedy* in the sense that it tries to make the best

|         | Day 1  | Day 2  | Day 3 | Day 4  | Day 5  |
|---------|--------|--------|-------|--------|--------|
| Action  | camera | camera | TV    | camera | camera |
| Rewards | 20     | 20     | 100   | 150    | 10     |

(a) actions and rewards

|                        | TV  | Camera | Printer |
|------------------------|-----|--------|---------|
| Value $q(a)$           | ?   | ?      | ?       |
| Estimated value $Q(a)$ | 100 | 50     | ?       |

(b) estimated values

**FIGURE 7.29**

An example of reinforcement learning.

use (*exploitation*) of the information we have collected so far regarding the true value  $q(a)$ . However, what if there is a big gap between the estimated value  $Q(a)$  and the true value  $q(a)$ ? In other words, it is quite possible that  $q(\text{camera}) > q(\text{TV})$ , even though  $Q(\text{camera}) < Q(\text{TV})$ , especially given that  $Q(\text{camera})$  and  $Q(\text{TV})$  are based on very limited data in this example (4 days for camera and 1 day for TV). Moreover, we have never tried printer advertisement at all and thus have zero knowledge about  $q(\text{printer})$ . What if the printer advertisement actually has the highest value?

Another alternative strategy is the random strategy (*exploration*). Each day, we choose a random product to advertise and observe the reward of that advertisement. If we keep running this for many days, the estimated value  $Q(a)$  is likely to be very close to the true value  $q(a)$ . After that, we choose the advertisement with the highest (estimated) value. This strategy is optimal in the long run. However, we might spend many days *before* we figure out the optimal action, during which the received rewards might be low.

A better strategy is to combine the greedy strategy and the random strategy together. That is, at each day, with the probability  $1 - \epsilon$  ( $0 < \epsilon < 1$ ), we choose the action (advertisement) with the highest estimated value  $Q(a)$ ; with the probability  $\epsilon$ , we choose a random action; and at the end of the day, we use the observed reward to update the estimated value  $Q(a)$ . This strategy (called  $\epsilon$ -greedy) is likely to obtain a better balance between the immediate (exploitation) and the long-term (exploration) return.

In reinforcement learning, the learning agent tries to figure out what to do (e.g., choose the best product to advertise in order to maximize the overall increased revenue) by interacting with the environment (e.g., trying a few different advertisements, observing their rewards, and adjusting the advertisement for next day accordingly). This is related to, but bears subtle difference from, classification. In classification, classifier receives the *instructive feedback* (i.e., the true labels for the training tuples) in order to construct the best classifier. In reinforcement learning, the learning agent receives *evaluative feedback* (e.g., the immediate reward of an action taken on a specific time step) in order to find the best action.

The above example is called *multiarmed bandit problem*, in that we can imagine a slot machine with multiple arms, and each arm corresponds to an action (an advertisement on a product) with an unknown value. Beyond the  $\epsilon$ -greedy algorithm, many alternative methods exist. For example, *upper-confidence-bound* method selects the action at each time step based on both the estimated value  $Q(a)$  and the uncertainty (or confidence) of the estimation on  $Q(a)$ . Multiarmed bandit problem is a classic setting in reinforcement learning, where the learning agent tries to find out the best action to act in a single situation (e.g., each advertisement has a fixed value). In more complicated setting, the learning agent needs to choose different actions in different situations. In such settings, reinforcement learning is often formulated as a (finite) Markov decision process. Reinforcement learning has been applied in many application domains, including online advertisement, robotics, chess, and many more.

---

## 7.8 Summary

- **Feature selection** aims to select a few most powerful features from a set of initial features. Typical methods include *filter methods*, *wrapper methods*, and *embedded methods*. **Feature engineering** aims to construct more powerful features based on the initial features.
- Unlike naïve Bayesian classification (which assumes class conditional independence), **Bayesian belief networks** allow class conditional independencies to be defined between subsets of variables. They provide a graphical model of causal relationships, on which learning can be performed. Trained Bayesian belief networks can be used for classification.
- A **support vector machine** is an algorithm for the classification of both linear and nonlinear data. It transforms the original data into a higher dimensional space, from which it can find a hyperplane for data separation using essential training tuples called **support vectors**.
- A **rule-based classifier** uses a set of IF-THEN rules for classification. Rules can be extracted from a decision tree. Rules may also be generated directly from training data using sequential covering algorithms. *Frequent patterns* reflect strong associations between attribute–value pairs (or items) in data and are used in **classification based on frequent patterns**. Approaches to this methodology include associative classification and discriminant frequent pattern–based classification. In **associative classification**, a classifier is built from association rules generated from frequent patterns. In **discriminative frequent pattern–based classification**, frequent patterns serve as combined features, which are considered in addition to single features when building a classification model.
- **Semisupervised classification** is useful when large amounts of unlabeled data exist. It builds a classifier using both labeled and unlabeled data. Examples of semisupervised classification include *self-training* and *cotraining*.
- **Active learning** is a form of supervised learning that is suitable for situations where data are abundant, yet the class labels are scarce or expensive to obtain. The learning algorithm can actively query a user (e.g., a human annotator) for labels. To keep the cost down, the active learner aims to achieve high accuracy using as few labeled instances as possible.
- **Transfer learning** aims to extract the knowledge from one or more *source tasks* and apply the knowledge to a *target task*. TrAdaBoost is an example of the *instance-based approach* to transfer learning, which reweights some of the data from the source task and uses it to learn the target task, thereby requiring fewer labeled target-task tuples.
- **Distant supervision** automatically generates a large number of noisy labeled tuples based on external knowledge or side information.
- **Zero-shot learning** builds a classifier that predicts a test tuple whose class label was never observed during the training stage. An example of zero-shot learning is based on *semantic attribute classifier*.
- In many applications, data tuples arrive in a stream fashion. The key challenges of **stream data classification** include scalability, one-pass constraint and concept drifting.
- A **sequence** is an *ordered* list of values. The goal of **sequence classification** is to build a classifier to predict the label of the entire sequence or each time stamp of the sequence. Approaches to sequence classification include feature engineering–based methods and distance measure–based methods.
- The goal of **graph data classification** is to build a classifier to predict the label of either nodes or entire graphs. Similar to sequence classification, graph data classification can be done through feature engineering or proximity measures.



- Binary classification schemes, such as support vector machines, can be adapted to handle **multiclass classification**. This involves constructing an ensemble of binary classifiers. Error-correcting codes can be used to increase the accuracy of the ensemble.
- **Distance metric learning** aims to learn the best distance metrics for a given classification task. The basic idea is to find the optimal distance metrics so that the distance between similar tuples is small, whereas the distance between dissimilar tuples is large.
- In **genetic algorithms**, populations of rules “evolve” via operations of crossover and mutation until all rules within a population satisfy a specified threshold.
- **LIME** (Local Interpretable Model-agnostic Explanation) is a model-agnostic interpretation method. It finds a surrogate model in the local vicinity of the test tuple that balances the model fidelity and the model complexity.
- In classification, the learning agent (i.e., classifier) receives the *instructive feedback* in order to construct the best classifier. In **reinforcement learning**, the learning agent receives *evaluative feedback* in order to find the best action. Effective reinforcement learning methods need to strike a balance between *exploitation* and *exploration*.

---

## 7.9 Exercises

- 7.1.** Feature selection aims to select a subset of features that will be used in training. In general, there are three major types of feature selection strategy: filter method, wrapper method and embedded method.
- Which type of feature selection strategy does Fisher Score belong to? Please justify your answer using 1–2 sentences.
  - Suppose we have six training examples shown below. Calculate Fisher scores for  $\theta_0$  and  $\theta_1$  and find out which one is more discriminative.

| example | $\theta_0$ | $\theta_1$ | label |
|---------|------------|------------|-------|
| 1       | 96         | 33         | -     |
| 2       | 86         | 30         | +     |
| 3       | 78         | 29         | +     |
| 4       | 92         | 36         | -     |
| 5       | 80         | 35         | +     |
| 6       | 90         | 32         | +     |

- Which type of feature selection strategy does LASSO belong to? Please justify your answer using 1–2 sentences.
  - Denoting the vector of feature coefficients as  $\mathbf{w}$ , suppose we have a tuning parameter  $\lambda$  for LASSO (i.e. the LASSO penalty term is  $\lambda\|\mathbf{w}\|_1$ ). If  $\lambda$  goes to infinity, what would happen to  $\mathbf{w}$ ? Why?
- 7.2.** The *support vector machine* is a highly accurate classification method. However, SVM classifiers suffer from slow processing when training with a large set of data tuples. Discuss how to

overcome this difficulty and develop a scalable SVM algorithm for efficient SVM classification in large data sets.

- 7.3.** Compare and contrast *associative classification* and *discriminative frequent pattern-based classification*. Why is classification based on frequent patterns able to achieve higher classification accuracy in many cases than a classic decision tree method?
- 7.4.** Example 7.8 showed the use of error-correcting codes for a *multiclass classification* problem having four classes.
- Suppose that, given an unknown tuple to label, the seven trained binary classifiers collectively output the codeword 0101110, which does not match a codeword for any of the four classes. Using error correction, what class label should be assigned to the tuple?
  - Explain why using a 4-bit vector for the codewords is insufficient for error correction.
- 7.5.** *Semisupervised classification*, *active learning*, and *transfer learning* are useful for situations in which unlabeled data are abundant.
- Describe *semisupervised classification*, *active learning*, and *transfer learning*. Elaborate on applications for which they are useful, as well as the challenges of these approaches to classification.
  - Research and describe an approach to semisupervised classification other than self-training and cotraining.
  - Research and describe an approach to active learning other than pool-based learning.
  - Research and describe an alternative approach to instance-based transfer learning.
- 7.6.** Given  $n$  training examples  $(\mathbf{x}_i, y_i)$  ( $i = 1, 2, \dots, n$ ) where  $\mathbf{x}_i$  is the feature vector of  $i$ th training example and  $y_i$  is its label, we training an support vector machine (SVM) with Radial Basis Function (RBF) kernel on the training data. Note that the RBF kernel is defined as  $K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2)$ .
- Let  $\mathbf{G}$  be the  $n \times n$  kernel matrix of RBF kernel, i.e.  $\mathbf{G}[i, j] = K_{\text{RBF}}(\mathbf{x}_i, \mathbf{x}_j)$ . Prove that all eigenvalues of  $\mathbf{G}$  are nonnegative.
  - Prove that RBF kernel is the sum of infinite number of polynomial kernels.
  - Suppose the distribution of training examples is shown in Fig. 7.30, where “+” denotes positive example and “-” denotes the negative sample. If we set  $\gamma$  large enough (say 1000 or larger), what could possibly be the decision boundary of the SVM after training? Please draw it on Fig. 7.30.
  - If we set  $\gamma$  to be infinitely large, what could possibly happen when training this SVM?
- 7.7.** Distance metric learning aims to learn a distance metric that best describe the distance between two data points. One of the most commonly used distance metric is Mahalanobis distance. It is of the form

$$d(\mathbf{x}, \mathbf{y})^2 = (\mathbf{x} - \mathbf{y})^T \mathbf{M}(\mathbf{x} - \mathbf{y}),$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are feature vectors for two different data points. Now, supposing we have  $n$  training examples  $(\mathbf{x}_i, y_i)$ , ( $i = 1, 2, \dots, n$ ), we aim to learn the matrix  $\mathbf{M}$  from the data. Research and describe one supervised method and one unsupervised method to learn the matrix  $\mathbf{M}$ .

- 7.8.** Machine learning and data mining techniques have great potential in automatic decision making. Despite the success in deploying these techniques, many end-users do not understand how the decisions are made. Thus it is important to study the explainability of machine learning models.

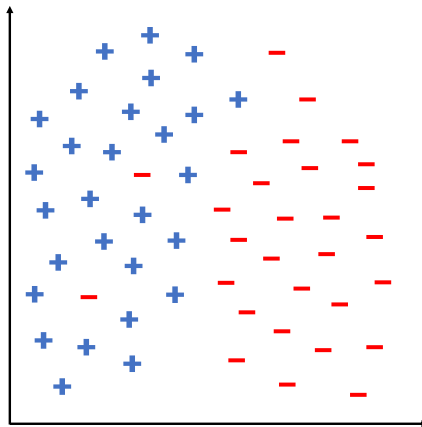


FIGURE 7.30

Distribution of training examples.

Research and describe two different methods to explain the predictions of machine learning models and justify why you think they are interpretable in your own words.

- 7.9. For graph mining using random walk with restart (RWR), the formula is  $\mathbf{r}_i = c\tilde{W}\mathbf{r}_i + (1 - c)\mathbf{e}_i$ , where the ranking vector  $\mathbf{r}_i$  will start the random walk from node  $i$ ,  $c$  is the restart probability,  $\tilde{W}$  is the normalized weight matrix, and  $\mathbf{e}_i$  is the starting vector. Please explain:
- Why RWR could capture multiple weighted relationship between nodes?
  - What is the similarity and the difference between random walk based graph kernel and RWR?
- 7.10. The traditional machine learning approach usually needs human experts to label the data examples (e.g., document, images, signals, etc.) to train a model to perform classification or regression. The human labeling process is normally expensive in terms of both time and money. Especially for the case of deep models, where the size of the training data could be extremely large.
- One alternative approach is called **distant supervision**, where the training data are generated by utilizing the existing database such as Freebase. For example, if our target is to extract the relation of friends, the item in Freebase that includes Buzz Lightyear and Woody Pride would be a positive example. By this mean, we can easily generate a large amount of labeled training data. However, for the model training, having only the positive examples are not enough. A more critical issue is how to generating the negative examples from the large-scale database. Please elaborate at least two ways to generating the negative examples in **distant supervision**.
- 7.11. In zero-shot learning, the model observes test samples from classes that were not observed during training, and needs to predict the category they belong to. Formally, the training data has a label space  $Y$  and the testing data has a different label space  $Y'$ , where  $Y \cap Y' = \emptyset$ . Given training data  $\{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, N\}$ , zero-shot learning aims to learn a function  $f : \mathbb{R}^n \rightarrow Y \cup Y'$ . Suppose for each label  $y \in Y \cup Y'$ , a label representation vector  $\mathbf{y}_i \in \mathbb{R}^m$  is also given.

- a. Suppose we aim to learn a mapping function  $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$  (e.g.,  $g(\mathbf{x}) = \mathbf{W}\mathbf{x}$ ) from the training data so that  $\mathbf{y}_i$  is close to  $g(\mathbf{x}_i)$  ( $i = 1, \dots, N$ ). Use this idea to design a zero-shot learning algorithm.
- b. Suppose we aim to learn a scoring function  $g : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$  (e.g.,  $g(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{W}\mathbf{y}$ ) from the training data so that  $g(\mathbf{x}_i, \mathbf{y}_i)$  is large ( $i = 1, \dots, N$ ). Use this idea to design a zero-shot learning algorithm.
- 7.12.** In streaming data classification, there is an infinite sequence of the form  $(\mathbf{x}, y)$ . The goal is to find a function  $y = f(\mathbf{x})$  that can predict the label  $y$  for an unseen instance  $\mathbf{x}$ . Due to evolving nature of streaming data, the data set size is not known. In this problem, we assume  $\mathbf{x} \in [-1, 1] \times [-1, 1]$  and  $y \in \{-1, 1\}$ , and the first eight training samples are shown below.

| training sample | $x_1$ | $x_2$ | label |
|-----------------|-------|-------|-------|
| 1               | 0.5   | -0.5  | 1     |
| 2               | -0.5  | 0.5   | -1    |
| 3               | 0.5   | 0.25  | 1     |
| 4               | 0.8   | 0.25  | 1     |
| 5               | 0.25  | 0.5   | -1    |
| 6               | -0.5  | -0.25 | -1    |
| 7               | -0.8  | -0.25 | -1    |
| 8               | -0.25 | -0.5  | 1     |

- a. Suppose we have stored the first eight training samples. Now a test data point  $\mathbf{x}_{test} = (0.7, 0.25)^T$  comes. Use the  $k$ -nearest neighbors algorithm ( $k$ -NN) to predict the label of  $\mathbf{x}_{test}$ . You can set  $k=1$ .
- b. Due to the unknown size of streaming data, it could be infeasible to store all the training samples. However, if we divide the feature space into several subareas (e.g.,  $A_1 = [0, 1] \times [0, 1]$ ,  $A_2 = [-1, 0] \times [0, 1]$ ,  $A_3 = [-1, 0] \times [-1, 0]$  and  $A_4 = [0, 1] \times [-1, 0]$ ), it is efficient to store and update the number of positive/negative training samples in a subarea. Based on this intuition, devise an algorithm to classify stream data. What is the prediction of your algorithm on the test point  $\mathbf{x}_{test} = (0.7, 0.25)^T$  given the first eight training samples?
- c. Give an example on which the  $k$ -NN algorithm and your algorithm give different predictions. To let your algorithm mimic the idea of  $k$ -NN, how to improve it given a sufficient number of training data?
- 7.13.** Briefly describe the (a) classification and (b) feature selection steps in the genetic algorithm.
- 7.14.** Suppose we are given  $M$  temporal sequences  $S = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)}\}$ , where each temporal sequence  $\mathbf{x}^{(m)}$ ,  $m = 1, \dots, M$  consists  $n^{(m)}$  temporal segments, that is,  $\mathbf{x}^{(m)} = \{x_1^{(m)}, \dots, x_{n^{(m)}}^{(m)}\}$ . Note that the length of temporal sequences could be different. There exist both normal sequence (labeled as  $Y^{(m)} = 0$ ) and abnormal sequence (labeled as  $Y^{(m)} = 1$ ) in  $S$ .
- a. In the unsupervised setting, we do not have any labels for either the abnormal sequences and normal sequences. We observe that (1) the majority of temporal sequences are normal, whereas only a small portion of temporal sequences in  $S$  correspond to abnormal sequences; and (2) the abnormal sequences often deviate a lot from the normal sequences. Can you propose your own solution to identify the abnormal sequences out of  $S$ ?
- b. In the supervised setting, we are given a training set with labeled abnormal sequences and normal sequences. Could you name one popular supervised sequence classification model

to identify the abnormal sequences? What are the pros and cons of the supervised method, compared to your proposed unsupervised solution in (a)?

- 7.15.** Both reinforcement learning (RL) and the multiarmed bandit (MAB) are well known for modeling the interactions between agents and outside environments in order to achieve the maximum rewards. Interestingly, MAB is often referred to as the one-state RL problem. Could you explain why and compare the differences between these two problems?

---

## 7.10 Bibliographic notes

A detailed introduction of Fisher score can be found in Duda, Stork, and Hart [DHS01]. Gu, Li, and Han proposed the generalized Fisher score for joint feature selection [GLH12]. Many alternative goodness measures for feature selection have been developed, such as Laplacian score by He, Cai, and Niyogi [HCN05], and trace ratio by Nie et al. [NXJ<sup>+</sup>08]. Kohavi and Sommerfield [KS95] introduced wrapper methods for feature selection. Tibshirani [Tib96] developed LASSO for feature selection with the regression task. Ravikumar, Wainwright, and Lafferty [RWL10] applied L1 regularization for feature selection with the logistic regression classifier. Many extensions and generalizations of LASSO exist, such as group LASSO by Meier, Van, and Sara [MVDGB08], fused LASSO by Tibshirani et al. [TSR<sup>+</sup>05], and graphical LASSO by Friedman, Hastie, and Tibshirani [FHT08]. For comprehensive reviews of feature selection, see Chandrashekar and Sahin [CS14], and Li et al. [LCW<sup>+</sup>17].

For an introduction to Bayesian belief networks, see Darwiche [Dar10] and Heckerman [Hec96]. For a thorough presentation of probabilistic networks, see Pearl [Pea88] and Koller and Friedman [KF09]. Solutions for learning the belief network structure from training data given observable variables are proposed in Cooper and Herskovits [CH92]; Buntine [Bun94]; and Heckerman, Geiger, and Chickering [HGC95]. Algorithms for inference on belief networks can be found in Russell and Norvig [RN95] and Jensen [Jen96]. The method of gradient descent, described in Section 7.2.2, for training Bayesian belief networks, is given in Russell, Binder, Koller, and Kanazawa [RBKK95]. The example given in Fig. 7.4 is adapted from Russell et al. [RBKK95].

Alternative strategies for learning belief networks with hidden variables include application of Dempster, Laird, and Rubin's [DLR77] EM (Expectation Maximization) algorithm (Lauritzen [Lau95]) and methods based on the minimum description length principle (Lam [Lam98]). Cooper [Coo90] showed that the general problem of inference in unconstrained belief networks is NP-hard. Limitations of belief networks, such as their large computational complexity (Laskey and Mahoney [LM97]), have prompted the exploration of hierarchical and composable Bayesian models (Pfeffer, Koller, Milch, and Takusagawa [PKMT99] and Xiang, Olesen, and Jensen [XOJ00]). These follow an object-oriented approach to knowledge representation. Fishelson and Geiger [FG02] present a Bayesian network for genetic linkage analysis.

Support vector machines (SVMs) grew out of early work by Vapnik and Chervonenkis on statistical learning theory [VC71]. The first paper on SVMs was presented by Boser, Guyon, and Vapnik [BGV92]. More detailed accounts can be found in books by Vapnik [Vap95, Vap98]. Good starting points include the tutorial on SVMs by Burges [Bur98], as well as textbook coverage by Haykin [Hay08], Kecman [Kec01], and Cristianini and Shawe-Taylor [CST00]. For methods for solving optimization problems, see Fletcher [Fle87] and Nocedal and Wright [NW99]. These references give

additional details alluded to as “fancy math tricks” in our text, such as transformation of the problem to a Lagrangian formulation and subsequent solving using Karush-Kuhn-Tucker (KKT) conditions. Shalev-Shwartz, Singer, Srebro, and Cotter developed a subgradient based solver named Pegasos to scale-up the computation of SVM [SSSSC11]. Many variants of SVMs exist, such as  $L_1$  SVM by Bradley and Mangasarian [BM98b],  $L_{2,1}$  SVM by Cai, Nie, Huang, and Ding [CNHD11], and  $L_p$  SVM by Nie, Wang, and Huang [NWH17]. For a comprehensive introduction of kernel methods, see Hofmann, Schölkopf, and Smola [HSS08].

For the application of SVMs to regression, see Schölkopf, Bartlett, Smola, and Williamson [SBSW98] and Drucker et al. [DBK<sup>+</sup>97]. Approaches to SVM for large data include the sequential minimal optimization algorithm by Platt [Pla98], decomposition approaches such as in Osuna, Freund, and Girosi [OFG97], and CB-SVM, a microclustering-based SVM algorithm for large data sets, by Yu, Yang, and Han [YYH03]. A library of software for support vector machines is provided by Chang and Lin at <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, which supports multiclass classification.

There are several examples of rule-based classifiers. These include AQ15 (Hong, Mozetic, and Michalski [HMM86]), CN2 (Clark and Niblett [CN89]), ITRULE (Smyth and Goodman [SG92]), RISE (Domingos [Dom94]), IREP (Furnkranz and Widmer [FW94]), RIPPER (Cohen [Coh95]), FOIL (Quinlan and Cameron-Jones [Qui90,QCJ93]), and Swap-1 (Weiss and Indurkha [WI98]). For the extraction of rules from decision trees, see Quinlan [Qui87,Qui93]. Rule refinement strategies that identify the most interesting rules among a given rule set can be found in Major and Mangano [MM95].

Many algorithms have been proposed that adapt frequent pattern mining to the task of classification. Early studies on associative classification include the CBA algorithm, proposed in Liu, Hsu, and Ma [LHM98]. A classifier that uses *emerging patterns* (itemsets with support that varies significantly from one data set to another) is proposed in Dong and Li [DL99] and Li, Dong, and Ramamohanarao [LDR00]. CMAR is presented in Li, Han, and Pei [LHP01]. CPAR is presented in Yin and Han [YH03]. Cong, Tan, Tung, and Xu describe RCBT, a method for mining top- $k$  covering rule groups for classifying high-dimensional gene expression data with high accuracy [CTTX05].

Wang and Karypis [WK05] present HARMONY (Highest confidence classificAtion Rule Mining foR iNstance-centric classifYing), which directly mines the final classification rule set with the aid of pruning strategies. Lent, Swami, and Widom [LSW97] propose the ARCS system regarding mining multidimensional association rules. It combines ideas from association rule mining, clustering, and image processing, and applies them to classification. Meretakos and Wüthrich [MW99] propose constructing a naïve Bayesian classifier by mining long itemsets. Veloso, Meira, and Zaki [VMZ06] propose an association rule-based classification method based on a lazy (noneager) learning approach, in which the computation is performed on a demand-driven basis.

Studies on discriminative frequent pattern-based classification were conducted by Cheng, Yan, Han, and Hsu [CYHH07] and Cheng, Yan, Han, and Yu [CYHY08]. The former work establishes a theoretical upper bound on the discriminative power of frequent patterns (based on either information gain [Qui86] or Fisher score [DHS01]), which can be used as a strategy for setting minimum support. The latter work describes the DDPMine algorithm, which is a direct approach to mining discriminative frequent patterns for classification in that it avoids generating the complete frequent pattern set. Kim et al. proposed an NDPMine algorithm that performs frequent and discriminative pattern-based classification by taking *repetitive* features into consideration [KKW<sup>+</sup>10]. Shang, Tong, Peng, and Han propose a concise and effective discriminative patterns-based classification framework named DPClass [STPH16].

Zhu [Zhu05] presents a comprehensive survey on semisupervised classification. For additional references, see the book edited by Chapelle, Schölkopf, and Zien [CIZ06]. Semisupervised SVM named S3SVM was developed by Bennett and Demiriz [BD<sup>+</sup>99]. For a survey on active learning, see Settles [Set10]. Tong and Chang develop an active SVM method based on the concept of version space [TC01]. He and Carbonell develop a nearest-neighbor-based active learning method to discover novel classes. Pan and Yang present a survey on transfer learning [PY10]. The TrAdaBoost boosting algorithm for transfer learning is given in Dai, Yang, Xue, and Yu [DYXY07]. For the negative transfer effect in transfer learning, see Rosenstein, Marx, Kaelbling, and Dietterich [RMKD05]. Ge et al. [GGN<sup>+</sup>14] developed an effective method to mitigate the negative transfer in multisource transfer learning based on Rademacher distribution.

Go, Bhayani, and Huang developed a sentiment classification method using distant supervision. Zubiaga and Ji proposed to use web page directories as distant supervision for Twitter classification. Alternatively, Magdy, Sajjad, El-Ganainy, and Sebastiani treated Youtube labels as distant supervision for the purpose of Twitter classification. Ratner et al. [RBE<sup>+</sup>17] developed Snorkel, a system that allows the user to rely on a labeling function instead of manual labeling to train a machine learning algorithm. Palatucci, Pomerleau, Hinton, and Mitchell [PPHM09] developed a zero-shot learning method based on semantic output codes. Romera-Paredes and Torr [RPT15] proposed a simple yet effective zero-shot learning method based on a two-layer linear network. Wang, Zheng, Yu, and Miao [WZYM19] present a comprehensive survey on zero-shot learning. For evaluations of zero-shot learning, see Xian, Schiele, and Akata [XSA17].

Wang, Fan, Yu, and Han [WFYH03] developed an ensemble method for classifying stream data with drifting concepts. Hulten, Spencer, and Domingos developed an alternative method based on decision trees for stream data classification. Aggarwal, Han, Wang, and Yu proposed an on-demand classification method that dynamically selects the time window in order to train a stream data classifier. Masud et al. [MGK<sup>+</sup>08] developed a semisupervised method for classifying data streams. The book by Aggarwal [Agg07] presents a broad coverage of various stream data mining tasks. Keogh and Pazzani [KP00] developed a scalable method to compute time warping distance. The application of string kernel in text classification can be found in Lodhi et al. [LSST<sup>+</sup>02]. For surveys of sequence data classification, see Xing, Pei, and Keogh [XPK10], and Dietterich [Die02]. Tong, Faloutsos, and Pan [TFP06] developed a fast random walk with restart method to compute node proximity on graphs. Henderson et al. [HGL<sup>+</sup>11] proposed to generate graph structure features in a recursive way. Many methods for graph similarity or kernel exist, such as random walk graph kernel by Vishwanathan, Schraudolph, Kondor, and Borgwardt [VSKB10], and Weisfeiler-Lehman graph kernels by Shervashidze et al. [SSVL<sup>+</sup>11]. For comparison of different graph similarity measures, see Koutra et al. [KSV<sup>+</sup>16].

Work on multiclass classification is described in Hastie and Tibshirani [HT98], Tax and Duin [TD02], and Allwein, Shapire, and Singer [ASS00]. Dietterich and Bakiri [DB95] propose the use of error-correcting codes for multiclass classification. Weinberger and Saul [WS09] introduced distance metric learning for nearest neighbor classification. Xing, Jordan, Russell, and Ng [XJRN02] studied distance metric learning in the context of clustering. Movshovitz-Attias et al. [MATL<sup>+</sup>17] proposed to use proxy-based loss to speed up the convergence of distance metric learning. For surveys of distance metric learning, see Yang and Jin [YJ06], and Wang and Sun [WS15]. LIME was proposed by Ribeiro, Singh, and Guestrin [RSG16] to explain the classifiers. Many alternative interpretation techniques have been developed, such as Shapley values by [LL17], influence function by Koh and Liang [KL17], and counterfactual explanation by Van Looveren and Klaise [VLK19]. For a comprehensive introduction

of interpretable machine learning, see Molnar [Mol20]. For texts on genetic algorithms, see Goldberg [Gol89], Michalewicz [Mic92], and Mitchell [Mit96]. For the application of genetic algorithm in feature selection, see Ghamisi and Benediktsson [GB14]. Suguna and Thanushkodi [ST10] applied genetic algorithm to improve  $k$ -nearest neighbor classifier. Feng et al. [FHZ<sup>+</sup>18] applied reinforcement learning for relation classification. For a comprehensive introduction of reinforcement learning, see Sutton and Barto [SB18].



This page intentionally left blank

# Cluster analysis: basic concepts and methods

**Imagine that you are** the director of Customer Relationships at a retail company. Managing millions of customers one by one is inefficient and ineffective. You would like to organize all customers of the company into a small number of groups so that each group can be assigned to a different manager. Strategically, you would like that the customers in each group are as similar as possible. Two customers having very different business patterns should not be placed in the same group. Your intention behind this business strategy is to develop customer relationship campaigns that specifically target each group, based on common features shared by the customers in the group. What kind of data mining techniques can help you accomplish this task?

Unlike in classification, the class label (i.e., the group-id in this context) of each customer is unknown in this new task. You need to *discover* these groupings. Given a large number of customers and many attributes describing customer profiles, it can be costly or even infeasible to manually study the data and come up with a way to partition the customers into strategic groups. You need a *clustering* tool to help.

*Clustering* is the process of grouping a set of data objects into multiple groups or *clusters* so that objects within a cluster have high similarity, but are dissimilar to objects in other clusters. Dissimilarities and similarities are assessed based on the attribute values describing the objects and often involve distance measures.<sup>1</sup> Clustering as a data mining tool has its roots in many application areas, such as biology, security, business intelligence, and Web search.

This chapter presents the basic concepts and methods of cluster analysis. In Section 8.1, we introduce the basic concept of clustering and study the requirements of clustering methods for massive amounts of data and various applications. You will learn several basic clustering techniques, organized into several categories, namely *partitioning methods* (Section 8.2), *hierarchical methods* (Section 8.3), and *density-based and grid-based methods* (Section 8.4). In Section 8.5, we discuss how to evaluate clustering methods. A discussion of advanced methods of clustering is reserved for Chapter 9.

## 8.1 Cluster analysis

This section sets up the groundwork for studying cluster analysis. Section 8.1.1 defines cluster analysis and presents examples where clustering is useful. In Section 8.1.2, you will learn aspects for comparing

---

<sup>1</sup> Data similarity and dissimilarity are discussed in detail in Chapter 2. You may want to refer to the corresponding section for a quick review.

clustering methods, as well as requirements for clustering. An overview of basic clustering techniques is presented in Section 8.1.3.

### 8.1.1 What is cluster analysis?

**Cluster analysis** or simply **clustering** is the process of partitioning a set of data objects (or observations) into subsets. Each subset is a **cluster**, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters. The set of clusters resulting from a cluster analysis can be referred to as a **clustering**. In this context, different clustering methods may generate different clusterings on the same data set. The same clustering method equipped with different parameters or even different initializations may also produce different clusterings. Such partitioning is not performed by humans, but by a clustering algorithm. Hence, clustering is useful in that it can lead to the discovery of previously unknown groups within the data.

Cluster analysis has been widely used in many applications such as business intelligence, image pattern recognition, Web search, biology, and security. For example, in business intelligence, clustering can be used to organize a large number of customers into groups, where customers within a group share strong similar characteristics. This facilitates the development of business strategies for enhanced customer relationship management. Moreover, consider a consultant company with a large number of projects. To improve project management, such as project delivery and outcome quality control, clustering can be applied to partition projects into categories based on similarities in, for example, business scenarios, customers, expertise required, period and size, so that project auditing and diagnosis can be conducted effectively.

In image recognition, as another example, clustering can be used to discover clusters or “subclasses” in photos. One application is to automatically group photos according to faces recognized in the images so that the photos of the same person may likely come together into a group. Here, we do not have to specify and label the persons in the photos beforehand, and thus a classification method cannot be applied. A clustering method can use faces as features and partition photos into groups so that the faces in the same group are similar and the faces in different groups are dissimilar. Moreover, typically there are many different ways to organize photos. Clustering can help automatically identify significant features and suggest meaningful ways to organize photos into groups accordingly. For example, a group of scenic pictures may be formed using the features of blue sky and beach, whereas another group may share the theme of snow, and a third group highlights group photos with many faces.

Clustering has also found many applications in Web search. For example, a keyword search may often return a large number of hits (i.e., pages relevant to the search) due to the extremely large number of web pages. Clustering can be used to organize the search results into groups and present the results in a concise and easily accessible way. Moreover, clustering techniques have been developed to cluster documents into topics, which are commonly used in information retrieval practice.

As a data mining function, cluster analysis can be used as a standalone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on the detected clusters and the selected attributes or features.

Because a cluster is a collection of data objects that are similar to one another within the cluster and dissimilar to objects in other clusters, a cluster of data objects can be treated as an implicit class.

In this sense, clustering is sometimes called **automatic classification** or **unsupervised classification**. Again, a critical difference here is that clustering can automatically find the groupings. This is a distinct advantage of cluster analysis.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their *similarity*. Clustering can also be used for **outlier detection**, where outliers (values that are “far away” from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card frauds and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and infrequent purchases at unusual locations, may be of interest as possible fraudulent activities. Outlier detection is the subject of Chapter 11.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning and deep learning, spatial database technology, information retrieval, Web search, biology, marketing, and many other application areas. Owing to the huge amounts of data collected in databases, cluster analysis has become a highly active topic in data mining research.

As a branch of statistics, cluster analysis has been extensively studied, with the main focus on *distance-based cluster analysis*. Cluster analysis tools based on  $k$ -means,  $k$ -medoids, and several other methods also have been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS. In machine learning, recall that classification is known as supervised learning because the class label information is given, that is, the learning algorithm is supervised in that it is told the class membership of each training tuple. Clustering is known as **unsupervised learning** because the class label information is not present. For this reason, clustering is a form of **learning by observation**, rather than *learning by examples*. In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in *large data sets*. Active themes of research focus on the *scalability* of clustering methods, the effectiveness of methods for clustering *complex shapes* (e.g., nonconvex) and *types of data* (e.g., text, graphs, and images), *high-dimensional* clustering techniques (e.g., clustering objects with thousands or even millions of features), and methods for clustering *mixed numerical and nominal data* in large data sets.

### 8.1.2 Requirements for cluster analysis

Clustering is a challenging research field. In this section, you will learn about the requirements for clustering as a data mining tool, as well as aspects that can be used for comparing clustering methods.

When we think about employing a clustering method, what requirements should we consider for the method? The following are typical requirements of clustering in data mining.

- **Ability to deal with various kinds of data objects:** Many algorithms are designed to cluster numeric (interval-based) data objects. However, applications may require clustering objects based on a mixed data types, such as binary, nominal (categorical), ordinal, and numerical data, as well as data objects of various kinds, such as text, graphs, sequences, images, and videos.
- **Scalability:** Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions or even billions of objects, such as in Web search scenarios. Clustering on only a sample of a given large data set may lead to biased results. Therefore highly scalable clustering algorithms are needed.
- **Discovery of clusters with arbitrary shape:** Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures (Chapter 2). Algorithms based on such distance mea-

tures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. For example, we may want to use clustering to find the frontier of a running forest fire in a satellite image, which is often not spherical. It is important to develop algorithms that can detect clusters of arbitrary shape.

- **Requirements for domain knowledge to determine input parameters:** Many clustering algorithms require users to provide domain knowledge in the form of input parameters such as the desired number of clusters. Consequently, the clustering results may be sensitive to such parameters. Parameters are often hard to determine, especially for data sets of high dimensionality where users have yet to grasp a deep understanding of their data. Requiring the specification of domain knowledge not only burdens users, but also makes the quality of clustering difficult to control. Clustering algorithms that do not heavily rely on domain knowledge input or can help users explore domain knowledge are highly preferable.
- **Ability to deal with noisy data:** Most real-world data sets contain outliers and/or missing, unknown, or erroneous data. For example, data collected by physical sensors is often noisy. Clustering algorithms can be sensitive to such noise and may produce poor-quality clusters. Therefore, we need clustering methods that are robust to noise.
- **Incremental clustering and insensitivity to input order:** In many applications, incremental updates (representing newer data) may arrive at any time. Some clustering algorithms cannot incorporate incremental updates into existing clustering structures and, instead, have to recompute a new clustering from scratch. Clustering algorithms may also be sensitive to the input data order. That is, given a set of data objects, clustering algorithms may return dramatically different clusterings depending on the order in which the objects are presented. Incremental clustering algorithms and algorithms that are insensitive to the input order are needed.
- **Capability of clustering high-dimensionality data:** A data set can contain numerous dimensions or attributes. When clustering documents, for example, each keyword can be regarded as a dimension, and there are often thousands of keywords. Most clustering algorithms are good at handling low-dimensional data such as data sets involving only two or three dimensions. Finding clusters of data objects in a high-dimensional space is challenging, especially considering that such data can be very sparse and highly skewed.
- **Constraint-based clustering:** Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your task is to choose the locations for a given number of new electric vehicle charging stations in a city. To decide upon this, you may cluster potential charging needs while considering constraints such as available spaces, electricity networks, and the river and highway networks in a city. A challenging task is to find data groups with good clustering behaviors that satisfy specified constraints.
- **Interpretability and usability:** Users want clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and clustering methods.

Given one data set, using different clustering methods or using different parameters or initializations, we may be able to obtain different clusterings. How can we evaluate and compare the clusterings? The following are the orthogonal aspects with which clustering methods can be compared.

- **Single vs. multilevel clustering:** In many clustering methods, all the objects are partitioned so that no hierarchy exists among the clusters. That is, all the clusters are at the same level conceptually. Such a method is useful, for example, for partitioning customers into groups so that each group has its own manager. Alternatively, other methods partition data objects hierarchically, where clusters can be formed at different semantic levels. For example, in text mining, we may want to organize a corpus of documents into multiple general topics, such as “politics” and “sports,” each of which may have subtopics. For instance, “football,” “basketball,” “baseball,” and “hockey” can exist as subtopics of “sports.” The latter four subtopics are at a lower level in the hierarchy than “sports.”
- **Separation of clusters:** Some methods partition data objects into mutually exclusive clusters. In some other situations, the clusters may not be exclusive, that is, a data object may belong to more than one cluster. For example, when clustering documents into topics, a document may be related to multiple topics. Thus the topics as clusters may not be exclusive.
- **Similarity measure:** Some methods determine the similarity between two objects by the distance between them. Such a distance can be defined on a Euclidean space, a road network, a vector space, or some other space. For some applications, similarity may also be defined by other means such as connectivity based on density or contiguity and thus may or may not rely on the absolute distance between two objects. Similarity measures play a fundamental role in the design of clustering methods. While distance-based methods can often take advantage of some computation and optimization techniques, density- and continuity-based methods can often find clusters of arbitrary shape.
- **Clustering in full space vs. subspace:** Many clustering methods search for clusters within the entire given data space. These methods are useful for low-dimensionality data sets. With high-dimensional data, however, there can be many irrelevant attributes, which can make similarity measurements unreliable. Consequently, clusters found in the full space are often meaningless. It is often better to instead search for clusters within different subspaces of the same data set. *Subspace clustering* discovers both clusters and subspaces (often of low dimensionality) containing interesting clusters.

To conclude, clustering algorithms have a series of requirements. These factors include the ability to deal with different kinds of data objects, scalability, robustness to noisy data, incremental updates, clusters of arbitrary shape, and constraints. Interpretability and usability are also important. In addition, clustering methods can differ with respect to single vs. multilevel clustering, whether or not clusters are mutually exclusive, the similarity measures used, and whether or not subspace clustering is performed.

### 8.1.3 Overview of basic clustering methods

There are many clustering algorithms in the literature. It is difficult to provide a crisp categorization of clustering methods because these categories may overlap so that a method may have features from several categories. Nevertheless, it is useful to present a relatively organized picture of clustering methods. In general, the major fundamental clustering methods can be classified into the following categories, which are discussed in the rest of this chapter.

**Partitioning methods:** Given a set of  $n$  objects, a partitioning method constructs  $k$  ( $k \leq n$ ) partitions of the data, where each partition represents a cluster. That is, it divides the data into  $k$  groups such that each group must contain at least one object. Typically,  $k$  is set to a small number, that is,  $k \ll n$ . In other words, a partitioning method conducts one-level partitioning on data sets. The basic partitioning methods typically adopt *exclusive cluster separation*. That is, each object must

belong to exactly one group. This requirement may be relaxed. For example, in fuzzy partitioning techniques an object may take probabilities to belong to more than one cluster. References to such techniques are given in the bibliographic notes (Section 8.8).

Most partitioning methods are distance-based. Given  $k$ , the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an **iterative relocation technique** that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are “close” or related to each other, whereas objects in different clusters are “far apart” or very different. There are various kinds of other criteria for judging the quality of partitions. Traditional partitioning methods can be extended for subspace clustering rather than searching the full data space. This is useful when there are many attributes and the data is sparse.

Achieving global optimality in partitioning-based clustering is often computationally prohibitive, potentially requiring an exhaustive enumeration of all the possible partitions. Instead, most applications adopt popular heuristic methods, such as greedy approaches like the *k-means* and the *k-medoids* algorithms, which progressively improve the clustering quality and approach a local optimum. These heuristic clustering methods work well for finding spherical-shaped clusters in small- to medium-sized data sets. To find clusters with complex shapes and for very large data sets, partitioning-based methods need to be extended. Partitioning-based clustering methods are studied in depth in Section 8.2.

**Hierarchical methods:** A hierarchical method creates a hierarchical decomposition of a given set of data objects. A hierarchical method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the *bottom-up* approach, starts with each object forming a separate group. It successively merges the objects or groups close to one another, until all the groups are merged into one (the topmost level of the hierarchy), or a termination condition holds. The *divisive approach*, also called the *top-down* approach, starts with all the objects in the same cluster. In each successive iteration, a cluster is split into smaller clusters, until eventually each object is in one cluster, or a termination condition holds.

Hierarchical clustering methods can be distance-, density-, or continuity-based. Various extensions of hierarchical methods consider clustering in subspaces as well.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be reverted. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices. Such techniques cannot correct erroneous decisions; however, methods for improving the quality of hierarchical clustering have been proposed. Hierarchical clustering methods are studied in Section 8.3.

**Density-based and grid-based methods:** Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty in discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of *density*. Their general idea is to continue growing a given cluster as long as the density (number of objects or data points) in the “neighborhood” exceeds some threshold. For example, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise or outliers and discover clusters of arbitrary shape.

Density-based methods can divide a set of objects into multiple exclusive clusters, or a hierarchy of clusters. Typically, density-based methods consider exclusive clusters only, and do not consider fuzzy clusters. Moreover, density-based methods can be extended from full space to subspace clustering.

One way to implement the idea of density-based clustering is grid-based methods, which quantize the object space into a finite number of cells that form a grid structure. All the clustering operations are performed on the grid structure (i.e., on the quantized space). For example, the dense cells, that is, those cells each containing a sufficient number of data points, are considered components of clusters, and are used to assemble clusters. The main advantage of the grid-based methods is the fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space. Using grids is often an efficient approach to many spatial data mining problems, including clustering. In addition to density-based clustering, grid-based methods can be integrated with other clustering methods, such as hierarchical methods. Density-based and grid-based clustering methods are studied in Section 8.4.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category. Furthermore, some applications may have clustering criteria that require an integration of several clustering techniques.

In the following sections, we examine representative clustering methods in detail. Advanced clustering methods and related issues are discussed in Chapter 9.

---

## 8.2 Partitioning methods

The simplest and most fundamental version of cluster analysis is partitioning. In partitioning clustering, we organize the objects in a given set into several exclusive groups or clusters. Each cluster can be typified by a representative. In other words, each object  $o$  can be assigned to the cluster whose representative that  $o$  is the closest or most similar to. To keep the problem specification concise, we can assume that the number of expected clusters is given. This parameter is the starting point for partitioning methods. There are two foremost technical issues in partitioning methods. First, how can we decide the representatives of clusters? Second, how can we measure the distance or similarity between objects or between objects and representatives.

Formally, given a data set,  $D$ , of  $n$  objects, and  $k$ , the number of clusters to form, a **partitioning algorithm** organizes the objects into  $k$  partitions ( $k \leq n$ ), where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance, so that the objects within a cluster are “similar” to one another and “dissimilar” to the objects in other clusters in terms of the data set attributes.

In this section you will learn the partitioning methods. We will start with the most prominent partitioning method,  $k$ -means (Section 8.2.1). Then, in Section 8.2.2 we will look at a series of variations of partitioning methods to address different types of data and different application scenarios. Last, we will discuss kernel  $k$ -means, an advanced version of partitioning method that can explore nonlinear separability between clusters in high dimensional data.



## 8.2.1 *k*-Means: a centroid-based technique

Suppose a data set,  $D$ , contains  $n$  objects in Euclidean space. Partitioning methods distribute the objects in  $D$  into  $k$  clusters,  $C_1, \dots, C_k$ , that is,  $C_i \subset D$ ,  $|C_i| \geq 1$ , and  $C_i \cap C_j = \emptyset$  for  $(1 \leq i, j \leq k, i \neq j)$ . Each cluster is required to have at least one object. An objective function is used to assess the partitioning quality so that objects within a cluster are similar to one another but dissimilar to objects in other clusters. This is, the objective function aims for high intracluster similarity and low intercluster similarity.

A centroid-based partitioning technique uses the *centroid* of a cluster,  $C_i$ , as the representative of that cluster. Conceptually, the centroid of a cluster is its center point. The centroid can be defined in various ways such as by the mean or medoid of the objects (or points) assigned to the cluster. The difference between an object  $\mathbf{p} \in C_i$  and  $\mathbf{c}_i$ , the representative of the cluster, is measured by  $\text{dist}(\mathbf{p}, \mathbf{c}_i)$ , where  $\text{dist}(\mathbf{x}, \mathbf{y})$  is the Euclidean distance between two points  $\mathbf{x}$  and  $\mathbf{y}$ . The quality of cluster  $C_i$  can be measured by the **within-cluster variation**, which is the sum of *squared error* between all objects in  $C_i$  and the centroid  $\mathbf{c}_i$ , defined as

$$E = \sum_{i=1}^k \sum_{\mathbf{p} \in C_i} \text{dist}(\mathbf{p}, \mathbf{c}_i)^2, \quad (8.1)$$

where  $E$  is the sum of the squared error for all objects in the data set;  $\mathbf{p}$  is the point in space representing a given object; and  $\mathbf{c}_i$  is the centroid of cluster  $C_i$  (both  $\mathbf{p}$  and  $\mathbf{c}_i$  are multidimensional). In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This objective function tries to make the resulting  $k$  clusters as compact and separate as possible. The task of partitioning clustering can be modeled as to minimize the within-cluster variation (Eq. (8.1)) among all possible assignments of objects into clusters.

Minimizing the within-cluster variation is computationally challenging. In the worst case, we would have to enumerate the number of all possible partitionings. It is easy to see that the number of all possible partitionings is exponential to the number of objects. (This is left to be an exercise.) It has been shown that the problem is NP-hard in the general Euclidean space even for two clusters (i.e.,  $k = 2$ ). To overcome the prohibitive computational cost for the exact solution, greedy approaches are often used in practice. A prime example is the *k*-means algorithm, which is simple and commonly used.

“How does the *k*-means algorithm work?” The *k*-means algorithm defines the centroid of a cluster as the mean value of the points within the cluster. It proceeds as follows. First, it randomly selects  $k$  objects from  $D$ , each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the Euclidean distance between the object and the chosen means. The *k*-means algorithm then iteratively improves the within-cluster variation. For each cluster, it computes the new mean using the objects assigned to the cluster in the previous iteration. All the objects are then reassigned using the updated means as the new cluster centers. The iterations continue until the assignment is stable, that is, the clusters formed in the current round are the same as those formed in the previous round. The *k*-means procedure is summarized in Fig. 8.1.

**Example 8.1. Clustering by *k*-means partitioning.** Consider a set of objects located in 2-D space, as depicted in Fig. 8.2(a). Let  $k = 3$ , that is, the user would like to partition the objects into three clusters.

According to the algorithm in Fig. 8.1, we arbitrarily choose three objects as the three initial cluster centers, where cluster centers are marked by a +. Each object is assigned to a cluster based on the cluster

**Algorithm:  $k$ -means.** The  $k$ -means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

**Input:**

- $k$ : the number of clusters,
- $D$ : a data set containing  $n$  objects.

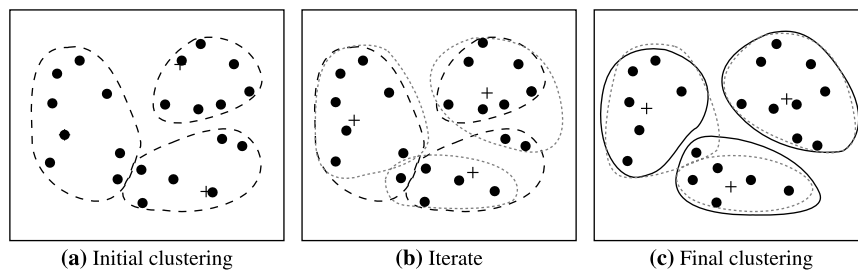
**Output:** A set of  $k$  clusters.

**Method:**

- (1) arbitrarily choose  $k$  objects from  $D$  as the initial cluster centers;
- (2) **repeat**
- (3) (re)assign each object to the cluster to which the object is the most similar;
- (4) update the cluster centers, that is, calculate the mean value of the objects for each cluster;
- (5) **until** no change;

**FIGURE 8.1**

The  $k$ -means partitioning algorithm.



**FIGURE 8.2**

Clustering of a set of objects using the  $k$ -means method; for (b) update cluster centers and reassign objects accordingly (the mean of each cluster is marked by a +).

center to which it is the nearest. Such an assignment forms silhouettes encircled by dotted curves, as shown in Fig. 8.2(a).

Next, the cluster centers are updated. That is, the mean value of each cluster is recalculated based on the current objects in the cluster. Using the new cluster centers, the objects are reassigned to the clusters based on which cluster center is the nearest. Such a reassignment forms new silhouettes encircled by dashed curves, as shown in Fig. 8.2(b).

This process iterates, leading to Fig. 8.2(c). The process of iteratively reassigning objects to clusters to improve the partitioning is referred to as *iterative relocation*. Eventually, no reassignment of the objects in any cluster occurs and so the process terminates. The resulting clusters are returned by the clustering process.  $\square$

The  $k$ -means method is not guaranteed to converge to the global optimum and often terminates at a local optimum. The results may depend on the initial random selection of cluster centers. (You will be asked to give an example to show this as an exercise.) To obtain good results in practice, it is common

to run the  $k$ -means algorithm multiple times with different initial cluster centers. The clustering with the smallest within-cluster variation should be returned as the final result.

The time complexity of the  $k$ -means algorithm is  $O(nkt)$ , where  $n$  is the total number of objects,  $k$  is the number of clusters, and  $t$  is the number of iterations. Normally,  $k \ll n$  and  $t \ll n$ . Therefore the method is relatively scalable and efficient in processing large data sets.

As the simplest version of partitioning methods, the  $k$ -means method has several advantages. First, the  $k$ -means method is conceptually intuitive and relatively simple to implement. Indeed, the  $k$ -means method is included in many software toolkits and open source suites in statistics, data mining, and machine learning. Second, as analyzed, the  $k$ -means method is scalable to large data sets. The runtime is linear with respect to the data set size (i.e., the number of data objects), the number of clusters, and the number of iterations. Third, the  $k$ -means method is guaranteed to converge to some local optimum, and thus likely does not produce very poor results. Fourth, if a user has some domain knowledge about the possible locations of the clusters, the user can set the initial means and then run the iteration steps. In other words, the  $k$ -means method can take a warm-start. Last, the  $k$ -means method can take new observed data easily. That is, if some new data objects arrive after a certain number of iterations, the  $k$ -means method still can easily take those data into the next iteration, and the updated clustering can adapt to the new data.

The  $k$ -means method also has some limitations. First, a user has to manually specify the number of clusters. When a user is not familiar with a data set, it is not easy to set this parameter properly. Second, the effect of result clustering heavily depends on the choice of initial means. When the number of clusters is small, to overcome this limitation, one may run the  $k$ -means method multiple times with different initial means. However, when the number of clusters is large, even running the  $k$ -means method multiple times may not help to mitigate the issue, since it is unlikely all clusters produced in a run are all good. Third, as to be illustrated later (see Fig. 8.16 as an example), the  $k$ -means method may meet difficulty in finding clusters of substantially different sizes and density. Moreover, outliers may distract the centers of clusters (see Example 8.2 for demonstration). Last, since the Euclidean distance is used in the  $k$ -means method, when the dimensionality increases, the distance measure is mainly dominated by noise. In expectation, the distance between any two data objects in a high dimensional space is the same. Thus the  $k$ -means method cannot scale up to high dimensional data straightforwardly.

In the rest of the section, we will discuss some variations of the  $k$ -means method to address some of the above limitations. Some other limitations are unfortunately shared by the partitioning methods, and thus have to be tackled by introducing other types of clustering methods.

## 8.2.2 Variations of $k$ -means

In order to tackle various limitations, there are multiple variants of the  $k$ -means method. In this subsection, we will study some of them that are popularly used in various applications.

### ***k*-Medoids: a representative object-based technique**

The  $k$ -means algorithm is sensitive to outliers because such objects are far away from the majority of the data, and thus, when assigned to a cluster, they can dramatically distort the mean value of the cluster. This inadvertently affects the assignment of other objects to clusters. This effect is particularly exacerbated due to the use of the *squared-error* function (Eq. (8.1)), as observed in Example 8.2.

**Example 8.2. A drawback of  $k$ -means.** Consider six points in a 1-D space having values 1, 2, 3, 8, 9, 10, and 25, respectively. Intuitively, by visual inspection we may imagine the points partitioned into the clusters {1, 2, 3} and {8, 9, 10}, where point 25 is excluded because it appears to be an outlier. How would  $k$ -means partition the values? If we apply  $k$ -means using  $k = 2$  and Eq. (8.1), the partitioning {{1, 2, 3}, {8, 9, 10, 25}} has the within-cluster variation

$$(1 - 2)^2 + (2 - 2)^2 + (3 - 2)^2 + (8 - 13)^2 + (9 - 13)^2 + (10 - 13)^2 + (25 - 13)^2 = 196,$$

given that the mean of cluster {1, 2, 3} is 2 and the mean of {8, 9, 10, 25} is 13. Compare this to the partitioning {{1, 2, 3, 8}, {9, 10, 25}}, for which  $k$ -means computes the within-cluster variation as

$$(1 - 3.5)^2 + (2 - 3.5)^2 + (3 - 3.5)^2 + (8 - 3.5)^2 + (9 - 14.67)^2 \\ + (10 - 14.67)^2 + (25 - 14.67)^2 = 189.67,$$

given that 3.5 is the mean of cluster {1, 2, 3, 8} and 14.67 is the mean of cluster {9, 10, 25}. The latter partitioning has the lower within-cluster variation; therefore the  $k$ -means method assigns the value 8 to a cluster different from that containing values 9 and 10 due to the outlier point 25. Moreover, the center of the second cluster, 14.67, is substantially far from all the members in the cluster.  $\square$

“How can we modify the  $k$ -means algorithm to diminish such sensitivity to outliers?” Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is assigned to the cluster of which the representative object is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object  $p$  and its corresponding representative object. That is, an **absolute-error criterion** is used, defined as

$$E = \sum_{i=1}^k \sum_{p \in C_i} \text{dist}(p, o_i), \quad (8.2)$$

where  $E$  is the sum of the absolute error for all objects  $p$  in the data set, and  $o_i$  is the representative object of  $C_i$ . This is the basis for the  **$k$ -medoids method**, which groups  $n$  objects into  $k$  clusters by minimizing the absolute error (Eq. (8.2)).

When  $k = 1$ , we can find the exact median in  $O(n^2)$  time. However, when  $k$  is a general positive number, the  $k$ -medoid problem is NP-hard.

The **Partitioning Around Medoids (PAM)** algorithm (see Fig. 8.4 later) is a popular realization of  $k$ -medoids clustering. It tackles the problem in an iterative, greedy way. Like the  $k$ -means algorithm, the initial representative objects (called seeds) are chosen arbitrarily. We consider whether replacing a representative object by a nonrepresentative object would improve the clustering quality. All the possible replacements are tried out. The iterative process of replacing representative objects by other objects continues until the quality of the resulting clustering cannot be improved by any replacement. This quality is measured by a cost function of the average dissimilarity between an object and the representative object of its cluster.

Specifically, let  $o_1, \dots, o_k$  be the current set of representative objects (i.e., medoids). To determine whether a nonrepresentative object, denoted by  $o_{\text{random}}$ , is a good replacement for a current medoid

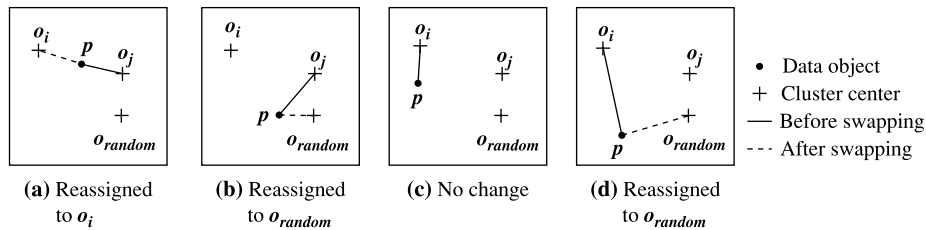


FIGURE 8.3

Four cases of the cost function for  $k$ -medoids clustering.

$o_j$  ( $1 \leq j \leq k$ ), we calculate the distance from every object  $p$  to the closest object in the set  $\{o_1, \dots, o_{j-1}, o_{j+1}, \dots, o_k\}$ , and use the distance to update the cost function. The reassignments of objects to  $\{o_1, \dots, o_{j-1}, o_{j+1}, \dots, o_k\}$  are simple. Suppose object  $p$  is currently assigned to a cluster represented by medoid  $o_j$  (Fig. 8.3a or b). Do we need to reassign  $p$  to a different cluster if  $o_j$  is being replaced by  $o_{random}$ ? Object  $p$  needs to be reassigned to either  $o_{random}$  or some other cluster represented by  $o_i$  ( $i \neq j$ ), whichever is the closest. For example, in Fig. 8.3(a),  $p$  is closest to  $o_i$  and therefore is reassigned to  $o_i$ . In Fig. 8.3(b), however,  $p$  is closest to  $o_{random}$  and so is reassigned to  $o_{random}$ . What if, instead,  $p$  is currently assigned to a cluster represented by some other object  $o_i$ ,  $i \neq j$ ? Object  $p$  remains assigned to the cluster represented by  $o_i$  as long as  $p$  is still closer to  $o_i$  than to  $o_{random}$  (Fig. 8.3c). Otherwise,  $p$  is reassigned to  $o_{random}$  (Fig. 8.3d).

Each time a reassignment occurs, a difference in absolute error,  $E$ , is contributed to the cost function. Therefore the cost function calculates the *difference* in absolute-error value if a current representative object is replaced by a nonrepresentative object. The total cost of swapping is the sum of costs incurred by all nonrepresentative objects. If the total cost is negative, then  $o_j$  is replaced or swapped with  $o_{random}$  because the actual absolute-error  $E$  is reduced. If the total cost is positive, the current representative object,  $o_j$ , is considered acceptable, and nothing is changed in the iteration.

“Which method is more robust— $k$ -means or  $k$ -medoids?” The  $k$ -medoids method is more robust than  $k$ -means in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than a mean. However, the complexity of each iteration in the  $k$ -medoids algorithm is  $O(k(n - k))$ . For large values of  $n$  and  $k$ , such computation becomes very costly and much more costly than the  $k$ -means method. Both methods require the user to specify  $k$ , the number of clusters.

### ***k*-Modes: clustering nominal data**

One limitation of the  $k$ -means method is that it can be applied only when the mean of a set of objects is defined. This may not be the case in some applications such as when data with nominal attributes is involved. The ***k*-modes method** is a variant of  $k$ -means, which extends the  $k$ -means paradigm to cluster nominal data by replacing the means of clusters with modes.

Recall that the mode for a set of data is the value that occurs most frequently in the set. In order to use modes in clustering, we need a new way to compute the distance between two objects. Given two

**Algorithm:  $k$ -medoids.** PAM, a  $k$ -medoids algorithm for partitioning based on medoid or central objects.

**Input:**

- $k$ : the number of clusters,
- $D$ : a data set containing  $n$  objects.

**Output:** A set of  $k$  clusters.

**Method:**

- (1) arbitrarily choose  $k$  objects in  $D$  as the initial representative objects or seeds;
- (2) **repeat**
- (3)     assign each remaining object to the cluster with the nearest representative object;
- (4)     randomly select a nonrepresentative object,  $o_{random}$ ;
- (5)     compute the total cost,  $S$ , of swapping representative object,  $o_j$ , with  $o_{random}$ ;
- (6)     **if**  $S < 0$  **then** swap  $o_j$  with  $o_{random}$  to form the new set of  $k$  representative objects;
- (7) **until** no change;

**FIGURE 8.4**

PAM, a  $k$ -medoids partitioning algorithm.

objects  $\mathbf{x} = (x_1, \dots, x_l)$  and  $\mathbf{y} = (y_1, \dots, y_l)$ , we define the distance

$$dist(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^l d(x_i, y_i), \quad (8.3)$$

where  $d(x, y) = 1$  if  $x \neq y$  and otherwise 0. With this change, the sum of squared error (Eq. (8.1)) remains valid, where  $c_i$  is the representative of cluster  $i$ .

The  $k$ -modes method works largely the same way as the  $k$ -means method. First, it selects  $k$  initial modes, one for each cluster. Second, it allocates an object to the cluster whose mode is the closest to the object using the distance function in Eq. (8.3). Third, it updates the mode of each cluster. For a cluster  $i$  and dimension  $j$ , the mode is updated to the most frequent value on the dimension of all objects assigned to this cluster. If there are more than one such a value, that is, if two or more values of the same frequency happen most frequently in the cluster, we can randomly choose one. The  $k$ -modes method iterates the object allocation and mode update steps until either the sum of squared error (Eq. (8.1)) stabilizes or a given number of iterations are conducted.

The  $k$ -means and the  $k$ -modes methods can be integrated to cluster data with mixed numeric and nominal values. This is known as the  $k$ -prototype method. On each dimension, according to whether it is a numeric attribute or a nominal attribute, we can use either the absolute error  $dist(x - y)$  or the mode difference  $d(x, y) = 1$  if  $x \neq y$  and otherwise 0. Since numeric attributes may have a much larger range in absolute error than the mode difference on nominal attributes, we can associate with each dimension a weight balancing the effect of each dimension. You will have an opportunity to explore the details of the  $k$ -prototype method in the exercise.

### **Initialization in partitioning methods**

It is interesting that by choosing the initial cluster centers carefully, we may be able to not only speed up the convergence of the  $k$ -means algorithm, but also guarantee the quality of the final clustering results. For example, the  $k$ -means++ algorithm chooses the initial centers in the following steps. First, it chooses one center uniformly at random from the objects in the data set. Iteratively, for each object

$p$  other than the chosen centers, it chooses the object as the new center at random with probability proportional to  $D(p)^2$ , where  $D(p)$  is the distance from  $p$  to the closest center that has already been chosen. The iteration continues until  $k$  centers are chosen.

Extensive experimental results have shown that the  $k$ -means++ algorithm can speed up the clustering process by a factor from 2 in most cases. Moreover, the  $k$ -means++ algorithm guarantees an approximation ratio of  $O(\log k)$ ; that is, the within-cluster variation obtained by  $k$ -means++ is not more than  $O(\log k)$  times larger than the global optimum.

### Estimating the number of clusters

The necessity for users to specify  $k$ , the number of clusters, in advance can be seen as a disadvantage. The desired number of  $k$  is often dependent on the shape and scale of the distribution of points in a data set and the desired clustering resolution of the user. There have been studies on how to estimate a desired number of clusters. For example, given a data set of  $n$  objects, let  $B(k)$  and  $W(k)$  be the sum of squares of the distances between and within clusters, respectively, when there are  $k$  clusters. The *Calinski-Harabasz index* is defined by

$$CH(k) = \frac{\frac{B(k)}{k-1}}{\frac{W(k)}{n-k}}. \quad (8.4)$$

The number of clusters  $k$  can be estimated by maximizing the Calinski-Harabasz index.

Gap statistic is another method to estimate the number of clusters. The sum of the pairwise distances for all points in a cluster  $C_i$  is

$$SD_{C_i} = \sum_{p, q \in C_i} dist(p, q).$$

If the data set is divided into  $k$  clusters, define

$$W_k = \sum_{i=1}^k \frac{SD_{C_i}}{2|C_i|},$$

which is the pooled within-cluster sum of squares around the cluster means. The *gap statistic* is

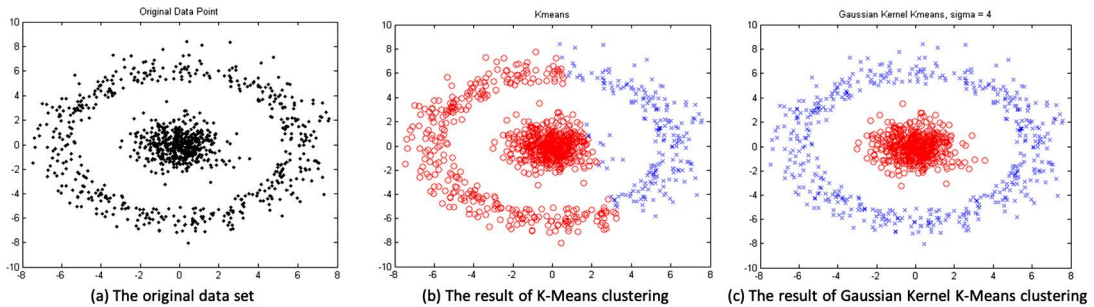
$$Gap_n(k) = E_n^*\{\log(W_k)\} - \log(W_k), \quad (8.5)$$

where  $E_n^*$  is the expectation under a sample of size  $n$  from the reference distribution, that is, the distribution producing the data set to be clustered. We can choose the value  $k$  that maximizes the gap statistic as the estimation of number of clusters.

In Section 8.5.2, we will introduce additional methods to estimate the number of clusters.

### Applying feature transformation

The  $k$ -means method employing the Euclidean distance or any metric measures in general can only output convex clusters. Here, a cluster is convex if for any two points  $a$  and  $b$  belonging to the cluster, every point between the two points on the line connecting  $a$  and  $b$  also belongs to the cluster. Moreover, the  $k$ -means method employing any metric measure can only detect clusters that are linearly separable. That is, two clusters can be separated by a linear hyperplane.



**FIGURE 8.5**

Concave and not linearly separable clusters can be detected by kernel  $k$ -means.

In many applications, clusters may not be convex or linearly separable. In Fig. 8.5(a), one can easily see that there are two clusters, the points at the center form a cluster, which is convex. The other points form another cluster in a “ring” shape. If we apply  $k$ -means on the data set, specify the number of clusters  $k = 2$  and use the Euclidean distance, the output is in Fig. 8.5(b). As you can see,  $k$ -means cuts the data set into two parts using a line. However, those two parts do not match the visual intuition.

Can we still use the  $k$ -means method to find clusters that are concave and not linearly separable? Indeed, kernel  $k$ -means is such a method. The general idea of kernel  $k$ -means is to map the data points in the original input space to a feature space of higher dimensionality where the points belonging to the same cluster are close to each other in the feature space. Explicitly defining a space of high dimensionality and mapping the points into that space is subtle and may be costly. Instead, a convenient way is to apply a kernel function to measure the distance between points.

Recall that Section 7.3.2 introduces the concept of kernel functions. For example, using the Gaussian radial basis function (RBF) kernel, we can calculate the distance between two points  $\mathbf{x}$  and  $\mathbf{y}$  by

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}},$$

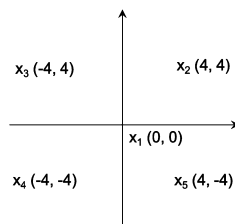
where  $\|\mathbf{x} - \mathbf{y}\|^2$  is indeed the squared Euclidean distance between the two points, and  $\sigma$  is a free parameter. Clearly, the RBF kernel has the range between 0 and 1 and decreases with respect to the Euclidean distance.

How does a kernel function, such as the RBF kernel, transform the similarity among data points? Consider the five points in Fig. 8.6. The Euclidean distance matrix is

$$\begin{bmatrix} 0 & 5.66 & 5.66 & 5.66 & 5.66 \\ 5.66 & 0 & 8 & 11.31 & 8 \\ 5.66 & 8 & 0 & 8 & 11.31 \\ 5.66 & 11.31 & 8 & 0 & 8 \\ 5.66 & 8 & 11.31 & 8 & 0 \end{bmatrix}$$

where the value at the  $i$ th row and the  $j$ th column is the Euclidean distance between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . Let  $\sigma = 4$ . We can apply the RBF kernel to the same five points. The corresponding RBF kernel similarity






---

**FIGURE 8.6**

An example of five points.

matrix is

$$\begin{bmatrix} 0 & e^{-1} & e^{-1} & e^{-1} & e^{-1} \\ e^{-1} & 0 & e^{-2} & e^{-4} & e^{-2} \\ e^{-1} & e^{-2} & 0 & e^{-2} & e^{-4} \\ e^{-1} & e^{-4} & e^{-2} & 0 & e^{-2} \\ e^{-1} & e^{-2} & e^{-4} & e^{-2} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0.37 & 0.37 & 0.37 & 0.37 \\ 0.37 & 1 & 0.135 & 0.02 & 0.135 \\ 0.37 & 0.135 & 1 & 0.135 & 0.02 \\ 0.37 & 0.02 & 0.135 & 1 & 0.135 \\ 0.37 & 0.135 & 0.02 & 0.135 & 1 \end{bmatrix}.$$

The magic here is that the RBF kernel indeed reduces the similarity between two points in a super-linear manner as the Euclidean distance between them increases. This nonlinear allocation of similarity enables  $k$ -means to assemble clusters using points that are not linearly separable and form clusters that are not convex. For example, if we apply the RBF kernel and  $k$ -means on the data set in Fig. 8.5(a), the output is Fig. 8.5(c), where the points in red (gray in print version) form a cluster and the points in blue (dark gray in print version) form another cluster. The output matches the visual intuition nicely.

---

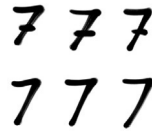
## 8.3 Hierarchical methods

While partitioning methods meet the basic clustering requirement of organizing a set of objects into a number of exclusive groups, in some situations we may want to partition our data into groups at different levels, or in general a hierarchy. A **hierarchical clustering method** works by grouping data objects into a hierarchy or “tree” of clusters.

In this section, you will study hierarchical clustering methods. Section 8.3.1 begins with a discussion about the basic concepts of hierarchical clustering. Then, Section 8.3.2 introduces the agglomerative, bottom-up approaches for hierarchical clustering. Section 8.3.3 presents the divisive, top-down approaches. Hierarchical clustering methods may be combined with other methods. Section 8.3.4 discusses BIRCH, a scalable hierarchical clustering method for large amounts of numeric data. Last, Section 8.3.5 describes the probabilistic hierarchical clustering methods.

### 8.3.1 Basic concepts of hierarchical clustering

Representing data objects in the form of a hierarchy is useful for data summarization and visualization. For example, as a manager of human resources in a company, you may organize your employees



**FIGURE 8.7**

Two clusters of handwritten digit 7s.

into major groups such as executives, managers, and staff. You can further partition these groups into smaller subgroups. For instance, the general group of staff can be further divided into subgroups of senior officers, officers, and trainees. All these groups form a hierarchy. We can easily summarize or characterize the data that is organized into a hierarchy, which can be used to find, say, the average salary of managers and of officers.

Consider handwritten character recognition as another example. A set of handwriting samples may be first partitioned into general groups where each group corresponds to a unique character. Some groups can be further partitioned into subgroups since a character may be written in multiple substantially different ways. For example, Fig. 8.7 shows a group of handwritten digit 7s. The group can be further divided into two subgroups, the first row being a subgroup where a short horizontal line is used in each writing, and the second row being another subgroup. If necessary, the hierarchical partitioning can be continued recursively until a desired granularity is reached.

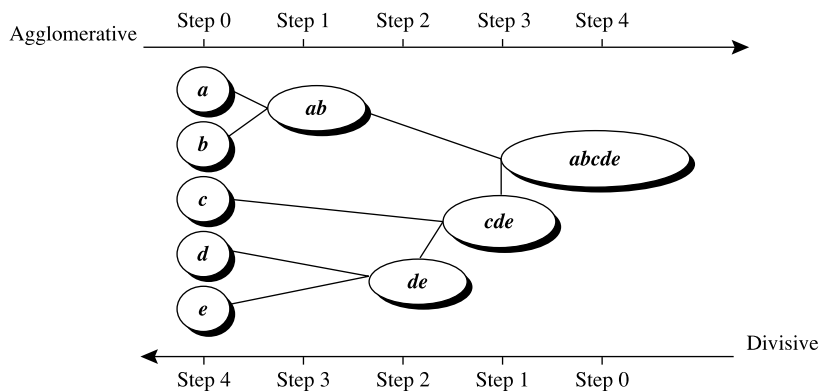
In the previous examples, although we partition the data hierarchically, we do not assume that the data has a hierarchical structure. Our use of a hierarchy here is just to summarize and represent the underlying data in a compressed way. Such a hierarchy is particularly useful for data visualization.

Alternatively, in some applications we may believe that the data bear an underlying hierarchical structure that we want to discover. For example, hierarchical clustering may uncover a hierarchy for the employees in a company structured on, say, salary. In the study of biological evolution, hierarchical clustering may group living creatures according to their biological features to uncover evolutionary paths, which are a hierarchy of species. As another example, grouping configurations of a strategic game (e.g., chess or checkers) in a hierarchical way may help to develop game strategies that can be used to train players.

A hierarchical clustering method can be either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top-down (splitting) fashion. Let us have a closer look at these strategies.

An **agglomerative hierarchical clustering method** uses a bottom-up strategy. It typically starts by letting each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termination conditions are satisfied. The single cluster becomes the hierarchy's root. For the merging step, it finds the two clusters that are closest to each other (according to some similarity measure) and combines the two to form one cluster. Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most  $n$  iterations.

A **divisive hierarchical clustering method** employs a top-down strategy. It starts by placing all objects in one cluster, which is the hierarchy's root. It then divides the root cluster into several smaller subclusters and recursively partitions those clusters into smaller ones. The partitioning process contin-

**FIGURE 8.8**

Agglomerative and divisive hierarchical clustering on data objects  $\{a, b, c, d, e\}$ .

ues until each cluster at the lowest level is coherent enough—either containing only one object, or the objects within a cluster are sufficiently similar to each other.

In either agglomerative or divisive hierarchical clustering, a user can specify the desired number of clusters as a termination condition.

**Example 8.3. Agglomerative vs. divisive hierarchical clustering.** Fig. 8.8 shows the application of an agglomerative hierarchical clustering method and a divisive hierarchical clustering method on a data set of five objects,  $\{a, b, c, d, e\}$ . Initially, the agglomerative method places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters  $C_1$  and  $C_2$  may be merged if an object in  $C_1$  and an object in  $C_2$  form the minimum Euclidean distance between any two objects from different clusters. This is a **single-linkage** approach in that each cluster is represented by all the objects in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. The cluster-merging process repeats until all the objects are eventually merged to form one cluster.

The divisive method proceeds in the contrasting way. All the objects are used to form one initial cluster. The cluster is split according to some principle such as the maximum Euclidean distance between the closest neighboring objects in the cluster. The cluster-splitting process repeats until, eventually, each new cluster contains only a single object.  $\square$

The selection of merge or split points is critical for hierarchical clustering methods, because once a group of objects is merged or split, the process at the next step will operate on the newly generated clusters. It will neither undo what was done previously, nor perform object swapping between clusters. Thus merge or split decisions, if not well chosen, may lead to low-quality clusters. Moreover, the methods do not scale well because each decision of merge or split needs to examine and evaluate many objects or clusters.

A promising direction for improving the clustering quality of hierarchical methods is to integrate hierarchical clustering with other clustering techniques, resulting in **multiple-phase clustering**. We introduce BIRCH as a representative method in Section 8.3.4. BIRCH begins by partitioning objects

hierarchically using tree structures, where the leaf or low-level nonleaf nodes can be viewed as “microclusters” depending on the resolution scale. It then applies other clustering algorithms to perform macroclustering on the microclusters.

There are several orthogonal ways to categorize hierarchical clustering methods. For instance, they may be categorized into *deterministic* methods and *probabilistic* methods. Agglomerative, divisive, and multiphase methods are *deterministic*, since they consider data objects as deterministic and compute clusters according to the deterministic distances between objects. Probabilistic methods use probabilistic models to capture clusters and measure the quality of clusters by the fitness of models. We discuss probabilistic hierarchical clustering in Section 8.3.5.

### 8.3.2 Agglomerative hierarchical clustering

In this section, we discuss some important issues in agglomerative hierarchical clustering methods.

#### *Similarity measures in hierarchical clustering*

*How can we choose which objects and clusters to merge in an agglomerative step?* The core is to measure the similarity between two clusters, where each cluster is generally a set of objects.

Four widely used measures for distance between clusters are as follows, where  $\|\mathbf{p} - \mathbf{p}'\|$  is the distance between two objects or points,  $\mathbf{p}$  and  $\mathbf{p}'$ ;  $\mathbf{m}_i$  is the mean for cluster  $C_i$ ; and  $n_i$  is the number of objects in  $C_i$ . They are also known as *linkage measures*.

$$\text{Minimum distance: } dist_{min}(C_i, C_j) = \min_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} \{\|\mathbf{p} - \mathbf{p}'\|\} \quad (8.6)$$

$$\text{Maximum distance: } dist_{max}(C_i, C_j) = \max_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} \{\|\mathbf{p} - \mathbf{p}'\|\} \quad (8.7)$$

$$\text{Mean distance: } dist_{mean}(C_i, C_j) = \|\mathbf{m}_i - \mathbf{m}_j\| \quad (8.8)$$

$$\text{Average distance: } dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\mathbf{p} \in C_i, \mathbf{p}' \in C_j} \|\mathbf{p} - \mathbf{p}'\| \quad (8.9)$$

When an algorithm uses the *minimum distance*,  $d_{min}(C_i, C_j)$ , to measure the distance between clusters, it is sometimes called a **nearest-neighbor clustering algorithm** or **single-linkage algorithm**. If we view the data points as nodes of a graph, with edges forming a path between the nodes in a cluster, then the merging of two clusters,  $C_i$  and  $C_j$ , corresponds to adding an edge between the nearest pair of nodes in  $C_i$  and  $C_j$ .

When an algorithm uses the *maximum distance*,  $d_{max}(C_i, C_j)$ , to measure the distance between clusters, it is sometimes called a **farthest-neighbor clustering algorithm** or **complete-linkage algorithm**. By viewing data points as nodes of a graph, with edges linking nodes, we can think of each cluster as a *complete* subgraph, that is, with edges connecting all the nodes in the clusters. The distance between two clusters is determined by the most distant nodes in the two clusters.

Similar to the situation in the  $k$ -means method, single-linkage and complete-linkage methods are sensitive to outliers. The use of *mean* or *average distance* is a compromise between the minimum and maximum distances and overcomes the outlier sensitivity problem. Whereas the *mean distance* is the simplest to compute, the *average distance* is advantageous in that it can handle categoric data and

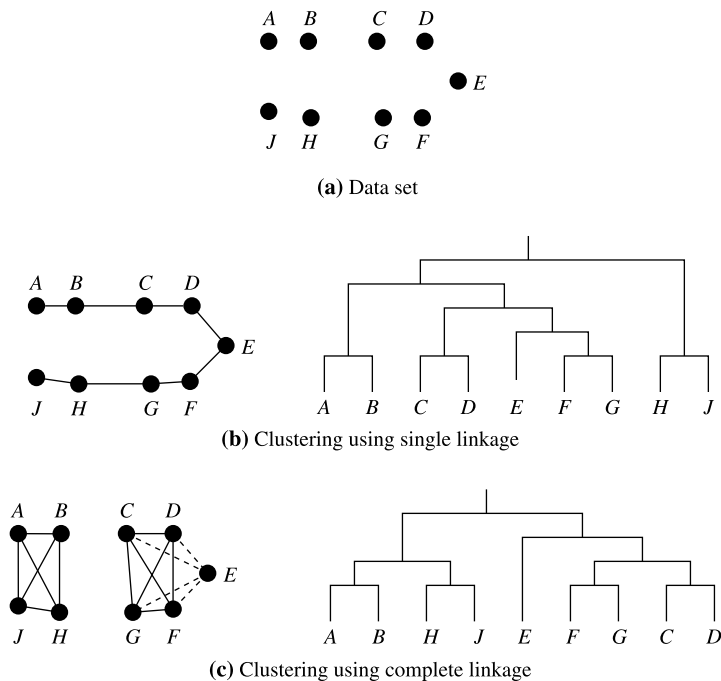


FIGURE 8.9

Hierarchical clustering using single and complete linkages.

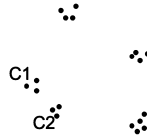
numeric data. The computation of the mean vector for categorical data can be difficult or impossible to define.

**Example 8.4. Single vs. complete linkages.** Let us apply hierarchical clustering to the data set of Fig. 8.9(a). Fig. 8.9(b) shows the hierarchy of clusters using single-linkage. Fig. 8.9(c) shows the case using complete linkage, where the edges between clusters  $\{A, B, J, H\}$  and  $\{C, D, G, F, E\}$  are omitted for ease of presentation. This example shows that by using single linkages we can find hierarchical clusters defined by local proximity, whereas complete linkage tends to find clusters opting for global closeness.  $\square$

There are variations of the four essential linkage measures just discussed. For example, we can measure the distance between two clusters by the distance between the centroids (i.e., the central objects) of the clusters.

### Connecting agglomerative hierarchical clustering and partitioning methods

Are there any connections between (agglomerative) hierarchical clustering and partitioning methods (Section 8.2)? Partitioning methods use the sum of squared errors (SSE) (Eq. (8.1)) to measure the compactness and quality of a possible clustering, that is a partitioning of points into clusters. Heuristically,



**FIGURE 8.10**

Ward's criterion.

agglomerative hierarchical clustering methods may also use the sum of squared errors (SSE) to guide the selection of clusters to merge.

For a data set of  $n$  points, if we set the number of clusters to  $n$ , a partitioning method naturally assigns each point into a cluster. This corresponds to the starting point of agglomerative hierarchical clustering. When we merge clusters in agglomerative clustering, we reduce the number of clusters. Which two clusters should we choose to merge? Heuristically, we may want to merge two clusters so that the resulting clustering also minimizes the sum of squared errors (SSE) (Eq. (8.1)), which is used as the criterion in partitioning methods like  $k$ -means.

Consider the five clusters in Fig. 8.10 as an illustrative example. Suppose we want to merge two of them into one so that we can have a hierarchy of clusters. Among all the possible pairs of clusters, merging  $C_1$  and  $C_2$  minimizes the SSE, and thus  $C_1$  and  $C_2$  should be merged in the next step in building the hierarchy.

The above intuition connecting agglomerative hierarchical clustering and partitioning methods gives us an alternative way to measure the similarity between two clusters. We can look at the increase of the SSE (Eq. (8.1)) if the two clusters are merged into one, the smaller the better. This is formulated by J.H. Ward and thus is known as **Ward's criterion**.

Suppose two disjoint clusters  $C_i$  and  $C_j$  are merged, and  $m_{(ij)}$  is the mean of the new cluster. Then, Ward's criterion is defined as

$$\begin{aligned} W(C_i, C_j) &= \sum_{\mathbf{x} \in C_i \cup C_j} \|\mathbf{x} - m_{(ij)}\|^2 - \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - m_i\|^2 - \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - m_j\|^2 \\ &= \frac{n_i n_j}{n_i + n_j} \|m_i - m_j\|^2. \end{aligned}$$

### **The Lance-Williams algorithm**

We discuss a few different proximity measures for clusters in agglomerative hierarchical clustering, is there a way to generalize them? Indeed, the **Lance-Williams formula** generalizes different measures in a uniform way. Suppose two exclusive clusters  $C_i$  and  $C_j$  are merged. We need to specify the distance between the merged cluster, denoted by  $C_{(ij)}$ , and every other cluster  $C_k$ . The similarity between the merged clusters  $C_{(ij)}$  and cluster  $C_k$  is given by

$$d(C_{(ij)}, C_k) = \alpha_i d(C_i, C_k) + \alpha_j d(C_j, C_k) + \beta d(C_i, C_j) + \gamma |d(C_i, C_k) - d(C_j, C_k)|,$$

where  $\alpha_i$ ,  $\alpha_j$ ,  $\beta$ , and  $\gamma$  are parameters that together with the similarity function  $d(C_i, C_j)$  determine the hierarchical clustering algorithm. As shown in the formula, the similarity between  $C_{(ij)}$  and  $C_k$  is decided by four terms. The first two terms are the similarities between  $C_i$  and  $C_j$  to  $C_k$ , respectively.

The third term relies on the similarity between  $C_i$  and  $C_j$ . The last term represents how the difference of the original similarities between  $C_i$  and  $C_j$  to  $C_k$  may contribute to the new similarity.

For example, in the exercise, you are asked to verify that the single-linkage method is equivalent to taking  $\alpha_i = \alpha_j = 0.5$ ,  $\beta = 0$ , and  $\gamma = -0.5$  in the Lance-Williams formula. Record that in the single link method, the similarity between two clusters is determined by the similarity between of the closest pair of data points belonging to different clusters. Thus the above parameters simply pick the smaller one between  $d(C_i, C_k)$  and  $d(C_j, C_k)$ . You can also verify that the complete-linkage method is equivalent to  $\alpha_i = \alpha_j = 0.5$ ,  $\beta = 0$ , and  $\gamma = 0.5$ . Moreover, to implement the Ward's criterion, we can take  $\alpha_i = \frac{n_i+n_k}{n_i+n_j+n_k}$ ,  $\alpha_j = \frac{n_j+n_k}{n_i+n_j+n_k}$ ,  $\beta = -\frac{n_k}{n_i+n_j+n_k}$ , and  $\gamma = 0$  in the Lance-Williams formula.

Using the Lance-Williams formula, the *Lance-Williams algorithm* generalizes agglomerative hierarchical clustering. It takes an agglomerative way and minimizes the sum of distance in each iteration until all points are merged into one cluster.

### 8.3.3 Divisive hierarchical clustering

A divisive hierarchical clustering method partitions a set of objects step by step into clusters. To design a divisive hierarchical clustering method, there are three important issues to consider.

First, a set of objects can be split in many different ways. A splitting criterion is needed to determine which splitting is the best. Technically, given two splittings, the splitting criterion should be able to tell which one is better. For example, the SSE (Eq. (8.1)) may be used on numeric data. If two splittings have the same number of clusters, then the one with a less SSE value is preferred. On nominal data, the Gini index (Chapter 6) may be used. The choice of splitting criteria is one of the most important decisions in designing a divisive hierarchical clustering method, since it determines what clustering results that the method may lead to.

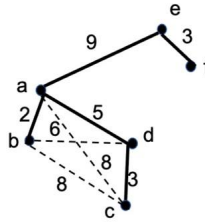
Second, when we decide to split a cluster, we need to design a splitting method. While in general the splitting method should optimize the splitting criterion, one major consideration is the computational cost. For example, enumerating all possible splittings and finding the best one is likely computationally prohibitive. Thus some heuristic or approximate methods, such as bisecting  $k$ -means, that is, setting  $k = 2$ , may be used.

Third, in the middle of divisive hierarchical clustering, there are in general multiple clusters. Then, which cluster should be split next? An intuitive idea is to choose the “loosest” cluster. More concretely, we may compute the average SSE of each cluster,  $E_{C_i} = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{m}_i)^2$ , and choose the one with the largest average SSE to split.

#### *The minimum spanning tree–based approach*

Let us use the minimum spanning tree–based approach to illustrate the basic idea of divisive hierarchical clustering. In a weighted graph  $G$ , a minimum spanning tree is an acyclic subgraph that contains all nodes in  $G$ , and the sum of edge weights of the tree is minimized. For example, consider the weighted graph in Fig. 8.11, where the edges of weights up to nine are shown in the figure, and the edges of weights over nine are omitted. The minimum spanning tree consists of the edges in solid lines, whereas the edges in dashed lines and those omitted are not included in the minimum spanning tree. Minimum spanning tree can be computed using, for example, Prim's algorithm and Kruskal's algorithm.

Given a set of points, we can construct a weighted graph such that each point is represented by a node in the graph, and the distance between two points is the weight of the edge connecting the two



**FIGURE 8.11**

A weighted graph and a minimum spanning tree.

corresponding nodes in the graph. Then, we can compute the minimum spanning tree of the weighted graph. Intuitively, the minimum spanning tree can be regarded as the most compact way that the points are connected into one cluster. In general, in the process of divisive hierarchical clustering based on minimum spanning tree, every cluster is a subset of nodes and is represented by the minimum spanning tree, which is a subtree of the minimum spanning tree of the whole data set. The splitting criterion is the total weights of all the edges in the spanning tree(s) of all the clusters, the smaller the better.

Based on this spanning tree, we can progressively divide the set of points in one cluster into smaller clusters. Suppose we want to conduct bisecting splitting, that is, every time we split one cluster into two smaller clusters. At each step, we consider all edges in the spanning trees of the current clusters and delete the edge of the largest weight. Deleting an edge in a tree divides one cluster into two. Thus the splitting method is to divide a cluster by deleting the edge in the minimum spanning tree of the largest weight.

For example, consider a set of points  $\{a, b, c, d, e, f\}$  as shown in Fig. 8.11. A weighted edge and the corresponding distance are plotted in the figure if the distance between two points is smaller than 10. Based on the minimum spanning tree method, the edge  $(a, e)$ , which has the largest weight in the minimum spanning tree, is first deleted, which divides the data set into two clusters,  $\{a, b, c, d\}$  and  $\{e, f\}$ . Next, by deleting edge  $(a, d)$ , which has the largest weight in the remaining minimum spanning trees, the cluster  $\{a, b, c, d\}$  is split into two smaller clusters  $\{a, b\}$  and  $\{c, d\}$ . The process continues until each cluster has only one point and all the edges in the minimum spanning tree are deleted.

### **Dendrogram**

A tree structure called a **dendrogram** is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together (in an agglomerative method) or partitioned (in a divisive method) step-by-step. Fig. 8.12 shows a dendrogram for the five objects presented in Fig. 8.8, where  $l = 0$  shows the five objects as singleton clusters at level 0. At  $l = 1$ , objects  $a$  and  $b$  are grouped together to form the first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects,  $\{a, b\}$  and  $\{c, d, e\}$ , is roughly 0.16, they are merged together to form a single cluster.

A complete dendrogram shows every data point as a node at the leaf level and the whole data set as one cluster at the root. In practice, however, too small clusters may not be very meaningful and too many such small clusters can be overwhelming. Thus a data analyst often shows and considers only the portion close to the root of a dendrogram. Moreover, when the graph contains a sufficiently small



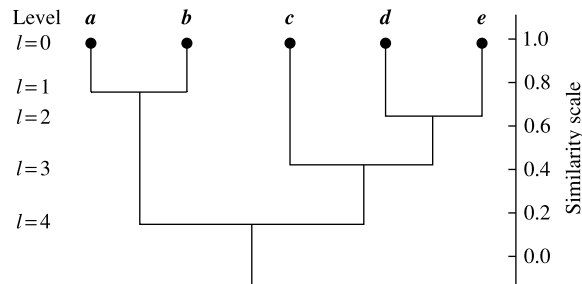


FIGURE 8.12

Dendrogram representation for hierarchical clustering of data objects  $\{a, b, c, d, e\}$ .

number of clusters, further merging them into even bigger ones may counter the objective of clustering analysis. Therefore the very root part of a dendrogram may also be ignored in analysis.

### 8.3.4 BIRCH: scalable hierarchical clustering using clustering feature trees

There are two major difficulties in the agglomerative and divisive hierarchical clustering methods discussed so far. First, all those methods cannot revisit any merge or split decisions made before. Thus an improper decision based on limited information may lead to low quality final clustering results. Moreover, scalability is a major bottleneck, since each merge or split needs to examine many possible options. To overcome those difficulties, we may conduct hierarchical clustering in multiple phases, so that clustering results can be improved over phases. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is such a method and is designed for clustering a large amount of numeric data by integrating hierarchical clustering (at the initial *microclustering* stage) and other clustering methods such as iterative partitioning (at the later *macroclustering* stage).

BIRCH uses the notions of *clustering feature* to summarize a cluster, and *clustering feature tree* (*CF-tree*) to represent a cluster hierarchy. These structures help the clustering method achieve good speed and scalability in large or even streaming databases and also make it effective for incremental and dynamic clustering of incoming objects.

Consider a cluster of  $n$   $d$ -D data objects or points. The **clustering feature (CF)** of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as

$$CF = \langle n, LS, SS \rangle, \quad (8.10)$$

where  $LS$  is the linear sum of the  $n$  points (i.e.,  $LS = \sum_{i=1}^n \mathbf{x}_i$ ) and  $SS$  is the square sum of the data points (i.e.,  $SS = \sum_{i=1}^n \|\mathbf{x}_i\|^2$ ).

A clustering feature is essentially a summary of the statistics for the given cluster. Using a clustering feature, we can easily derive many useful statistics of a cluster. For example, the cluster's centroid,  $\mathbf{x}_0$ , radius,  $R$ , and diameter,  $D$ , are

$$\mathbf{x}_0 = \frac{\sum_{i=1}^n \mathbf{x}_i}{n} = \frac{LS}{n}, \quad (8.11)$$

$$R = \sqrt{\frac{\sum_{i=1}^n (x_i - x_0)^2}{n}} = \sqrt{\frac{SS}{n} - \left(\frac{\|\mathbf{LS}\|}{n}\right)^2}, \quad (8.12)$$

$$D = \sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2\|\mathbf{LS}\|^2}{n(n-1)}}. \quad (8.13)$$

Here,  $R$  is the average distance from member objects to the centroid, and  $D$  is the average pairwise distance within a cluster. Both  $R$  and  $D$  reflect the tightness of the cluster around the centroid.

Summarizing a cluster using the clustering feature can avoid storing the detailed information about individual objects or points. Instead, we only need a constant size of space to store the clustering feature. This is the key to the efficiency of BIRCH in space. Moreover, clustering features are *additive*. That is, for two disjoint clusters,  $C_1$  and  $C_2$ , with the clustering features  $\mathbf{CF}_1 = \langle n_1, \mathbf{LS}_1, SS_1 \rangle$  and  $\mathbf{CF}_2 = \langle n_2, \mathbf{LS}_2, SS_2 \rangle$ , respectively, the clustering feature for the cluster that formed by merging  $C_1$  and  $C_2$  is simply

$$\mathbf{CF}_1 + \mathbf{CF}_2 = \langle n_1 + n_2, \mathbf{LS}_1 + \mathbf{LS}_2, SS_1 + SS_2 \rangle. \quad (8.14)$$

**Example 8.5. Clustering feature.** Suppose there are three points, (2, 5), (3, 2), and (4, 3), in a cluster,  $C_1$ . The clustering feature of  $C_1$  is

$$\mathbf{CF}_1 = \langle 3, (2 + 3 + 4, 5 + 2 + 3), (2^2 + 3^2 + 4^2) + (5^2 + 2^2 + 3^2) \rangle = \langle 3, (9, 10), 67 \rangle.$$

Suppose that  $C_1$  is disjoint to a second cluster,  $C_2$ , where  $\mathbf{CF}_2 = \langle 3, (35, 36), 857 \rangle$ . The clustering feature of a new cluster,  $C_3$ , that is formed by merging  $C_1$  and  $C_2$ , is derived by adding  $\mathbf{CF}_1$  and  $\mathbf{CF}_2$ . That is,

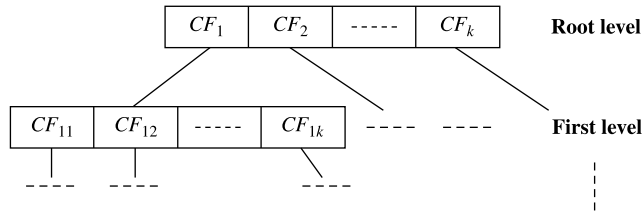
$$\mathbf{CF}_3 = \langle 3 + 3, (9 + 35, 10 + 36), 67 + 857 \rangle = \langle 6, (44, 46), 924 \rangle.$$

□

A **CF-tree** is a height-balanced tree that stores the clustering features for a hierarchical clustering. An example is shown in Fig. 8.13. By definition, a nonleaf node in a tree has descendants or “children.” The nonleaf nodes store sums of the CFs of their children and thus summarize clustering information about their children. A CF-tree has two parameters: *branching factor*,  $B$ , and *threshold*,  $T$ . The branching factor specifies the maximum number of children per nonleaf node. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters implicitly control the size of the resulting CF-tree.

Given a limited amount of main memory, an important consideration in BIRCH is to minimize the time required for input/output (I/O). BIRCH applies a *multiphase* clustering technique: A single scan of the data set yields a basic, good clustering, and one or more additional scans can optionally be used to further improve the quality. The primary phases are as follows:

- **Phase 1:** BIRCH scans the database to build an initial in-memory CF-tree, which can be viewed as a multilevel compression of the data that tries to preserve the inherent clustering structure of the data.



**FIGURE 8.13**

CF-tree structure.

- **Phase 2:** BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF-tree, which removes sparse clusters as outliers and groups dense clusters into larger ones.

For Phase 1, the CF-tree is built dynamically as objects are inserted. Thus the method is incremental. An object is inserted into the closest leaf entry (subcluster). If the diameter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After the insertion of the new object, information about the object is passed toward the root of the tree. The size of the CF-tree can be changed by modifying the threshold. If the size of the memory that is needed for storing the CF-tree is larger than the size of the main memory available, then a larger threshold value can be specified and the CF-tree is rebuilt.

The rebuild process is performed by building a new tree from the leaf nodes of the old tree. Thus the process of rebuilding the tree is done without the necessity of rereading all the objects or points. This is similar to the insertion and node split in the construction of B+-trees. Therefore for building the tree, data have to be read just once. Some heuristics and methods have been introduced to deal with outliers and improve the quality of CF-trees by additional scans of the data. Once the CF-tree is built, any clustering algorithm, such as a typical partitioning algorithm, can be used with the CF-tree in Phase 2.

“How effective is BIRCH?” The time complexity of the algorithm is  $O(n)$ , where  $n$  is the number of objects to be clustered. Experiments have shown the linear scalability of the algorithm with respect to the number of objects, and good quality of clustering of data. However, since each node in a CF-tree can hold only a limited number of entries due to its size, a CF-tree node does not always correspond to what a user may consider a natural cluster. Moreover, if the clusters are not spherical in shape, BIRCH does not perform well because it uses the notion of radius or diameter to control the boundary of a cluster.

The ideas of clustering features and CF-trees have been applied beyond BIRCH. The ideas have been borrowed by many others to tackle problems of clustering streaming and dynamic data.

### 8.3.5 Probabilistic hierarchical clustering

Hierarchical clustering methods using linkage measures tend to be easy to understand and are often efficient in clustering. They are commonly used in many clustering analysis applications. However, hierarchical clustering methods can suffer from several drawbacks. First, choosing a good distance measure for hierarchical clustering is often far from trivial. Second, to apply such a method, the data

objects cannot have any missing attribute values. In the case where data is partially observed (i.e., some attribute values of some objects are missing), it is not easy to apply a hierarchical clustering method because the distance computation cannot be conducted. Third, most of the hierarchical clustering methods are heuristic and search locally at each step for a good merging/splitting decision. Consequently, the optimization goal of the resulting cluster hierarchy can be unclear.

**Probabilistic hierarchical clustering** aims to overcome some of these disadvantages by using probabilistic models to measure distances between clusters.

One way to look at the clustering problem is to regard the set of data objects to be clustered as a sample of the underlying data generation mechanism to be analyzed or, formally, the *generative model*. For example, when we conduct clustering analysis on a set of marketing surveys, we assume that the surveys collected are a sample of the opinions of all possible customers. Here, the data generation mechanism is a probability distribution of opinions with respect to different customers, which cannot be obtained directly and completely. The task of clustering is to estimate the generative model as accurately as possible using the observed data objects to be clustered.

In practice, we can assume that the data generative models adopt common distribution functions, such as Gaussian distribution or Bernoulli distribution, which are governed by parameters. The task of learning a generative model is then reduced to finding the parameter values for which the model best fits the observed data set.

**Example 8.6. Generative model.** Suppose we are given a set of 1-D points  $X = \{x_1, \dots, x_n\}$  for clustering analysis. Let us assume that the data points are generated by a Gaussian distribution,

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (8.15)$$

where the parameters are  $\mu$  (the mean) and  $\sigma^2$  (the variance).

The probability that a point  $x_i \in X$  is then generated by the model is

$$P(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \quad (8.16)$$

Consequently, the likelihood that the data set  $X$  observed is generated by the model is

$$L(\mathcal{N}(\mu, \sigma^2) : X) = P(X|\mu, \sigma^2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \quad (8.17)$$

The task of learning the generative model is to find the parameters  $\mu$  and  $\sigma^2$  such that the likelihood  $L(\mathcal{N}(\mu, \sigma^2) : X)$  is maximized, that is, finding

$$\mathcal{N}(\mu_0, \sigma_0^2) = \arg \max\{L(\mathcal{N}(\mu, \sigma^2) : X)\}, \quad (8.18)$$

where  $\max\{L(\mathcal{N}(\mu, \sigma^2) : X)\}$  is called the *maximum likelihood*.  $\square$

Given a set of objects, the quality of a cluster formed by all the objects can be measured by the maximum likelihood. For a set of objects partitioned into  $m$  clusters  $C_1, \dots, C_m$ . Then the quality can

be measured by

$$Q(\{C_1, \dots, C_m\}) = \prod_{i=1}^m P(C_i), \quad (8.19)$$

where  $P()$  is the maximum likelihood. To calculate  $P(C_i)$ , we can fit each cluster  $C_i$  ( $1 \leq i \leq m$ ) by a generative model  $M_i$ , and estimate the probability by  $P(C_i) = \prod_{x \in C_i} P(x|M_i)$ . If we merge two clusters,  $C_{j_1}$  and  $C_{j_2}$ , into a cluster,  $C_{j_1} \cup C_{j_2}$ , then, the change in quality of the overall clustering is

$$\begin{aligned} & Q(\{C_1, \dots, C_m\} - \{C_{j_1}, C_{j_2}\} \cup \{C_{j_1} \cup C_{j_2}\}) - Q(\{C_1, \dots, C_m\}) \\ &= \frac{\prod_{i=1}^m P(C_i) \cdot P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - \prod_{i=1}^m P(C_i) \\ &= \prod_{i=1}^m P(C_i) \left( \frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - 1 \right). \end{aligned} \quad (8.20)$$

When choosing to merge two clusters in hierarchical clustering,  $\prod_{i=1}^m P(C_i)$  is constant for any pair of clusters. Therefore given clusters  $C_1$  and  $C_2$ , the dissimilarity between them can be measured by

$$dist(C_1, C_2) = -\log \frac{P(C_1 \cup C_2)}{P(C_1)P(C_2)}. \quad (8.21)$$

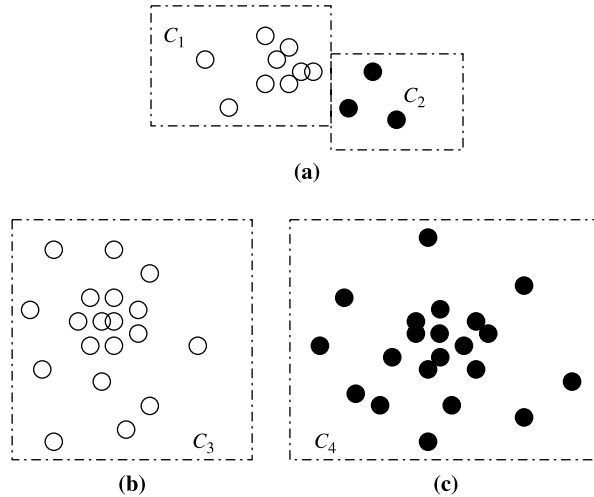
A probabilistic hierarchical clustering method can adopt the agglomerative clustering framework, but use probabilistic models (Eq. (8.21)) to measure the similarity between clusters.

Upon close observation of Eq. (8.20), we see that merging two clusters may not always lead to an improvement in clustering quality, that is,  $\frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})}$  may be less than 1. For example, assume that Gaussian distribution functions are used in the model of Fig. 8.14. Although merging clusters  $C_1$  and  $C_2$  results in a cluster that better fits a Gaussian distribution, merging clusters  $C_3$  and  $C_4$  lowers the clustering quality because no Gaussian functions can fit the merged cluster well.

Based on this observation, a probabilistic hierarchical clustering scheme can start with one cluster per object and merge two clusters,  $C_i$  and  $C_j$ , if the distance between them is negative. In each iteration, we try to find  $C_i$  and  $C_j$  so as to maximize  $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$ . The iteration continues as long as  $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$ , that is, as long as there is an improvement in clustering quality. The pseudocode is given in Fig. 8.15.

Probabilistic hierarchical clustering methods are easy to understand and generally have the same efficiency as agglomerative hierarchical clustering methods; in fact, they share the same framework. Probabilistic models are more interpretable, but sometimes less flexible than distance metrics. Probabilistic models can handle partially observed data. For example, given a multidimensional data set where some objects have missing values on some dimensions, we can learn a Gaussian model on each dimension independently using the observed values on the dimension. The resulting cluster hierarchy accomplishes the optimization goal of fitting data to the selected probabilistic models.

A drawback of using probabilistic hierarchical clustering is that it outputs only one hierarchy with respect to a chosen probabilistic model. It cannot handle the uncertainty of cluster hierarchies. Given a

**FIGURE 8.14**

Merging clusters in probabilistic hierarchical clustering: (a) Merging clusters  $C_1$  and  $C_2$  leads to an increase in overall cluster quality, but merging clusters (b)  $C_3$  and (c)  $C_4$  does not.

**Algorithm:** A probabilistic hierarchical clustering algorithm.

**Input:**

- $D = \{o_1, \dots, o_n\}$ : a data set containing  $n$  objects;

**Output:** A hierarchy of clusters.

**Method:**

- (1) **create** a cluster for each object  $C_i = \{o_i\}$ ,  $1 \leq i \leq n$ ;
- (2) **for**  $i = 1$  to  $n$
- (3)     **find** pair of clusters  $C_i$  and  $C_j$  such that  $C_i, C_j = \arg \max_{i \neq j} \log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$ ;
- (4)     **if**  $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$  **then** merge  $C_i$  and  $C_j$ ;
- (5)     **else** stop;

**FIGURE 8.15**

A probabilistic hierarchical clustering algorithm.

data set, there may exist multiple hierarchies that fit the observed data. Neither algorithmic approaches nor probabilistic approaches can find the distribution of such hierarchies. Recently, Bayesian tree-structured models have been developed to handle such problems. Bayesian and other sophisticated probabilistic clustering methods are considered advanced topics and are not covered in this book.

## 8.4 Density-based and grid-based methods

Most of the partitioning and hierarchical methods are designed to find spherical-shaped clusters. They have difficulty finding clusters of arbitrary shape such as the “S” shape and oval clusters in Fig. 8.16.



FIGURE 8.16

Clusters of arbitrary shape.

Although some feature transformation methods, such as kernel  $k$ -means, may help, it is often tricky to choose appropriate kernel functions. Given such data, they would likely inaccurately identify convex regions, where noises or outliers are included in the clusters.

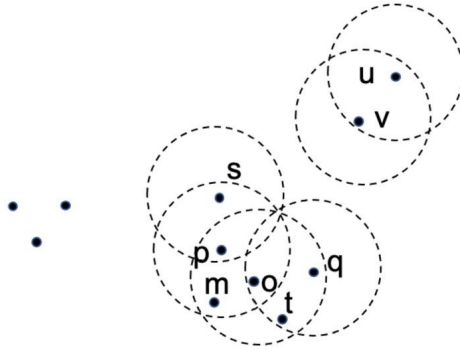
To find clusters of arbitrary shape, alternatively, we can model clusters as dense regions in the data space, separated by sparse regions. This is the main strategy behind *density-based clustering methods*, which can discover clusters of nonspherical shape. In this section, you will learn the basic techniques of density-based clustering by studying two representative methods, namely, DBSCAN (Section 8.4.1) and DENCLUE (Section 8.4.2). To tackle the computational cost in density-based clustering, the data space may be partitioned into a grid. This idea motivates the grid-based clustering methods (Section 8.4.3).

### 8.4.1 DBSCAN: density-based clustering based on connected regions with high density

“How can we find dense regions in density-based clustering?” The *density* of an object  $o$  can be measured by the number of objects close to  $o$ . **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) finds *core objects*, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters. In this section, let us explain **DBSCAN\***, an improved version of the original **DBSCAN**.

“How does **DBSCAN\*** quantify the neighborhood of an object?” **DBSCAN\*** employs a user-specified parameter  $\epsilon > 0$  to specify the radius of a neighborhood that is considered for every object. The  $\epsilon$ -**neighborhood** of an object  $o$  is the space within a radius  $\epsilon$  centered at  $o$ .

Due to the fixed neighborhood size parameterized by  $\epsilon$ , the **density of a neighborhood** can be measured simply by the number of objects in the neighborhood. To determine whether a neighborhood is dense or not, **DBSCAN\*** uses another user-specified parameter, *MinPts*, which specifies the density



**FIGURE 8.17**

Density-reachability and density-connectivity in DBSCAN.

threshold of dense regions. An object is a **core object** if the  $\epsilon$ -neighborhood of the object contains at least  $MinPts$  objects, otherwise, it is **noise**. Core objects are the pillars of dense regions.

Given a set,  $D$ , of objects, we can identify all core objects with respect to the given parameters,  $\epsilon$  and  $MinPts$ . The clustering task is therein reduced to using core objects and their neighborhoods to form dense regions, where the dense regions are clusters.

Two core objects  $p$  and  $q$  are  **$\epsilon$ -reachable** if  $d(p, q) \leq \epsilon$ , that is,  $p$  is in the  $\epsilon$ -neighbor of  $q$  and vice versa. Two core objects  $p$  and  $q$  are **density-connected** if  $p$  and  $q$  are  $\epsilon$ -reachable or transitively  $\epsilon$ -reachable, where  $p$  and  $q$  are transitively  $\epsilon$ -reachable if there exist one or multiple core objects  $r_1, \dots, r_l$  such that  $p$  and  $r_1$  are  $\epsilon$ -reachable,  $r_i$  and  $r_{i+1}$  ( $1 \leq i < l$ ) are  $\epsilon$ -reachable, and  $r_l$  and  $q$  are  $\epsilon$ -reachable. Then, a cluster  $C$  with respect to parameters  $\epsilon$  and  $MinPts$  is simply a nonempty maximal subset of core objects that every pair of objects in  $C$  is density connected.

In DBSCAN\*, a cluster contains only core objects. However, it is easy to assign a noncore object that is in the  $\epsilon$ -neighborhood of a core object to the cluster containing that core object. DBSCAN explicitly identifies such noncore objects as **border objects**, and DBSCAN\* can use a postprocessing step to pick up those border objects. Those objects that do not belong to any  $\epsilon$ -neighborhood are outliers.

**Example 8.7. Density-reachability and density-connectivity.** Consider Fig. 8.17 for a given  $\epsilon$  represented by the radius of the circles, and, say, let  $MinPts = 3$ .

Of the labeled objects,  $p, m, o, q,$  and  $t$  are core objects, since each of the  $\epsilon$ -neighborhoods (dashed circles in the figure) of them contains at least three objects. Objects  $p$  and  $o$  are  $\epsilon$ -reachable, so are  $o$  and  $q$ . Thus  $p$  and  $q$  are density-connected.

It can be verified that the core objects  $p, m, o, q,$  and  $t$  form a cluster, since each two among them are density-connected and no other core objects can be added into this group so that the pairwise density-connectivity is maintained.

Object  $s$  is not a core object, since the  $\epsilon$ -neighborhood of  $s$  contains only two objects. However,  $s$  is in the  $\epsilon$ -neighborhood of core object  $p$ , thus  $s$  is a border object.

Objects  $u$  and  $v$  are not core objects, and they do not belong to the  $\epsilon$ -neighborhood of any core objects. Thus they are outliers.  $\square$



**Algorithm: DBSCAN\*:** a density-based clustering algorithm.

**Input:**

- $D$ : a data set containing  $n$  objects,
- $\epsilon$ : the radius parameter, and
- $MinPts$ : the neighborhood density threshold.

**Output:** A set of density-based clusters.

**Method:**

```

(1) mark all objects as unvisited;
(2) do
(3)   randomly select an unvisited object  $p$ ;
(4)   mark  $p$  as visited;
(5)   if the  $\epsilon$ -neighborhood of  $p$  has at least  $MinPts$  objects
(6)     create a new cluster  $C$ , and add  $p$  to  $C$ ;
(7)     let  $N$  be the set of objects in the  $\epsilon$ -neighborhood of  $p$ ;
(8)     for each point  $p'$  in  $N$ 
(9)       if  $p'$  is unvisited
(10)        mark  $p'$  as visited;
(11)        if the  $\epsilon$ -neighborhood of  $p'$  has at least  $MinPts$  points,
            add those points to  $N$  and add  $p'$  to  $C$ ;
(12)     end for
(13)     output  $C$ ;
(14)   else mark  $p$  as noise;
(15) until no object is unvisited;

```

**FIGURE 8.18**

DBSCAN\* algorithm.

“How does DBSCAN\* find clusters?” Initially, all objects in a given data set  $D$  are marked as “unvisited.” DBSCAN\* randomly selects an unvisited object  $p$ , marks  $p$  as “visited,” and checks whether the  $\epsilon$ -neighborhood of  $p$  contains at least  $MinPts$  objects. If not,  $p$  is marked as a noise point. Otherwise, a new cluster  $C$  is created for  $p$ , and all the objects in the  $\epsilon$ -neighborhood of  $p$  are added to a candidate set,  $N$ .

DBSCAN\* iteratively adds to  $C$  those core objects in  $N$  that do not belong to any cluster. In this process, for an object  $p'$  in  $N$  that carries the label “unvisited,” DBSCAN\* marks it as “visited” and checks its  $\epsilon$ -neighborhood. If the  $\epsilon$ -neighborhood of  $p'$  has at least  $MinPts$  objects,  $p'$  is labeled as a core object and added into  $C$ , those objects in the  $\epsilon$ -neighborhood of  $p'$  are added to  $N$ . DBSCAN\* continues adding objects to  $C$  until  $C$  can no longer be expanded, that is,  $N$  is empty. At this time, cluster  $C$  is completed, and thus is output.

To find the next cluster, DBSCAN\* randomly selects an unvisited object from the remaining ones. The clustering process continues until all objects are visited. The pseudocode of the DBSCAN\* algorithm is given in Fig. 8.18.

If a spatial index is used, the computational complexity of DBSCAN\* is  $O(n \log n)$ , where  $n$  is the number of database objects. Otherwise, the complexity is  $O(n^2)$ . With appropriate settings of the user-defined parameters,  $\epsilon$  and  $MinPts$ , the algorithm is effective in finding arbitrary-shaped clusters.

*It is not easy to specify two parameters,  $\epsilon$  and  $MinPts$ , in DBSCAN\*. Moreover, density-based clusters may also have hierarchies. For example, within a dense area there may be a sub-area is sub-*

*stantially denser. Can we find hierarchical density-based clusters?* Indeed, DBSCAN\* can be extended to HDBSCAN\*, which achieves density-based hierarchical clustering.

HDBSCAN\* only takes one parameter, *MinPts*. For an object  $p$ , the **core distance** of  $p$ , denoted by  $d_{core}(p)$  is the distance from  $p$  to its *MinPts*th nearest neighbor (including  $p$  itself). In other words,  $d_{core}(p)$  is the minimum radius with respect to which  $p$  is a core object in DBSCAN\*. For two objects  $p$  and  $q$ , the **mutual reachability distance** between them is  $d_{mreach}(p, q) = \max\{d_{core}(p), d_{core}(q), d(p, q)\}$ . In other words,  $d_{mreach}(p, q)$  is the minimum radius  $\epsilon$  such that  $p$  and  $q$  are  $\epsilon$ -reachable in DBSCAN\*.

Given a set of objects as the input to HDBSCAN\*, we can construct a **mutual reachability graph**  $G_{MinPts}$ , which is a complete graph. Every object in the input is a node in the mutual reachability graph. The weight of the edge between  $p$  and  $q$  is the mutual reachability distance  $d_{mreach}(p, q)$ . We can apply the minimum spanning tree approach (Section 8.3.3) on the mutual reachability graph to find density-based hierarchical clusters.

To further reduce the demand of setting parameters, a cluster analysis method called **OPTICS** was proposed. OPTICS does not explicitly produce a data set clustering. Instead, it outputs a **cluster ordering**, a linear list of all objects under analysis, representing the *density-based clustering structure* of the data. Objects in a denser cluster are listed closer to each other in the cluster ordering. This ordering is equivalent to density-based clustering obtained from a wide range of parameter settings. Thus, OPTICS does not require the user to provide a specific density threshold. The cluster ordering can be used to extract basic clustering information (e.g., cluster centers, or arbitrary-shaped clusters), derive the intrinsic clustering structure, and provide a visualization of the clustering.

To construct the different clusterings simultaneously, the objects are processed in a specific order. This order selects an object that is density-reachable with respect to the lowest  $\epsilon$  value so that clusters with higher density (i.e., lower  $\epsilon$ ) will be finished first. For example, Fig. 8.19 shows the reachability plot for a simple 2-D data set, which presents a general overview of how the data are structured and clustered. The data objects are plotted in the clustering order (horizontal axis) together with their respective reachability-distances (vertical axis). The three Gaussian “bumps” in the plot reflect three clusters in the data set.

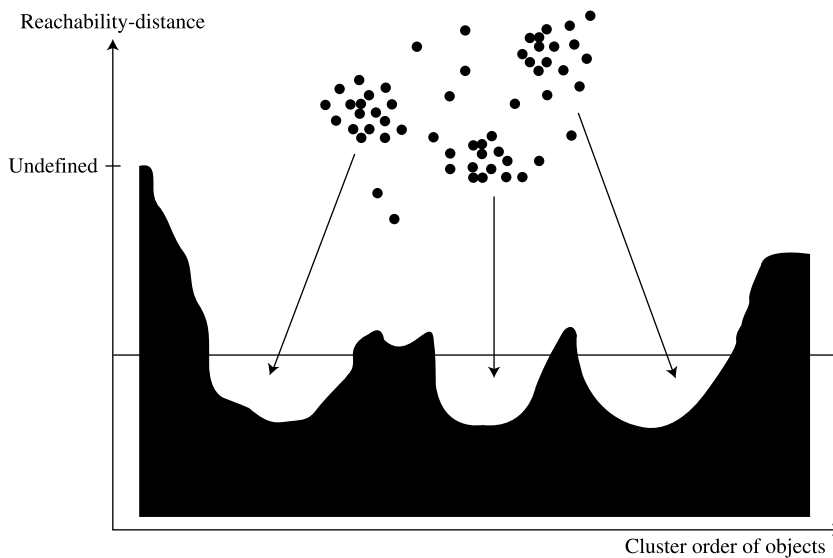
OPTICS can be seen as a generalization of DBSCAN that replaces the  $\epsilon$  parameter with a maximum value that mostly affects performance. *MinPts* then essentially becomes the minimum cluster size to find. While the algorithm is much easier to parameterize than DBSCAN, it usually produces a hierarchical clustering instead of the simple data partitioning that DBSCAN produces.

## 8.4.2 DENCLUE: clustering based on density distribution functions

Density estimation is a core issue in density-based clustering. **DENCLUE** (DENSITY-based CLUstEring) is a clustering method based on a set of density distribution functions. We first give some background on density estimation and then describe the DENCLUE algorithm.

In probability and statistics, **density estimation** is the estimation of an unobservable underlying probability density function based on a set of observed data. In the context of density-based clustering, the unobservable underlying probability density function is the true distribution of the population of all possible objects to be analyzed. The observed data set is regarded as a random sample from that population.

In DENCLUE, **kernel density estimation** is used, which is a nonparametric density estimation approach from statistics. The general idea behind kernel density estimation is simple. We treat an observed



**FIGURE 8.19**

Cluster ordering in OPTICS. *Source:* Adapted from Ankerst, Breunig, Kriegel, and Sander [ABKS99].

object as an indicator of high-probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects.

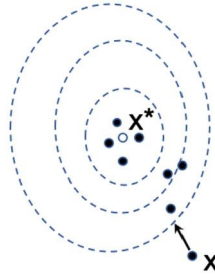
Formally, let  $\mathbf{x}_1, \dots, \mathbf{x}_n$  be an independent and identically distributed sample of a random variable  $f$ . The *kernel density approximation of the probability density function* is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \quad (8.22)$$

where  $K()$  is a kernel function and  $h$  is the bandwidth serving as a smoothing parameter. A **kernel** can be regarded as a function modeling the influence of a sample point within its neighborhood. Technically, a kernel  $K()$  is a nonnegative real-valued integrable function that should satisfy two requirements:  $\int_{-\infty}^{+\infty} K(u)du = 1$  and  $K(-u) = K(u)$  for all values of  $u$ . A frequently used kernel is a standard Gaussian function with a mean of 0 and a variance of 1:

$$K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(\mathbf{x} - \mathbf{x}_i)^2}{2h^2}}. \quad (8.23)$$

DENCLUE uses a Gaussian kernel to estimate density based on the given set of objects to be clustered. A point  $\mathbf{x}^*$  is called a **density attractor** if it is a local maximum of the estimated density function. To avoid trivial local maximum points, DENCLUE uses a noise threshold,  $\xi$ , and only considers those density attractors  $\mathbf{x}^*$  such that  $\hat{f}(\mathbf{x}^*) \geq \xi$ . These nontrivial density attractors are the centers of clusters.

**FIGURE 8.20**

Hill-climbing in DENCLUE.

The objects under analysis are assigned to clusters through density attractors using a stepwise hill-climbing procedure. For an object,  $\mathbf{x}$ , the hill-climbing procedure starts from  $\mathbf{x}$  and is guided by the gradient of the estimated density function. That is, the density attractor for  $\mathbf{x}$  is computed as

$$\begin{aligned} \mathbf{x}^0 &= \mathbf{x} \\ \mathbf{x}^{j+1} &= \mathbf{x}^j + \delta \frac{\nabla \hat{f}(\mathbf{x}^j)}{|\nabla \hat{f}(\mathbf{x}^j)|}, \end{aligned} \quad (8.24)$$

where  $\delta$  is a parameter to control the speed of convergence, and

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{h^{d+2n} \sum_{i=1}^n K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) (\mathbf{x}_i - \mathbf{x})}. \quad (8.25)$$

The hill-climbing procedure stops at step  $k > 0$  if  $\hat{f}(\mathbf{x}^{k+1}) < \hat{f}(\mathbf{x}^k)$ , and assigns  $\mathbf{x}$  to the density attractor  $\mathbf{x}^* = \mathbf{x}^k$ . An object  $\mathbf{x}$  is an outlier or noise if it converges in the hill-climbing procedure to a local maximum  $\mathbf{x}^*$  with  $\hat{f}(\mathbf{x}^*) < \xi$ .

Fig. 8.20 illustrates the hill-climbing idea. For a point  $\mathbf{x}$ , the density attractor for  $\mathbf{x}$  is initialized to  $\mathbf{x}^0 = \mathbf{x}$ . In the next iteration, the density attractor moves a small step towards the direction indicated by the gradient of the density function, shown by the arrow in the figure, until the point  $\mathbf{x}^*$  where density is stable (the white circle point in the figure), which is a local optimal.

A cluster in DENCLUE is a set of density attractors  $X$  and a set of input objects  $C$  such that each object in  $C$  is assigned to a density attractor in  $X$ , and there exists a path between every pair of density attractors where the density is above  $\xi$ . By using multiple density attractors connected by paths, DENCLUE can find clusters of arbitrary shape.

DENCLUE has several advantages. It can be regarded as a generalization of several well-known clustering methods such as single-linkage approaches and DBSCAN. Moreover, DENCLUE is invariant against noise. The kernel density estimation can effectively reduce the influence of noise by uniformly distributing noise into the input data.

### 8.4.3 Grid-based methods

As analyzed in the previous subsections, computing density for density-based clustering may be costly, particularly on large data sets and data sets of high dimensionality. To tackle the efficiency and scalability challenges, one idea is to partition the data space into cells using a grid. This motivates the grid-based clustering methods.

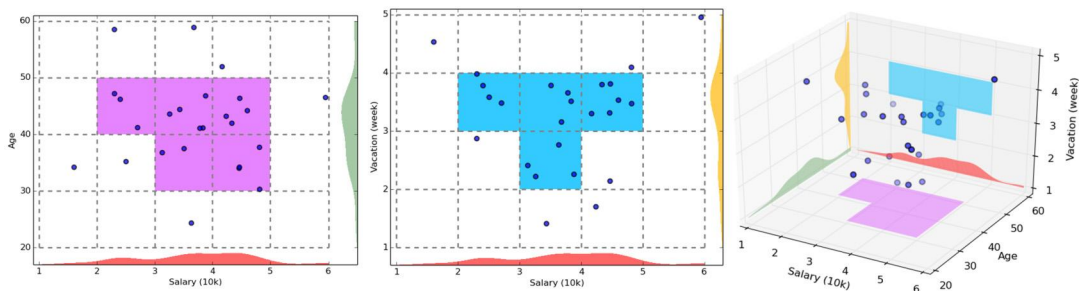
The *grid-based clustering* approach uses a multiresolution grid data structure. It quantizes the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed. The main advantage of the approach is its fast processing time, which is typically independent of the number of data objects, yet dependent on only the number of cells in each dimension in the quantized space.

Typically, grid-based clustering takes three steps.

1. We create a grid structure so that the data space is partitioned into a finite number of cells.
2. For each cell, we calculate the cell density. By a carefully designed method, we may be able to scan the data once and derive the densities for all cells. This step is a key to gain efficiency and scalability.
3. We use the dense cells to assemble clusters and optionally summarize dense cells and the corresponding clusters.

Let us illustrate this using an example. **CLIQUE** (CLustering In QUest) is a simple grid-based method for finding density-based clusters in subspaces. CLIQUE partitions each dimension into nonoverlapping intervals, thereby partitioning the entire data space into cells. It uses a density threshold to identify *dense* cells and *sparse* ones. A cell is dense if the number of objects mapped to it exceeds the density threshold.

The main strategy behind CLIQUE for identifying a candidate search space uses the monotonicity of dense cells with respect to dimensionality. This is based on the *Apriori property* used in frequent pattern and association rule mining (Chapter 4). In the context of clusters in subspaces, the monotonicity says the following. A  $k$ -dimensional cell  $c$  ( $k > 1$ ) can have at least  $l$  points only if every  $(k - 1)$ -dimensional projection of  $c$ , which is a cell in a  $(k - 1)$ -dimensional subspace, has at least  $l$  points. Consider Fig. 8.21, where the data space contains three dimensions: *age*, *salary*, and *vacation*. A 2-D



**FIGURE 8.21**

Dense units found with respect to *age* for the dimensions *salary* and *vacation* are intersected to provide a candidate search space for dense units of higher dimensionality.

cell, say in the subspace formed by *age* and *salary*, contains  $l$  points only if the projection of this cell in every dimension, that is, *age* and *salary*, respectively, contains at least  $l$  points.

CLIQUE performs clustering in three steps. In the first step, CLIQUE partitions the  $d$ -dimensional data space into nonoverlapping rectangular units.

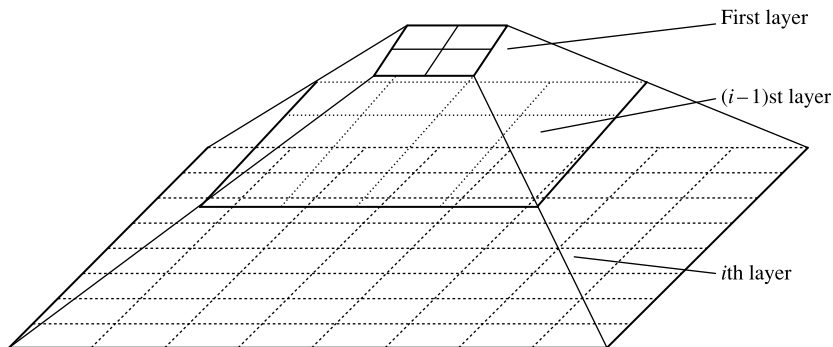
In the second step, CLIQUE identifies the dense units among these. CLIQUE finds dense cells in all of the subspaces. To do so, CLIQUE partitions every dimension into intervals, and identifies intervals containing at least  $l$  points, where  $l$  is the density threshold. CLIQUE then iteratively joins two  $k$ -dimensional dense cells,  $c_1$  and  $c_2$ , in subspaces  $(D_{i_1}, \dots, D_{i_k})$  and  $(D_{j_1}, \dots, D_{j_k})$ , respectively, if  $D_{i_1} = D_{j_1}, \dots, D_{i_{k-1}} = D_{j_{k-1}}$ , and  $c_1$  and  $c_2$  share the same intervals in those dimensions. The join operation generates a new  $(k + 1)$ -dimensional candidate cell  $c$  in space  $(D_{i_1}, \dots, D_{i_{k-1}}, D_{i_k}, D_{j_k})$ . CLIQUE checks whether the number of points in  $c$  passes the density threshold. The iteration terminates when no candidates can be generated or no candidate cells are dense.

In the last step, CLIQUE uses the dense cells in each subspace to assemble clusters, which can be of arbitrary shape. The idea is to apply the Minimum Description Length (MDL) principle (Chapter 7) to use the *maximal regions* to cover connected dense cells, where a maximal region is a hyperrectangle where every cell falling into this region is dense, and the region cannot be extended further in any dimension in the subspace. Finding the best description of a cluster in general is NP-hard. Thus CLIQUE adopts a simple greedy approach. It starts with an arbitrary dense cell, finds a maximal region covering the cell, and then works on the remaining dense cells that have not yet been covered. The greedy method terminates when all dense cells are covered.

“*How effective is CLIQUE?*” CLIQUE automatically finds subspaces of the highest dimensionality such that high-density clusters exist in those subspaces. It is insensitive to the order of input objects and does not presume any canonical data distribution. It scales linearly with the size of the input and has good scalability as the number of dimensions in the data increases. However, obtaining a meaningful clustering is dependent on proper tuning of the grid size (which is a stable structure here) and the density threshold. This can be difficult in practice because the grid size and density threshold are used across all combinations of dimensions in the data set. Thus the accuracy of the clustering results may be degraded at the expense of the method’s simplicity. Moreover, for a given dense region, all projections of the region onto lower-dimensionality subspaces will also be dense. This can result in a large overlap among the reported dense regions. Furthermore, it is difficult to find clusters of rather different densities within different dimensional subspaces.

**STING** is another representative grid-based multiresolution clustering technique. In STING, the spatial area of the input objects is divided into rectangular cells. The space can be divided in a hierarchical and recursive way. Several levels of such rectangular cells correspond to different levels of resolution and form a hierarchical structure: Each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell, such as the mean, maximum, and minimum values, is precomputed and stored as *statistical parameters*. These statistical parameters are useful for query processing and for other data analysis tasks.

Fig. 8.22 shows a hierarchical structure for STING clustering. The statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; and the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum), and the type of *distribution* that the attribute value in the cell follows such as *normal*, *uniform*, *exponential*, or *none* (if the distribution is unknown). Here, the attribute is a selected measure for analysis such as *price* for house objects.



**FIGURE 8.22**

Hierarchical structure for STING clustering.

When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as the  $\chi^2$  test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If the distributions of the lower-level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

“How is this statistical information useful for query answering?” The statistical parameters can be used in a top-down, grid-based manner as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated probability range) reflecting the relevancy of the cell to the given query. The irrelevant cells are removed from further consideration. Processing of the next lower level examines only the remaining relevant cells. This process repeats until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data points that fall into the relevant cells are retrieved and further processed until they meet the query’s requirements.

An interesting property of STING is that it approaches the clustering result of DBSCAN if the granularity approaches 0 (i.e., toward very low-level data). In other words, using the count and cell size information, dense clusters can be identified approximately using STING. Therefore STING can also be regarded as a density-based clustering method.

“What advantages does STING offer over other clustering methods?” STING offers several advantages. First, the grid-based computation is *query-independent* because the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query. Moreover, the grid structure facilitates parallel processing and incremental updating. Last, the efficiency of STING is a major advantage: STING goes through the database once to compute the statistical parameters of the cells and hence the time complexity of generating clusters is  $O(n)$ , where  $n$  is the total number of objects. After generating the hierarchical structure, the query processing time

is  $O(g)$ , where  $g$  is the total number of grid cells at the lowest level, which is usually much smaller than  $n$ .

Because STING uses a multiresolution approach to cluster analysis, the quality of STING clustering depends on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of processing increases substantially; however, if the bottom level of the grid structure is too coarse, it may reduce the quality of cluster analysis. Moreover, STING does not consider the spatial relationship between the children and their neighboring cells for construction of a parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the fast processing time of the technique.

---

## 8.5 Evaluation of clustering

By now you have learned what clustering is and know several popular clustering methods. You may ask, “*When I try out a clustering method on a data set, how can I evaluate whether the clustering results are good?*” In general, *cluster evaluation* assesses the feasibility of clustering analysis on a data set and the quality of the results generated by a clustering method. The major tasks of clustering evaluation include the following:

- *Assessing clustering tendency.* In this task, for a given data set, we assess whether a nonrandom structure exists in the data. Blindly applying a clustering method on a data set will return clusters; however, the clusters mined may be misleading. Clustering analysis on a data set is meaningful only when there is a nonrandom structure in the data.
- *Determining the number of clusters in a data set.* A few algorithms, such as  $k$ -means, require the number of clusters in a data set as the parameter. Moreover, the number of clusters can be regarded as an interesting and important summary statistic of a data set. Therefore it is desirable to estimate this number even before a clustering algorithm is used to derive detailed clusters.
- *Measuring clustering quality.* After applying a clustering method on a data set, we want to assess how good the resulting clusters are. A number of measures can be used. Some methods measure how well the clusters fit the data set, while others measure how well the clusters match the ground truth, if such truth is available. There are also measures that score clusterings and thus can compare two sets of clustering results on the same data set.

In this section, we discuss these three topics one by one.

### 8.5.1 Assessing clustering tendency

Clustering tendency assessment determines whether a given data set has a nonrandom structure, which may lead to meaningful clusters. Consider a data set that does not have any nonrandom structure, such as a set of uniformly distributed points in a data space. Even though a clustering algorithm may return clusters for the data, those clusters are random and thus are not meaningful.

**Example 8.8. Clustering requires nonuniform distribution of data.** Fig. 8.23 shows a data set that is uniformly distributed in 2-D data space. Although a clustering algorithm may still artificially partition



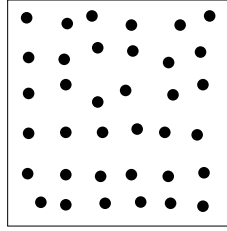


FIGURE 8.23

A data set that is uniformly distributed in the data space.

the points into groups, the groups will unlikely mean anything significant to the application due to the uniform distribution of the data.  $\square$

“How can we assess the clustering tendency of a data set?” Intuitively, we can try to measure the probability that the data set is generated by a uniform data distribution. This can be achieved using statistical tests for spatial randomness. To illustrate this idea, let us look at a simple yet effective statistic called the Hopkins statistic.

The **Hopkins Statistic** is a spatial statistic that tests the spatial randomness of a variable as distributed in a space. Given a data set,  $D$ , which is regarded as a sample of a random variable,  $o$ , we want to determine how far away  $o$  is from being uniformly distributed in the data space. We calculate the Hopkins Statistic as follows:

1. Sample  $n$  points,  $p_1, \dots, p_n$  from the data space. For each point,  $p_i$  ( $1 \leq i \leq n$ ), we find the nearest neighbor in  $D$ , and let  $x_i$  be the distance between  $p_i$  and its nearest neighbor in  $D$ . That is,

$$x_i = \min_{v \in D} \{dist(p_i, v)\}. \quad (8.26)$$

2. Sample  $n$  points,  $q_1, \dots, q_n$  uniformly from  $D$  without replacement. That is, each point in  $D$  has the same probability of being included in this sample, and one point can only be included in the sample at most once. For each  $q_i$  ( $1 \leq i \leq n$ ), we find the nearest neighbor of  $q_i$  in  $D - \{q_i\}$ , and let  $y_i$  be the distance between  $q_i$  and its nearest neighbor in  $D - \{q_i\}$ . That is,

$$y_i = \min_{v \in D, v \neq q_i} \{dist(q_i, v)\}. \quad (8.27)$$

3. Calculate the Hopkins statistic,  $H$ , as

$$H = \frac{\sum_{i=1}^n x_i^d}{\sum_{i=1}^n x_i^d + \sum_{i=1}^n y_i^d}, \quad (8.28)$$

where  $d$  is the dimensionality of the data set  $D$ .

“What does the Hopkins statistic tell us about how likely data set  $D$  follows a uniform distribution in the data space?” If  $D$  is uniformly distributed, then  $\sum_{i=1}^n y_i^d$  and  $\sum_{i=1}^n x_i^d$  are close to each other,

and thus  $H$  tends to be about 0.5. However, if  $D$  is highly skewed, then the points in  $D$  are closer to their nearest neighbors than the random points  $p_1, \dots, p_n$  are, and thus  $\sum_{i=1}^n x_i^d$  shall be substantially larger than  $\sum_{i=1}^n y_i^d$  in expectation, and  $H$  tends to be close to 1.

**Example 8.9. Hopkins statistic.** Consider a 1-D data set  $D = \{0.9, 1, 1.3, 1.4, 1.5, 1.8, 2, 2.1, 4.1, 7, 7.4, 7.5, 7.7, 7.8, 7.9, 8.1\}$  in the data space  $[0, 10]$ . We draw a sample of four points from  $D$  without replacement, say, 1.3, 1.8, 7.5, and 7.9. We also draw a sample of four points uniformly from the data space  $[0, 10]$ , say, 1.9, 4, 6, 8. Then, the Hopkins statistic can be calculated as

$$\begin{aligned} H &= \frac{|1.9 - 2| + |4 - 4.1| + |6 - 7| + |8 - 8.1|}{(|1.9 - 2| + |4 - 4.1| + |6 - 7| + |8 - 8.1|) + (|1.3 - 1.4| + |1.8 - 2| + |7.5 - 7.4| + |7.9 - 7.8|)} \\ &= \frac{1.3}{1.3 + 0.5} = \frac{1.3}{1.8} = 0.72. \end{aligned}$$

Since the Hopkins statistic is substantially larger than 0.5 and is close to 1, the data set  $D$  has a strong clustering tendency. Indeed, there are two clusters, one around 1.5 and the other one around 7.8.  $\square$

In addition to Hopkins statistic, there are some other methods, such as spatial histogram and distance distribution, comparing statistics between a data set under clustering tendency analysis and the corresponding uniform distribution. For example, distance distribution compares the distribution of pairwise distance in the target data set and that in a random uniform sample from the data space.

## 8.5.2 Determining the number of clusters

Determining the “right” number of clusters in a data set is important, not only because some clustering algorithms like  $k$ -means require such a parameter, but also because the appropriate number of clusters controls the proper granularity of cluster analysis. It can be regarded as finding a good balance between *compressibility* and *accuracy* in cluster analysis. Consider two extreme cases. What if you were to treat the entire data set as a cluster? This would maximize the compression of the data, but such a cluster analysis has no value. In contrast, treating each object in a data set as a cluster gives the finest clustering resolution (i.e., most accurate due to the zero distance between an object and the corresponding cluster center). In some methods like  $k$ -means, this even achieves the minimum cost. However, having one object per cluster does not enable any data summarization.

Determining the number of clusters is far from easy, often because the “right” number is ambiguous. Figuring out the right number of clusters often depends on the distribution’s shape and scale in the data set, as well as the clustering resolution required by the user. There are many possible ways to estimate the number of clusters.

For example, a simple method is to set the number of clusters to about  $\sqrt{\frac{n}{2}}$  for a data set of  $n$  points. In expectation, each cluster has  $\sqrt{2n}$  points. Section 8.2.2 introduces the Calinski-Harabasz index, which estimates the number of clusters for  $k$ -means.

Let us look at two more alternative methods.

The **elbow method** is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster. This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other. However, the marginal

effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction. Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Technically, given a number,  $k > 0$ , we can form  $k$  clusters on the data set in question using a clustering algorithm like  $k$ -means, and calculate the sum of within-cluster variances,  $var(k)$ . We can then plot the curve of  $var$  with respect to  $k$ . The first (or most significant) turning point of the curve suggests the “right” number.

More advanced methods can determine the number of clusters using information criteria or information theoretic approaches. Please refer to the bibliographic notes for further information (Section 8.8).

The “right” number of clusters in a data set can also be determined by **cross-validation**, a technique often used in classification (Chapter 6). First, we divide the given data set,  $D$ , into  $m$  parts. Next, we use  $m - 1$  parts to build a clustering model, and use the remaining part to test the quality of the clustering. For example, for each point in the test set, we can find the closest centroid. Consequently, we can use the sum of the squared distances between all points in the test set and the closest centroids to measure how well the clustering model fits the test set. For any integer  $k > 0$ , we repeat this process  $m$  times to derive clusterings of  $k$  clusters, using each part in turn as the test set. The average of the quality measure is taken as the overall quality measure. We can then compare the overall quality measure with respect to different values of  $k$  and find the number of clusters that best fits the data.

### 8.5.3 Measuring clustering quality: extrinsic methods

Suppose you have assessed the clustering tendency of a given data set. You may have also tried to predetermine the number of clusters in the set. You can now apply one or multiple clustering methods to obtain clusterings of the data set. “*How good is the clustering generated by a method, and how can we compare the clusterings generated by different methods?*”

#### ***Extrinsic vs. intrinsic methods***

We have a few methods to choose from for measuring the quality of a clustering. In general, these methods can be categorized into two groups according to whether ground truth is available. Here, *ground truth* is the ideal clustering that is often built using human experts.

If ground truth is available, it can be used by the **extrinsic methods**, which compare the clustering against the ground truth and measure. If the ground truth is unavailable, we can use the **intrinsic methods**, which evaluate the goodness of a clustering by considering how well the clusters are separated. Ground truth can be considered as supervision in the form of “cluster labels.” Hence, extrinsic methods are also known as *supervised methods*, whereas intrinsic methods are *unsupervised methods*.

In this section, we focus on extrinsic methods. We will discuss intrinsic methods in the next section.

#### ***Desiderata of extrinsic methods***

When the ground truth is available, we can compare it with a clustering to assess the quality of the clustering. Thus the core task in extrinsic methods is to assign a score,  $Q(\mathcal{C}, \mathcal{C}_g)$ , to a clustering,  $\mathcal{C}$ , given the ground truth,  $\mathcal{C}_g$ . Whether an extrinsic method is effective largely depends on the measure,  $Q$ , it uses.

In general, a measure  $Q$  on clustering quality is effective if it satisfies the following four essential criteria:

- **Cluster homogeneity.** This requires that the purer the clusters in a clustering are, the better the clustering. Suppose that the ground truth says that the objects in a data set,  $D = \{a, b, c, d, e, f, g, h\}$ , can belong to three categories. Objects  $a$  and  $b$  are in category  $L_1$ , objects  $c$  and  $d$  belong to category  $L_2$ , and the others are in category  $L_3$ . Consider clustering,  $\mathcal{C}_1 = \{\{a, b, c, d\}, \{e, f, g, h\}\}$ , wherein a cluster  $\{a, b, c, d\} \in \mathcal{C}_1$  contains objects from two categories,  $L_1$  and  $L_2$ . Also consider clustering  $\mathcal{C}_2 = \{\{a, b\}, \{c, d\}, \{e, f, g, h\}\}$ , which is identical to  $\mathcal{C}_1$  except that  $\mathcal{C}_2$  is split into two clusters containing the objects in  $L_1$  and  $L_2$ , respectively. A clustering quality measure,  $Q$ , respecting cluster homogeneity should give a higher score to  $\mathcal{C}_2$  than  $\mathcal{C}_1$ , that is,  $Q(\mathcal{C}_2, \mathcal{C}_g) > Q(\mathcal{C}_1, \mathcal{C}_g)$ .
- **Cluster completeness.** This is the counterpart of cluster homogeneity. Cluster completeness requires that for a clustering, if any two objects belong to the same category according to the ground truth, then they should be assigned to the same cluster. Cluster completeness requires that a clustering should assign objects belonging to the same category (according to the ground truth) to the same cluster. Continue our previous example. Suppose clustering  $\mathcal{C}_3 = \{\{a, b\}, \{c, d\}, \{e, f\}, \{g, h\}\}$ .  $\mathcal{C}_3$  and  $\mathcal{C}_2$  are identical except that  $\mathcal{C}_3$  split the objects in category  $L_3$  into two clusters. Then, a clustering quality measure,  $Q$ , respecting cluster completeness should give a higher score to  $\mathcal{C}_2$ , that is,  $Q(\mathcal{C}_2, \mathcal{C}_g) > Q(\mathcal{C}_1, \mathcal{C}_g)$ .
- **Rag bag.** In many practical scenarios, there is often a “rag bag” category containing objects that cannot be merged with other objects. Such a category is often called “miscellaneous,” “other,” and so on. The rag bag criterion states that putting a heterogeneous object into a pure cluster should be penalized more than putting it into a rag bag. Consider a clustering  $\mathcal{C}_1$  and a cluster  $C \in \mathcal{C}_1$  such that all objects in  $C$  except for one, denoted by  $o$ , belong to the same category according to the ground truth. Consider a clustering  $\mathcal{C}_2$  identical to  $\mathcal{C}_1$  except that  $o$  is assigned to a cluster  $C' \neq C$  in  $\mathcal{C}_2$  such that  $C'$  contains objects from various categories according to ground truth, and thus is noisy. In other words,  $C'$  in  $\mathcal{C}_2$  is a rag bag. Then, a clustering quality measure  $Q$  respecting the rag bag criterion should give a higher score to  $\mathcal{C}_2$ , that is,  $Q(\mathcal{C}_2, \mathcal{C}_g) > Q(\mathcal{C}_1, \mathcal{C}_g)$ .
- **Small cluster preservation.** If a small category is split into small pieces in a clustering, those small pieces may likely become noise and thus the small category cannot be discovered from the clustering. The small cluster preservation criterion states that splitting a small category into pieces is more harmful than splitting a large category into pieces. Consider an extreme case. Let  $D$  be a data set of  $n + 2$  objects such that, according to ground truth,  $n$  objects, denoted by  $o_1, \dots, o_n$ , belong to one category and the other two objects, denoted by  $o_{n+1}, o_{n+2}$ , belong to another category. Suppose clustering  $\mathcal{C}_1$  has three clusters,  $\mathcal{C}_1^1 = \{o_1, \dots, o_n\}$ ,  $\mathcal{C}_1^2 = \{o_{n+1}\}$ , and  $\mathcal{C}_1^3 = \{o_{n+2}\}$ . Let clustering  $\mathcal{C}_2$  have three clusters, too, namely  $\mathcal{C}_2^1 = \{o_1, \dots, o_{n-1}\}$ ,  $\mathcal{C}_2^2 = \{o_n\}$ , and  $\mathcal{C}_2^3 = \{o_{n+1}, o_{n+2}\}$ . In other words,  $\mathcal{C}_1$  splits the small category and  $\mathcal{C}_2$  splits the big category. A clustering quality measure  $Q$  preserving small clusters should give a higher score to  $\mathcal{C}_2$ , that is,  $Q(\mathcal{C}_2, \mathcal{C}_g) > Q(\mathcal{C}_1, \mathcal{C}_g)$ .

### Categories of extrinsic methods

The ground truth may be used in different ways to evaluate clustering quality, which lead to different extrinsic methods. In general, the extrinsic methods can be categorized according to how the ground truth is used as follows.

- **The matching-based methods** examine how well the clustering results match the ground truth in partitioning the objects in the data set. For example, the purity methods assess how a cluster matches only those objects in one group in the ground truth.
- **The information theory-based methods** compare the distribution of the clustering results and that of the ground truth. Entropy or other measures in information theory are often employed to quantify the comparison. For example, we can measure the conditional entropy between the clustering results and the ground truth to measure whether there exists dependency between the information of the clustering results and the ground truth. The higher the dependency, the better the clustering results.
- **The pairwise comparison-based methods** treat each group in the ground truth as a class and then check the pairwise consistency of the objects in the clustering results. The clustering results are good if more pairs of objects of the same class are put into the same cluster, less pairs of objects of different classes are put into the same cluster, and less pairs of objects of the same class are put into different clusters.

Next, let us use some examples to illustrate the above categories of extrinsic methods.

### Matching-based methods

The matching-based methods compare clusters in the clustering results and the groups in the ground truth. Let us use an example to explain the ideas.

Suppose a clustering method partitions a set of objects  $D = \{o_1, \dots, o_n\}$  into clusters  $\mathcal{C} = \{C_1, \dots, C_m\}$ . The ground truth  $\mathcal{G}$  also partitions the same set of objects into groups  $\mathcal{G} = \{G_1, \dots, G_l\}$ . Let  $C(o_x)$  and  $G(o_x)$  ( $1 \leq x \leq n$ ) be the cluster-id and the group-id of object  $o_x$  in the clustering results and the ground truth, respectively.

For a cluster  $C_i$  ( $1 \leq i \leq m$ ), how well  $C_i$  matches group  $G_j$  in the ground truth can be measured by  $|C_i \cap G_j|$ , the larger the better.  $\frac{|C_i \cap G_j|}{|C_i|}$  can be regarded as the purity of cluster  $C_i$ , where  $G_j$  matching  $C_i$  maximizes  $|C_i \cap G_j|$ . The purity of the whole clustering results can be calculated as the weighted sum of the purity of the clusters. That is,

$$purity = \sum_{i=1}^m \frac{|C_i|}{n} \max_{j=1}^l \left\{ \frac{|C_i \cap G_j|}{|C_i|} \right\} = \frac{1}{n} \sum_{i=1}^m \max_{j=1}^l \{|C_i \cap G_j|\}. \quad (8.29)$$

The higher the purity, the purer are the clusters, that is, the more objects in each cluster belong to the same group in the ground truth. When the purity is 1, each cluster either matches a group perfectly or is a subset of a group. In other words, no two objects belong to two groups are mixed in one cluster. However, it is possible that multiple clusters partition a group in the ground truth.

**Example 8.10. Purity.** Consider the set of objects  $D = \{a, b, c, d, e, f, g, h, i, j, k\}$ . The clustering ground truth and two clusterings  $\mathcal{C}_1$  and  $\mathcal{C}_2$  output by two methods are shown in Table 8.1.

The purity of clustering  $\mathcal{C}_1$  is calculated by  $\frac{1}{11} \times (4 + 2 + 4 + 1) = \frac{11}{11} = 1$  and that of clustering  $\mathcal{C}_2$  is  $\frac{1}{11} (2 + 3 + 1) = \frac{6}{11}$ . In terms of purity,  $\mathcal{C}_1$  is better than  $\mathcal{C}_2$ . Please note that, although  $\mathcal{C}_1$  has purity 1, it splits  $G_1$  in the ground truth into two clusters,  $C_1$  and  $C_2$ .  $\square$

There are some other matching based methods further refine the measurement of matching quality, such as maximum matching and using F-measure.

**Table 8.1** A set of objects, the clustering ground truth, and two clusterings.

| Object                     | <i>a</i> | <i>b</i> | <i>c</i> | <i>d</i> | <i>e</i> | <i>f</i> | <i>g</i> | <i>h</i> | <i>i</i> | <i>j</i> | <i>k</i> |
|----------------------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Ground truth $\mathcal{G}$ | $G_1$    | $G_1$    | $G_1$    | $G_1$    | $G_1$    | $G_1$    | $G_2$    | $G_2$    | $G_2$    | $G_2$    | $G_3$    |
| Clustering $\mathcal{C}_1$ | $C_1$    | $C_1$    | $C_1$    | $C_1$    | $C_2$    | $C_2$    | $C_3$    | $C_3$    | $C_3$    | $C_3$    | $C_4$    |
| Clustering $\mathcal{C}_2$ | $C_1$    | $C_1$    | $C_2$    | $C_2$    | $C_2$    | $C_3$    | $C_1$    | $C_2$    | $C_2$    | $C_1$    | $C_3$    |

**Information theory–based methods**

A clustering assigns objects to clusters and thus can be regarded as a compression of the information carried by the objects. In other words, a clustering can be regarded as a compressed representation of a given set of objects. Therefore we can use information theory to compare a clustering and the ground truth as representations. This is the general idea behind the information theory–based methods.

For example, we can measure the amount of information needed to describe the ground truth given the distribution of a clustering output by a method. Better the clustering results approach the ground truth, less amount information is needed. This leads to a natural approach using conditional entropy.

Concretely, according to information theory, the entropy of a clustering  $\mathcal{C}$  is

$$H(\mathcal{C}) = - \sum_{i=1}^m \frac{|C_i|}{n} \log \frac{|C_i|}{n},$$

and the entropy of the ground truth is

$$H(\mathcal{G}) = - \sum_{i=1}^l \frac{|G_i|}{n} \log \frac{|G_i|}{n}.$$

The conditional entropy of  $\mathcal{G}$  given cluster  $C_i$  is

$$H(\mathcal{G}|C_i) = - \sum_{j=1}^l \frac{|C_i \cap G_j|}{|C_i|} \log \frac{|C_i \cap G_j|}{|C_i|}.$$

The conditional entropy of  $\mathcal{G}$  given clustering  $\mathcal{C}$  is

$$H(\mathcal{G}|\mathcal{C}) = \sum_{i=1}^m \frac{|C_i|}{n} H(\mathcal{G}|C_i) = - \sum_{i=1}^m \sum_{j=1}^l \frac{|C_i \cap G_j|}{n} \log \frac{|C_i \cap G_j|}{|C_i|}.$$

In addition to the simple conditional entropy, more sophisticated information theory–based measures may be used, such as normalized mutual information and variation of information.

Taking the case in Table 8.1 as an example, we can calculate

$$H(\mathcal{G}|\mathcal{C}_1) = - \left( \frac{4}{11} \log \frac{4}{4} + \frac{2}{11} \log \frac{2}{2} + \frac{4}{11} \log \frac{4}{4} + \frac{1}{11} \log \frac{1}{1} \right) = 0$$

and

$$\begin{aligned} H(\mathcal{G}|\mathcal{C}_2) &= -\left(\frac{2}{11} \log \frac{2}{4} + \frac{2}{11} \log \frac{2}{4} + \frac{3}{11} \log \frac{3}{5} + \frac{2}{11} \log \frac{2}{5} + \frac{1}{11} \log \frac{1}{2} + \frac{1}{11} \log \frac{1}{2}\right) \\ &= 0.297. \end{aligned}$$

Clustering  $\mathcal{C}_1$  has better quality than  $\mathcal{C}_2$  in terms of conditional entropy. Again, although  $H(\mathcal{G}|\mathcal{C}_1) = 0$ , conditional entropy cannot detect the issue that  $\mathcal{C}_1$  splits the objects in  $G_1$  into two clusters.

### Pairwise comparison-based methods

The pairwise comparison-based methods treat each group in the ground truth as a class. For each pair of objects  $o_i, o_j \in D$  ( $1 \leq i, j \leq n, i \neq j$ ), if they are assigned to the same cluster/group, the assignment is regarded as positive, and otherwise, negative. Then, depending on assignments of  $o_i$  and  $o_j$  into clusters  $C(o_i), C(o_j), G(o_i)$ , and  $G(o_j)$ , we have four possible cases.

|                      | $C(o_i) = C(o_j)$ | $C(o_i) \neq C(o_j)$ |
|----------------------|-------------------|----------------------|
| $G(o_i) = G(o_j)$    | true positive     | false negative       |
| $G(o_i) \neq G(o_j)$ | false positive    | true negative        |

Using the statistics on pairwise comparison, we can assess the quality of the clustering results approaching the ground truth. For example, we can use the Jaccard coefficient, which is defined as

$$J = \frac{\text{true positive}}{\text{true positive} + \text{false negative} + \text{false positive}}.$$

Many other measures can be built based on the pairwise comparison statistics, such as Rand statistic, fowlkes-Mallows measure, BCubed precision, and recall. The pairwise comparison results can be further used to conduct correlation analysis. For example, we can form a binary matrix  $\mathbf{G}$  according to the ground truth, where element  $v_{ij} = 1$  if  $G(o_i) = G(o_j)$ , and otherwise 0. A binary matrix  $\mathbf{C}$  can also be constructed in a similar way based on a clustering  $\mathcal{C}$ . We can analyze the element-wise correlation between the two matrixes and use the correlation to measure the quality of the clustering results. Clearly, the more correlated the two matrixes, the better the clustering results.

## 8.5.4 Intrinsic methods

When the ground truth of a data set is not available, we have to use an intrinsic method to assess the clustering quality. Unable to reference any external supervision information, the intrinsic methods have to come back to the fundamental intuition in clustering analysis, that is, examining how compact clusters are and how well clusters are separated. Many intrinsic methods take the advantage of a similarity or distance measure between objects in the data set.

For example, the **Dunn index** measures the compactness of clusters by the maximum distance between two points that belong to the same cluster, that is,  $\Delta = \max_{C(o_i)=C(o_j)}\{d(o_i, o_j)\}$ . It measures the degree of separation among different clusters by the minimum distance between two points that belong to different clusters, that is  $\delta = \min_{C(o_i) \neq C(o_j)}\{d(o_i, o_j)\}$ . Then, the Dunn index is simply the ration  $DI = \frac{\delta}{\Delta}$ . The larger the ratio, the farther away the clusters are separated comparing to the compactness of the clusters.

The Dunn index uses the extreme distances to measure the cluster compactness and intercluster separation. The measures  $\delta$  and  $\Delta$  may be affected by the outliers. Many methods consider the average situations. The **silhouette coefficient** is such a measure. For a data set,  $D$ , of  $n$  objects, suppose  $D$  is partitioned into  $k$  clusters,  $C_1, \dots, C_k$ . For each object  $\mathbf{o} \in D$ , we calculate  $a(\mathbf{o})$  as the average distance between  $\mathbf{o}$  and all other objects in the cluster to which  $\mathbf{o}$  belongs. Similarly,  $b(\mathbf{o})$  is the minimum average distance from  $\mathbf{o}$  to all clusters to which  $\mathbf{o}$  does not belong. Formally, suppose  $\mathbf{o} \in C_i$  ( $1 \leq i \leq k$ ). Then,

$$a(\mathbf{o}) = \frac{\sum_{\mathbf{o}' \in C_i, \mathbf{o}' \neq \mathbf{o}} \text{dist}(\mathbf{o}, \mathbf{o}')}{|C_i| - 1} \quad (8.30)$$

and

$$b(\mathbf{o}) = \min_{C_j: 1 \leq j \leq k, j \neq i} \left\{ \frac{\sum_{\mathbf{o}' \in C_j} \text{dist}(\mathbf{o}, \mathbf{o}')}{|C_j|} \right\}. \quad (8.31)$$

The **silhouette coefficient** of  $\mathbf{o}$  is then defined as

$$s(\mathbf{o}) = \frac{b(\mathbf{o}) - a(\mathbf{o})}{\max\{a(\mathbf{o}), b(\mathbf{o})\}}. \quad (8.32)$$

The value of the silhouette coefficient is between  $-1$  and  $1$ . The value of  $a(\mathbf{o})$  reflects the compactness of the cluster to which  $\mathbf{o}$  belongs. The smaller the value, the more compact the cluster. The value of  $b(\mathbf{o})$  captures the degree to which  $\mathbf{o}$  is separated from other clusters. The larger  $b(\mathbf{o})$  is, the more separated  $\mathbf{o}$  is from other clusters. Therefore when the silhouette coefficient value of  $\mathbf{o}$  approaches  $1$ , the cluster containing  $\mathbf{o}$  is compact and  $\mathbf{o}$  is far away from other clusters, which is the preferable case. However, when the silhouette coefficient value is negative (i.e.,  $b(\mathbf{o}) < a(\mathbf{o})$ ), this means that, in expectation,  $\mathbf{o}$  is closer to the objects in another cluster than to the objects in the same cluster as  $\mathbf{o}$ . In many cases, this is a bad situation and should be avoided.

To measure a cluster's fitness within a clustering, we can compute the average silhouette coefficient value of all objects in the cluster. To measure the quality of a clustering, we can use the average silhouette coefficient value of all objects in the data set. The silhouette coefficient and other intrinsic measures can also be used in the elbow method to heuristically derive the number of clusters in a data set by replacing the sum of within-cluster variances.

---

## 8.6 Summary

- A **cluster** is a collection of data objects that are *similar* to one another within the same cluster and are *dissimilar* to the objects in other clusters. The process of grouping a set of physical or abstract objects into classes of *similar* objects is called **clustering**.
- Cluster analysis has extensive **applications**, including business intelligence, image pattern recognition, Web search, biology, and security. Cluster analysis can be used as a standalone data mining tool to gain insight into the data distribution or as a preprocessing step for other data mining algorithms operating on the detected clusters.
- Clustering is a dynamic field of research in data mining. It is related to **unsupervised learning** in machine learning.



- Clustering is a challenging field. Typical **requirements** of it include scalability, the ability to deal with different types of data and attributes, the discovery of clusters in arbitrary shape, minimal requirements for domain knowledge to determine input parameters, the ability to deal with noisy data, incremental clustering and insensitivity to input order, the capability of clustering high-dimensionality data, constraint-based clustering, and interpretability and usability.
- Many clustering algorithms have been developed. These can be categorized from several **orthogonal aspects** such as those regarding partitioning criteria, separation of clusters, similarity measures used, and clustering space. This chapter discusses major fundamental clustering methods of the following categories: *partitioning methods*, *hierarchical methods*, and *density-based and grid-based methods*. Some algorithms may belong to more than one category.
- A **partitioning method** first creates an initial set of  $k$  partitions, where parameter  $k$  is the number of partitions to construct. It then uses an *iterative relocation technique* that attempts to improve the partitioning by moving objects from one group to another. Typical partitioning methods include  $k$ -means,  $k$ -medoids, and  $k$ -modes.
- The **centroid-based partitioning technique** uses the **within-cluster variation** to measure the quality of clusters, which is the sum of squared error between all objects in a cluster and the centroid of the cluster. Minimizing the within-cluster variation is computationally challenging and thus some greedy approaches are often used. The  **$k$ -means method** uses the mean value of the points within a cluster as the centroid. It randomly selects  $k$  objects as the initial centroids of the clusters and then iteratively conducts object assignment and mean update steps until the assignment becomes stable or a certain number of iterations are reached.
- As a variation of  $k$ -means to overcome the effect of outliers, the  **$k$ -medoids method** uses actual objects to represent clusters. While  $k$ -medoids is more robust against noise and outliers, it incurs higher computational cost in each iteration. As another variation of  $k$ -means, the  **$k$ -modes method** uses modes to measure the similarity on nominal data. We can also use kernel functions, such as the Gaussian radial basis function, in  $k$ -means to find clusters that are concave and not linearly separable.
- A **hierarchical method** creates a hierarchical decomposition of the given set of data objects. The method can be classified as being either *agglomerative (bottom-up)* or *divisive (top-down)*, based on how the hierarchical decomposition is formed. **Linkage measures** can be used to assess the distance between clusters in hierarchical clustering. Some widely used measures include *minimum distance (single-linkage)*, *maximum distance (complete-linkage)*, *mean distance*, and *average distance*. The **Lance-Williams algorithm** generalizes different measures and the agglomerative hierarchical clustering framework. The **minimum spanning tree based approach** is a representative method for divisive hierarchical clustering. Hierarchical clustering results can be represented using a **dendrogram**.
- **BIRCH** is a method combining hierarchical clustering and other clustering methods. In BIRCH, clusters are represented using **clustering features** (CF for short) and a hierarchical clustering is represented by a **CF-tree**.
- To overcome some of the drawbacks of hierarchical clustering methods, **probabilistic hierarchical clustering** uses probabilistic models to measure distances between clusters. It shares the same framework and thus has the same efficiency as agglomerative hierarchical clustering methods.
- A **density-based method** clusters objects based on the notion of density. It grows clusters either according to the density of neighborhood objects (e.g., in DBSCAN) or according to a density

function (e.g., in DENCLUE). OPTICS is a density-based method that generates an augmented ordering of the data's clustering structure.

- A **grid-based method** first quantizes the object space into a finite number of cells that form a grid structure, and then performs clustering on the grid structure. STING is a typical example of a grid-based method based on statistical information stored in grid cells. CLIQUE is a grid-based and subspace clustering algorithm.
- **Clustering evaluation** assesses the feasibility of clustering analysis on a data set and the quality of the results generated by a clustering method. The major tasks include assessing clustering tendency, determining the number of clusters, and measuring clustering quality.
- Some statistics, such as **Hopkins statistic**, can be used to assess clustering tendency. In addition to the Calinski-Harabasz index, the **elbow method**, and the **cross-validation** technique can be used to decide the number of clusters in a data set. Depending on whether the ground truth is available, the methods measuring clustering quality can be divided into **extrinsic methods** and **intrinsic methods**. The extrinsic methods try to address the desiderata of *cluster homogeneity*, *cluster completeness*, *rag bag*, and *small cluster preservation*. The extrinsic methods can be divided into the **matching-based methods** (e.g., *purity*), the **information theory-based methods** (e.g., *entropy*), and the **pairwise comparison-based methods** (e.g., using Jaccard coefficient). Examples of the intrinsic methods are the **Dunn index** and the **silhouette coefficient**.

## 8.7 Exercises

- 8.1. Briefly describe and give examples of each of the following approaches to clustering: *partitioning* methods, *hierarchical* methods, *density-based and grid-based* methods, and *bi-clustering* methods.
- 8.2. Suppose that the data mining task is to cluster points (with  $(x, y)$  representing location) into three clusters, where the points are

$$A_1(2, 10), A_2(2, 5), A_3(8, 4), B_1(5, 8), B_2(7, 5), B_3(6, 4), C_1(1, 2), C_2(4, 9).$$

The distance function is Euclidean distance. Suppose initially we assign  $A_1$ ,  $B_1$ , and  $C_1$  as the center of each cluster, respectively. Use the *k-means* algorithm to show *only*

- a. The three cluster centers after the first round of execution.
  - b. The final three clusters.
- 8.3. Use an example to show why the *k-means* algorithm may not find the global optimum, that is, optimizing the within-cluster variation.
  - 8.4. For the *k-means* algorithm, it is interesting to note that by choosing the initial cluster centers carefully, we may be able to not only speed up the algorithm's convergence, but also guarantee the quality of the final clustering. The **k-means++** algorithm is a variant of *k-means*, which chooses the initial centers as follows. First, it selects one center uniformly at random from the objects in the data set. Iteratively, for each object  $p$  other than the chosen center, it chooses an object as the new center. This object is chosen at random with probability proportional to  $dist(p)^2$ , where  $dist(p)$  is the distance from  $p$  to the closest center that has already been chosen. The iteration continues until  $k$  centers are selected.

Explain why this method will not only speed up the convergence of the  $k$ -means algorithm, but also guarantee the quality of the final clustering results.

- 8.5. Provide the pseudocode of the object reassignment step of the PAM algorithm.
- 8.6. Both  $k$ -means and  $k$ -medoids algorithms can perform effective clustering.
  - a. Illustrate the strength and weakness of  $k$ -means in comparison with  $k$ -medoids.
  - b. Illustrate the strength and weakness of these schemes in comparison with a hierarchical clustering scheme.
- 8.7. Show that the single-linkage method is equivalent to taking  $\alpha_i = \alpha_j = 0.5$ ,  $\beta = 0$ , and  $\gamma = -0.5$  in the Lance-Williams formula; the complete-linkage method is equivalent to  $\alpha_i = \alpha_j = 0.5$ ,  $\beta = 0$ , and  $\gamma = 0.5$ ; and the Ward's criterion is equivalent to  $\alpha_i = \frac{n_i+n_k}{n_i+n_j+n_k}$ ,  $\alpha_j = \frac{n_j+n_k}{n_i+n_j+n_k}$ ,  $\beta = -\frac{n_k}{n_i+n_j+n_k}$ , and  $\gamma = 0$ .
- 8.8. Prove that in DBSCAN\*, the density-connectedness is an equivalence relation.
- 8.9. Prove that in DBSCAN\*, for a fixed *MinPts* value and two neighborhood thresholds,  $\epsilon_1 < \epsilon_2$ , a cluster  $C$  with respect to  $\epsilon_1$  and *MinPts* must be a subset of a cluster  $C'$  with respect to  $\epsilon_2$  and *MinPts*.
- 8.10. Provide the pseudocode of the OPTICS algorithm.
- 8.11. Why is it that BIRCH encounters difficulties in finding clusters of arbitrary shape but OPTICS does not? Propose modifications to BIRCH to help it find clusters of arbitrary shape.
- 8.12. Provide the pseudocode of the step in CLIQUE that finds dense cells in all subspaces.
- 8.13. Present conditions under which density-based clustering is more suitable than partitioning-based clustering and hierarchical clustering. Give application examples to support your argument.
- 8.14. Give an example of how specific clustering methods can be *integrated*, for example, where one clustering algorithm is used as a preprocessing step for another. In addition, provide reasoning as to why the integration of two methods may sometimes lead to improved clustering quality and efficiency.
- 8.15. Clustering is recognized as an important data mining task with broad applications. Give one application example for each of the following cases:
  - a. An application that uses clustering as a major data mining function.
  - b. An application that uses clustering as a preprocessing tool for data preparation for other data mining tasks.
- 8.16. Data cubes and multidimensional databases contain nominal, ordinal, and numeric data in hierarchical or aggregate forms. Based on what you have learned about the clustering methods, design a clustering method that finds clusters in large data cubes effectively and efficiently.
- 8.17. Describe each of the following clustering algorithms in terms of the following criteria: (1) shapes of clusters that can be determined; (2) input parameters that must be specified; and (3) limitations.
  - a.  $k$ -means
  - b.  $k$ -medoids
  - c. BIRCH
  - d. DBSCAN\*
- 8.18. Human eyes are fast and effective at judging the quality of clustering methods for 2-D data. Can you design a data visualization method that may help humans visualize data clusters and judge the clustering quality for 3-D data? What about for even higher-dimensional data?

- 8.19.** Discuss how well purity, entropy, and the method using Jaccard coefficient satisfy the four essential requirements for extrinsic clustering evaluation methods.

---

## 8.8 Bibliographic notes

Clustering has been extensively studied for over 40 years and across many disciplines due to its broad applications. Most books on pattern classification and machine learning contain chapters on cluster analysis or unsupervised learning. Several textbooks are dedicated to the methods of cluster analysis, including Hartigan [Har75]; Jain and Dubes [JD88]; Kaufman and Rousseeuw [KR90]; and Arabie, Hubert, and De Sorte [AHS96]. There are also many survey articles on different aspects of clustering methods. Recent ones include Jain, Murty, and Flynn [JMF99]; Parsons, Haque, and Liu [PHL04]; Xu and Wunsch [XW05]; Jain [Jai10]; Greenlaw and Kantabutra [GK13]; Xu and Tian [XT15]; and Berkhin [Ber06].

For partitioning methods, the  $k$ -means algorithm was first introduced by Lloyd [Llo57], and then by MacQueen [Mac67]. Arthur and Vassilvitskii [AV07] presented the  $k$ -means++ algorithm. A filtering algorithm, which uses a spatial hierarchical data index to speed up the computation of cluster means, is given in Kanungo et al. [KMN<sup>+</sup>02].

The  $k$ -medoids algorithms of PAM and CLARA were proposed by Kaufman and Rousseeuw [KR90]. The  $k$ -modes (for clustering nominal data) and  $k$ -prototypes (for clustering hybrid data) algorithms were proposed by Huang [Hua98]. The  $k$ -modes clustering algorithm was also proposed independently by Chaturvedi, Green, and Carroll [CGC94,CGC01]. The CLARANS algorithm was proposed by Ng and Han [NH94]. Ester, Kriegel, and Xu [EKX95] proposed techniques for further improvement of the performance of CLARANS using efficient spatial access methods such as  $R^*$ -tree and focusing techniques. A  $k$ -means-based scalable clustering algorithm was proposed by Bradley, Fayyad, and Reina [BFR98]. The kernel  $k$ -means method was developed by Dhillon, Guan, and Kulis [DGK04].

An early survey of agglomerative hierarchical clustering algorithms was conducted by Day and Edelsbrunner [DE84]. Murtagh and Contreras [MC12] provided a more recent survey on hierarchical clustering. Zhao, Karypis, and Fayyad [ZKF05] surveyed the hierarchical clustering algorithms for document databases. Agglomerative hierarchical clustering, such as AGNES, and divisive hierarchical clustering, such as DIANA, were introduced by Kaufman and Rousseeuw [KR90]. Rohlf [Roh73] developed the essential idea of hierarchical clustering using the minimum spanning tree. An interesting direction for improving the clustering quality of hierarchical clustering methods is to integrate hierarchical clustering with distance-based iterative relocation or other nonhierarchical clustering methods. For example, BIRCH, by Zhang, Ramakrishnan, and Livny [ZRL96], first performs hierarchical clustering with a CF-tree before applying other techniques. Hierarchical clustering can also be performed by sophisticated linkage analysis, transformation, or nearest-neighbor analysis, such as CURE by Guha, Rastogi, and Shim [GRS98]; ROCK (for clustering nominal attributes) by Guha, Rastogi, and Shim [GRS99]; and Chameleon by Karypis, Han, and Kumar [KHK99].

Ward [War63] proposed the Ward's criterion. Murtagh and Legendre [ML14] surveyed how the Ward's criterion is implemented. Lance and Williams [LW67] proposed the Lance-Williams algorithm. A probabilistic hierarchical clustering framework following normal linkage algorithms and using probabilistic models to define cluster similarity was developed by Friedman [Fri03] and Heller and Ghahramani [HG05].

For density-based clustering methods, DBSCAN was proposed by Ester, Kriegel, Sander, and Xu [EKSS96]. Campello, Moulavi, Zimek, and Sander [CMZS15] developed both DBSCAN\* and HDBSCAN. Ankerst, Breunig, Kriegel, and Sander [ABKS99] developed OPTICS, a cluster-ordering method that facilitates density-based clustering without worrying about parameter specification. The DENCLUE algorithm, based on a set of density distribution functions, was proposed by Hinneburg and Keim [HK98]. Hinneburg and Gabriel [HG07] developed DENCLUE 2.0, which includes a new hill-climbing procedure for Gaussian kernels that adjusts the step size automatically.

STING, a grid-based multiresolution approach that collects statistical information in grid cells, was proposed by Wang, Yang, and Muntz [WYM97]. WaveCluster, developed by Sheikholeslami, Chatterjee, and Zhang [SCZ98], is a multiresolution clustering approach that transforms the original feature space by wavelet transform.

Scalable methods for clustering nominal data were studied by Gibson, Kleinberg, and Raghavan [GKR98]; Guha, Rastogi, and Shim [GRS99]; and Ganti, Gehrke, and Ramakrishnan [GGR99]. There are also many other clustering paradigms. For example, fuzzy clustering methods are discussed in Kaufman and Rousseeuw [KR90], Bezdek [Bez81], and Bezdek and Pal [BP92].

For high-dimensional clustering, an Apriori-based dimension-growth subspace clustering algorithm called CLIQUE was proposed by Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98]. It integrates density-based and grid-based clustering methods.

Recent studies have proceeded to clustering stream data (Babcock et al. [BBD<sup>+</sup>02]). A  $k$ -median-based data stream clustering algorithm was proposed by Guha, Mishra, Motwani, and O'Callaghan [GMMO00] and by O'Callaghan et al. [OMM<sup>+</sup>02]. A method for clustering evolving data streams was proposed by Aggarwal, Han, Wang, and Yu [AHWY03]. A framework for projected clustering of high-dimensional data streams was proposed by Aggarwal, Han, Wang, and Yu [AHWY04].

Clustering evaluation is discussed in a few monographs and survey articles such as Jain and Dubes [JD88] and Halkidi, Batistakis, and Vazirgiannis [HBV01]. The Hopkins statistic was proposed by Hopkins and Skellam [HS54]. The problem of determining the number of clusters in a data set was discussed by Sugar and James [SJ03] and Cordeiro De Amorim and Hennig [CH15b], for example.

The extrinsic methods for clustering quality evaluation are extensively explored. Some recent studies include Meilă [Mei03,Mei05] and Amigó, Gonzalo, Artiles, and Verdejo [AGAV09]. The four essential criteria introduced in this chapter are formulated in Amigó, Gonzalo, Artiles, and Verdejo [AGAV09], whereas some individual criteria were also mentioned earlier, for example, in Meilă [Mei03] and Rosenberg and Hirschberg [RH07]. Bagga and Baldwin [BB98] introduced the BCubed metrics. The silhouette coefficient is described in Kaufman and Rousseeuw [KR90].

# Cluster analysis: advanced methods

# 9

You learned the fundamentals of cluster analysis in Chapter 8. In this chapter, we discuss advanced topics of cluster analysis. Specifically, we investigate four major perspectives:

- **Probabilistic model-based clustering:** Section 9.1 introduces a general framework and a method for deriving clusters where each object is assigned a probability of belonging to a cluster. Probabilistic model-based clustering is widely used in many data mining applications, such as text mining and natural language processing.
- **Clustering high-dimensional data:** When the dimensionality is high, conventional distance measures can be dominated by noise. Cluster analysis on high-dimensional data is an important area. This chapter introduces the general principles and some fundamental methods. Section 9.2 introduces the basic framework for cluster analysis on high-dimensional data. Section 9.3 discusses biclustering, which clusters objects and attributes simultaneously and enjoys many applications, such as bioinformatics and recommender systems. Section 9.4 discusses dimensionality reduction methods for clustering.
- **Clustering graph and network data:** Graph and network data is increasingly popular in applications such as online social networks, the World Wide Web, and digital libraries. In Section 9.5, you will study the key issues in clustering graph and network data, including similarity measurement and clustering methods.
- **Semisupervised clustering:** In our discussion so far, we do not assume any user knowledge in clustering. In some applications, however, various user knowledge may be available, which can be used to strengthen cluster analysis. For example, a user may provide useful constraints risen from background knowledge or spatial distribution of the objects to be clustered. Semisupervised clustering provides a general framework to accommodate such background knowledge. You will learn how to conduct semisupervised clustering in Section 9.6.

By the end of this chapter, you will have a good grasp of the issues and techniques regarding advanced cluster analysis.

In this chapter, we write a variable of an object in *italic* if we do not involve the vector representation of the object. If vector representations and matrix operations are involved, then variables in **boldface** are used.

---

## 9.1 Probabilistic model-based clustering

In most of the cluster analysis methods we have discussed so far, each data object can be assigned to only one of a number of clusters. This kind of strict cluster assignments are required in some applica-

tions, such as assigning customers to marketing managers. However, in some other applications, this rigid requirement may not be desirable. In this section, we demonstrate the need for fuzzy or flexible cluster assignment in some applications and introduce a general method to compute probabilistic clusters and assignments.

“*In what situations may a data object belong to more than one cluster?*” Consider Example 9.1.

**Example 9.1. Clustering product reviews.** Imagine that an e-commerce company has an online store, where customers not only purchase online, but also create reviews of products. Not every product receives reviews; instead, some products may have many reviews, whereas many others have none or only a few. Moreover, a review may involve multiple products. Thus as the review editor of the company, your task is to cluster the reviews.

Ideally, a cluster is about a *topic*, for example, a group of products, services, or issues that are highly related. Assigning a review to one cluster exclusively would not work well for your task. Suppose there is a cluster for “cameras and camcorders” and another for “computers.” What if a review talks about the compatibility between a camcorder and a computer? The review is related to both clusters; however, it does not exclusively belong to either cluster.

You would like to use a clustering method that allows a review to belong to more than one cluster if the review indeed involves more than one topic. To reflect the strength that a review belongs to a cluster, you want the assignment of a review to a cluster to carry a weight representing the partial membership. □

The scenario where an object may belong to multiple clusters occurs often in many applications. Let us consider another example.

**Example 9.2. Clustering to study user search intent.** The online store of the e-commerce company discussed in Example 9.1 records all customer browsing and purchasing behavior in a log. An important data mining task is to use the log data to categorize and understand *user search intent*. For example, consider a user *session* (a short period in which a user interacts with the online store). Is the user searching for a product, making comparisons among different products, or looking for customer support information? Cluster analysis helps here because it is difficult to predefine user behavior patterns thoroughly. A cluster that contains similar user browsing trajectories may represent similar user behavior.

However, not every session belongs to only one cluster. For example, suppose the user sessions involving the purchase of digital cameras form one cluster, and the user sessions that compare laptop computers form another cluster. What if a user in one session makes an order for a digital camera and at the same time compares several laptop computers? Such a session should belong to both clusters to some extent. □

In this section, we systematically study the theme of clustering that allows an object to belong to more than one cluster. We start with the notion of fuzzy clusters in Section 9.1.1. We then generalize the concept to probabilistic model-based clusters in Section 9.1.2. In Section 9.1.3, we introduce the expectation-maximization algorithm, a general framework for mining such clusters.

**Table 9.1** A set of digital cameras and their sales at an e-commerce company.

| Camera | Sales (units) |
|--------|---------------|
| A      | 50            |
| B      | 1320          |
| C      | 860           |
| D      | 270           |

### 9.1.1 Fuzzy clusters

Given a set of objects,  $X = \{x_1, \dots, x_n\}$ , a **fuzzy set**  $S$  is a subset of  $X$  that allows each object in  $X$  to have a membership degree between 0 and 1. Formally, a fuzzy set,  $S$ , can be modeled as a function,  $F_S: X \rightarrow [0, 1]$ .

**Example 9.3. Fuzzy set.** The more digital camera units that are sold, the more popular the camera is. In an e-commerce company, we can use the following formula to compute the degree of popularity of a digital camera,  $o$ , given the sales of  $o$ :

$$pop(o) = \begin{cases} 1 & \text{if 1000 or more units of } o \text{ are sold} \\ \frac{i}{1000} & \text{if } i \text{ (} i < 1000 \text{) units of } o \text{ are sold.} \end{cases} \quad (9.1)$$

Function  $pop()$  defines a fuzzy set of popular digital cameras. For example, suppose the sales of digital cameras at the e-commerce company are as shown in Table 9.1. The fuzzy set of popular digital cameras is  $\{A(0.05), B(1), C(0.86), D(0.27)\}$ , where the degrees of membership are written in parentheses.  $\square$

We can apply the fuzzy set idea on clusters. That is, given a set of objects, a cluster is a fuzzy set of objects. Such a cluster is called a fuzzy cluster. Consequently, a clustering contains multiple fuzzy clusters.

Formally, given a set of objects,  $o_1, \dots, o_n$ , a **fuzzy clustering** of  $k$  **fuzzy clusters**,  $C_1, \dots, C_k$ , can be represented using a **partition matrix**,  $\mathbf{M} = [w_{ij}]$  ( $1 \leq i \leq n, 1 \leq j \leq k$ ), where  $w_{ij}$  is the membership degree of object  $o_i$  in fuzzy cluster  $C_j$ . The partition matrix should satisfy the following three requirements:

- For each object,  $o_i$ , and cluster,  $C_j$ ,  $0 \leq w_{ij} \leq 1$ . This requirement enforces that a fuzzy cluster is a fuzzy set.
- For each object,  $o_i$ ,  $\sum_{j=1}^k w_{ij} = 1$ . This requirement ensures that every object participates in the clustering with the equivalent total weight.
- For each cluster,  $C_j$ ,  $0 < \sum_{i=1}^n w_{ij} < n$ . This requirement ensures that for every cluster, there is at least one object for which the membership value is nonzero.

**Example 9.4. Fuzzy clusters.** Suppose the online store of the e-commerce company has six reviews. The keywords contained in these reviews are listed in Table 9.2.



**Table 9.2 Set of reviews and the keywords used.**

| Review_ID | Keywords                       |
|-----------|--------------------------------|
| $R_1$     | digital camera, lens           |
| $R_2$     | digital camera                 |
| $R_3$     | lens                           |
| $R_4$     | digital camera, lens, computer |
| $R_5$     | computer, CPU                  |
| $R_6$     | computer, computer game        |

We can group the reviews into two fuzzy clusters,  $C_1$  and  $C_2$ .  $C_1$  is for “digital camera” and “lens,” and  $C_2$  is for “computer.” The partition matrix is

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \frac{2}{3} & \frac{1}{3} \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

Here, we use the keywords “digital camera” and “lens” as the features of cluster  $C_1$  and “computer” as the feature of cluster  $C_2$ . For review  $R_i$  and cluster  $C_j$  ( $1 \leq i \leq 6, 1 \leq j \leq 2$ ),  $w_{ij}$  is defined as

$$w_{ij} = \frac{|R_i \cap C_j|}{|R_i \cap (C_1 \cup C_2)|} = \frac{|R_i \cap C_j|}{|R_i \cap \{\text{digital camera, lens, computer}\}|}.$$

In this fuzzy clustering, review  $R_4$  belongs to clusters  $C_1$  and  $C_2$  with membership degrees  $\frac{2}{3}$  and  $\frac{1}{3}$ , respectively.  $\square$

“How can we evaluate how well a fuzzy clustering describes a data set?” Consider a set of objects,  $o_1, \dots, o_n$ , and a fuzzy clustering  $\mathcal{C}$  of  $k$  clusters,  $C_1, \dots, C_k$ . Let  $\mathbf{M} = [w_{ij}]$  ( $1 \leq i \leq n, 1 \leq j \leq k$ ) be the partition matrix. Let  $c_1, \dots, c_k$  be the centers of clusters  $C_1, \dots, C_k$ , respectively. Here, a center can be defined either as the mean or the medoid or in other ways specific to the application.

As discussed in Chapter 8, the distance or similarity between an object and the center of the cluster to which the object is assigned can be used to measure how well the object belongs to the cluster. This idea can be extended to fuzzy clustering. For any object  $o_i$  and cluster  $C_j$ , if  $w_{ij} > 0$ , then  $\text{dist}(o_i, c_j)$  measures how well  $o_i$  is represented by  $c_j$ , and thus belongs to cluster  $C_j$ . Because an object can participate in more than one cluster, the sum of distances to the corresponding cluster centers weighted by the degrees of membership captures how well the object fits the clustering.

Formally, for an object  $o_i$ , the **sum of the squared error** (SSE) is given by

$$\text{SSE}(o_i) = \sum_{j=1}^k w_{ij}^p \text{dist}(o_i, c_j)^2, \quad (9.2)$$

where the parameter  $p$  ( $p \geq 1$ ) controls how fuzzy the clusters would be. The larger the value of  $p$ , the fuzzier the clusters. Orthogonally, the SSE for a cluster,  $C_j$ , is

$$\text{SSE}(C_j) = \sum_{i=1}^n w_{ij}^p \text{dist}(o_i, c_j)^2. \quad (9.3)$$

Finally, the SSE of the clustering is defined as

$$\text{SSE}(C) = \sum_{i=1}^n \sum_{j=1}^k w_{ij}^p \text{dist}(o_i, c_j)^2. \quad (9.4)$$

The SSE can be used to measure how well a fuzzy clustering fits a data set.

Fuzzy clustering is also called *soft clustering* because it allows an object to belong to more than one cluster. It is easy to see that traditional (rigid) clustering, which enforces each object to belong to only one cluster exclusively, is a special case of fuzzy clustering. We defer the discussion of how to compute fuzzy clustering to Section 9.1.3.

## 9.1.2 Probabilistic model-based clusters

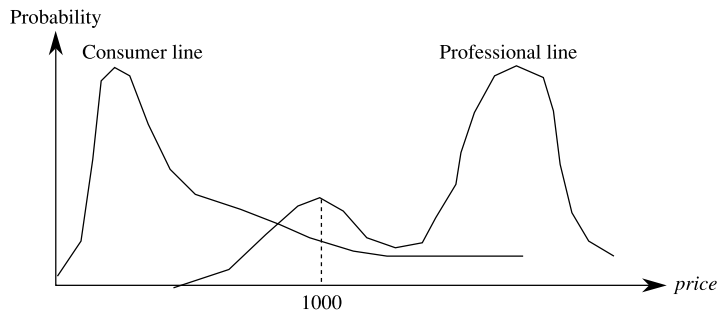
“Fuzzy clusters (Section 9.1.1) provide the flexibility of allowing an object to participate in multiple clusters. Is there a general framework to specify clusterings where objects may participate in multiple clusters in a probabilistic way?” In this section, we introduce the general notion of probabilistic model-based clusters to answer this question.

As discussed in Chapter 8, we conduct cluster analysis on a data set because we assume that the objects in the data set in fact belong to different inherent categories. Recall that clustering tendency analysis (Section 8.5.1) can be used to examine whether a data set contains objects that may lead to meaningful clusters. Here, the inherent categories hidden in the data are *latent*, which means that they cannot be directly observed. Instead, we have to infer them using the data observed. For example, the topics hidden in a set of reviews in the online store of an e-commerce company are latent because one cannot read the topics directly. However, the topics can be inferred from the reviews because each review is about one or multiple topics.

Therefore the goal of cluster analysis is to find hidden categories. A data set that is the subject of cluster analysis can be regarded as a sample of the possible instances of the hidden categories, but without any category labels. The clusters derived from cluster analysis are inferred using the data set, and are designed to approach the hidden categories.

Statistically, we can assume that a hidden category is a distribution over the data space, which can be mathematically represented using a probability density function (or distribution function). We call such a hidden category a *probabilistic cluster*. For a probabilistic cluster,  $C$ , its probability density function,  $f$ , and a point,  $o$ , in the data space,  $f(o)$  is the relative likelihood that an instance of  $C$  appears at  $o$ .

**Example 9.5. Probabilistic clusters.** Suppose the digital cameras sold by an e-commerce company can be divided into two categories:  $C_1$ , a consumer line (e.g., point-and-shoot cameras), and  $C_2$ , a professional line (e.g., single-lens reflex cameras). Their respective probability density functions,  $f_{\text{consumer}}$  and  $f_{\text{professional}}$ , are shown in Fig. 9.1 with respect to the attribute *price*.



**FIGURE 9.1**

The probability density functions of two probabilistic clusters.

For a price value of, say, \$1000,  $f_{\text{consumer}}(1000)$  is the relative likelihood that the price of a consumer-line camera is \$1000. Similarly,  $f_{\text{professional}}(1000)$  is the relative likelihood that the price of a professional-line camera is \$1000.

The probability density functions,  $f_{\text{consumer}}$  and  $f_{\text{professional}}$ , cannot be observed directly. Instead, the company can only infer these distributions by analyzing the prices of the digital cameras it sells. Moreover, a camera often does not come with a well-determined category (e.g., “consumer line” or “professional line”). Instead, such categories are typically based on user background knowledge and can vary. For example, a camera in the *prosumer* segment may be regarded at the high end of the consumer line by some customers and the low end of the professional line by others.

As an analyst, you can consider each category as a probabilistic cluster and conduct cluster analysis on the price of cameras to approach these categories.  $\square$

Suppose we want to find  $k$  probabilistic clusters,  $C_1, \dots, C_k$ , through cluster analysis. For a data set,  $D$ , of  $n$  objects, we can regard  $D$  as a finite sample of the possible instances of the clusters. Conceptually, we can assume that  $D$  is formed as follows. Each cluster,  $C_j$  ( $1 \leq j \leq k$ ), is associated with a probability,  $\omega_j$ , that some instance is sampled from the cluster. It is often assumed that  $\omega_1, \dots, \omega_k$  are given as part of the problem setting, and that  $\sum_{j=1}^k \omega_j = 1$ , which ensures that all objects are generated by the  $k$  clusters. Here, parameter  $\omega_j$  captures the background knowledge about the relative population of cluster  $C_j$ .

We then run the following two steps to generate an object in  $D$ . The steps are executed  $n$  times in total to generate  $n$  objects,  $o_1, \dots, o_n$ , in  $D$ .

1. Choose a cluster,  $C_j$ , according to probabilities  $\omega_1, \dots, \omega_k$ .
2. Choose an instance of  $C_j$  according to its probability density function,  $f_j$ .

The data generation process here is the basic assumption in mixture models. Formally, a **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters. Conceptually, each observed object is generated independently by two steps: first choosing a probabilistic cluster according to the probabilities of the clusters, and then choosing a sample according to the probability density function of the chosen cluster.

Given a data set  $D$  and  $k$ , the number of clusters required, the task of *probabilistic model-based cluster analysis* is to infer a set of  $k$  probabilistic clusters that is most likely to generate  $D$  using this data generation process. An important question remaining is how we can measure the likelihood that a set of  $k$  probabilistic clusters and their probabilities will generate an observed data set.

Consider a set,  $\mathcal{C}$ , of  $k$  probabilistic clusters,  $C_1, \dots, C_k$ , with probability density functions  $f_1, \dots, f_k$ , respectively, and their probabilities,  $\omega_1, \dots, \omega_k$ . For an object,  $o$ , the probability that  $o$  is generated by cluster  $C_j$  ( $1 \leq j \leq k$ ) is given by  $P(o|C_j) = \omega_j f_j(o)$ . Therefore the probability that  $o$  is generated by the set  $\mathcal{C}$  of clusters is

$$P(o|\mathcal{C}) = \sum_{j=1}^k \omega_j f_j(o). \quad (9.5)$$

Since the objects are assumed to have been generated independently, for a data set,  $D = \{o_1, \dots, o_n\}$ , of  $n$  objects, we have

$$P(D|\mathcal{C}) = \prod_{i=1}^n P(o_i|\mathcal{C}) = \prod_{i=1}^n \sum_{j=1}^k \omega_j f_j(o_i). \quad (9.6)$$

Now, it is clear that the task of probabilistic model-based cluster analysis on a data set,  $D$ , is to find a set  $\mathcal{C}$  of  $k$  probabilistic clusters such that  $P(D|\mathcal{C})$  is maximized. Maximizing  $P(D|\mathcal{C})$  is often intractable because, in general, the probability density function of a cluster can take an arbitrarily complicated form. To make probabilistic model-based clusters computationally feasible, we often compromise by assuming that the probability density functions are parameterized distributions.

Formally, let  $o_1, \dots, o_n$  be the  $n$  observed objects, and  $\Theta_1, \dots, \Theta_k$  be the parameters of the  $k$  distributions, denoted by  $\mathbf{O} = \{o_1, \dots, o_n\}$  and  $\Theta = \{\Theta_1, \dots, \Theta_k\}$ , respectively. Then, for any object  $o_i \in \mathbf{O}$  ( $1 \leq i \leq n$ ), Eq. (9.5) can be rewritten as

$$P(o_i|\Theta) = \sum_{j=1}^k \omega_j P_j(o_i|\Theta_j), \quad (9.7)$$

where  $P_j(o_i|\Theta_j)$  is the probability that  $o_i$  is generated from the  $j$ th distribution using parameter  $\Theta_j$ . Consequently, Eq. (9.6) can be rewritten as

$$P(\mathbf{O}|\Theta) = \prod_{i=1}^n \sum_{j=1}^k \omega_j P_j(o_i|\Theta_j). \quad (9.8)$$

Using the parameterized probability distribution models, the task of probabilistic model-based cluster analysis is to infer a set of parameters,  $\Theta$ , that maximizes Eq. (9.8).

**Example 9.6. Univariate Gaussian mixture model.** Let us use univariate Gaussian distributions as an example to illustrate probabilistic model-based clustering. That is, we assume that the probability density function of each cluster follows a 1-D Gaussian distribution. Suppose there are  $k$  clusters. The two parameters for the probability density function of each cluster are center,  $\mu_j$ , and standard deviation,  $\sigma_j$  ( $1 \leq j \leq k$ ). We denote the parameters as  $\Theta_j = (\mu_j, \sigma_j)$  and  $\Theta = \{\Theta_1, \dots, \Theta_k\}$ . Let the data set be  $\mathbf{O} = \{o_1, \dots, o_n\}$ , where  $o_i$  ( $1 \leq i \leq n$ ) is a real number. For any point,  $o_i \in \mathbf{O}$ , we have

$$P(o_i|\Theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma_j^2}}. \quad (9.9)$$

Assuming that each cluster has the same probability, that is  $\omega_1 = \omega_2 = \dots = \omega_k = \frac{1}{k}$ , and plugging Eq. (9.9) into Eq. (9.7), we have

$$P(o_i|\Theta) = \frac{1}{k} \sum_{j=1}^k \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma_j^2}}. \quad (9.10)$$

Applying Eq. (9.8), we have

$$P(\mathbf{O}|\Theta) = \frac{1}{k} \prod_{i=1}^n \sum_{j=1}^k \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i-\mu_j)^2}{2\sigma_j^2}}. \quad (9.11)$$

The task of probabilistic model-based cluster analysis using a univariate Gaussian mixture model is to infer  $\Theta$  such that Eq. (9.11) is maximized.  $\square$

### 9.1.3 Expectation-maximization algorithm

“How can we compute fuzzy clusterings and probabilistic model-based clusterings?” In this section, we introduce a principled approach. Let us start with a review of the  $k$ -means clustering problem and the  $k$ -means algorithm studied in Section 8.2.

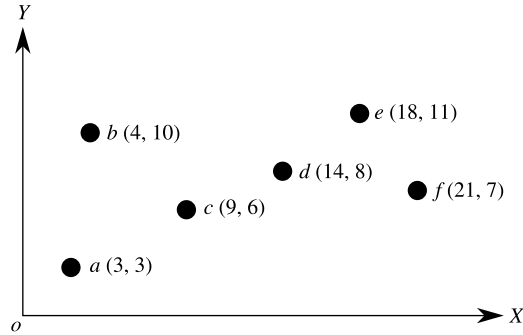
It can easily be shown that  $k$ -means clustering is a special case of fuzzy clustering (Exercise 9.1). The  $k$ -means algorithm iterates until the clustering cannot be improved. Each iteration consists of two steps:

**The expectation step (E-step):** Given the current cluster centers, each object is assigned to the cluster with a center that is closest to the object. Here, an object is expected to belong to the closest cluster.

**The maximization step (M-step):** Given the cluster assignment, for each cluster, the algorithm adjusts the center so that the sum of the distances from the objects assigned to this cluster and the new center is minimized. That is, the similarity of objects assigned to a cluster is maximized.

We can generalize this two-step method to tackle fuzzy clustering and probabilistic model-based clustering. In general, an **expectation-maximization (EM) algorithm** is a framework that approaches maximum likelihood or maximum a posteriori estimates of parameters in statistical models. In the context of fuzzy or probabilistic model-based clustering, an EM algorithm starts with an initial set of parameters and iterates until the clustering cannot be improved, that is, until the clustering converges or the change is sufficiently small (less than a preset threshold). Each iteration also consists of two steps:

- The **expectation step** assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters.
- The **maximization step** finds the new clustering or parameters that minimize the SSE in fuzzy clustering (Eq. (9.4)) or the expected likelihood in probabilistic model-based clustering.


**FIGURE 9.2**

Data set for fuzzy clustering.

**Example 9.7. Fuzzy clustering using the EM algorithm.** Consider the six points in Fig. 9.2, where the coordinates of the points are also shown. Let us compute two fuzzy clusters using the EM algorithm.

We randomly select two points, say  $c_1 = a$  and  $c_2 = b$ , as the initial centers of the two clusters. The first iteration conducts the expectation step and the maximization step as follows.

In the **E-step**, for each point we calculate its membership degree in each cluster. For any point  $o$ , we assign  $o$  to  $c_1$  with membership weight

$$\frac{\frac{1}{\text{dist}(o, c_1)^2}}{\frac{1}{\text{dist}(o, c_1)^2} + \frac{1}{\text{dist}(o, c_2)^2}} = \frac{\text{dist}(o, c_2)^2}{\text{dist}(o, c_1)^2 + \text{dist}(o, c_2)^2}$$

and to  $c_2$  with membership weight  $\frac{\text{dist}(o, c_1)^2}{\text{dist}(o, c_1)^2 + \text{dist}(o, c_2)^2}$ , where  $\text{dist}(\cdot)$  is the Euclidean distance. The rationale is that, if  $o$  is close to  $c_1$  and  $\text{dist}(o, c_1)$  is small, the membership degree of  $o$  with respect to  $c_1$  should be high. We also normalize the membership degrees so that the sum of degrees for an object is equal to 1.

For point  $a$ , we have  $w_{a, c_1} = 1$  and  $w_{a, c_2} = 0$ . That is,  $a$  exclusively belongs to  $c_1$ . For point  $b$ , we have  $w_{b, c_1} = 0$  and  $w_{b, c_2} = 1$ . For point  $c$ , we have  $w_{c, c_1} = \frac{41}{45+41} = 0.48$  and  $w_{c, c_2} = \frac{45}{45+41} = 0.52$ . The degrees of membership of the other points are shown in the partition matrix in Table 9.3.  $\square$

In the **M-step**, we recalculate the centroids according to the partition matrix, minimizing the SSE given in Eq. (9.4) where we assume the parameter  $p = 2$  in this example. The new centroid should be adjusted to

$$c_j = \frac{\sum_{\text{each point } o} w_{o, c_j}^2 o}{\sum_{\text{each point } o} w_{o, c_j}^2}, \quad (9.12)$$

where  $j = 1, 2$ .

**Table 9.3** Intermediate results from the first three iterations of the EM algorithm in Example 9.7.

| Iteration | E-Step                                                                                                                            | M-Step                                        |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| 1         | $\mathbf{M}^T = \begin{bmatrix} 1 & 0 & 0.48 & 0.42 & 0.41 & 0.47 \\ 0 & 1 & 0.52 & 0.58 & 0.59 & 0.53 \end{bmatrix}$             | $c_1 = (8.47, 5.12)$<br>$c_2 = (10.42, 8.99)$ |
| 2         | $\mathbf{M}^T = \begin{bmatrix} 0.73 & 0.49 & 0.91 & 0.26 & 0.33 & 0.42 \\ 0.27 & 0.51 & 0.09 & 0.74 & 0.67 & 0.58 \end{bmatrix}$ | $c_1 = (8.51, 6.11)$<br>$c_2 = (14.42, 8.69)$ |
| 3         | $\mathbf{M}^T = \begin{bmatrix} 0.80 & 0.76 & 0.99 & 0.02 & 0.14 & 0.23 \\ 0.20 & 0.24 & 0.01 & 0.98 & 0.86 & 0.77 \end{bmatrix}$ | $c_1 = (6.40, 6.24)$<br>$c_2 = (16.55, 8.64)$ |

In this example,

$$c_1 = \left( \frac{1^2 \times 3 + 0^2 \times 4 + 0.48^2 \times 9 + 0.42^2 \times 14 + 0.41^2 \times 18 + 0.47^2 \times 21}{1^2 + 0^2 + 0.48^2 + 0.42^2 + 0.41^2 + 0.47^2}, \frac{1^2 \times 3 + 0^2 \times 10 + 0.48^2 \times 6 + 0.42^2 \times 8 + 0.41^2 \times 11 + 0.47^2 \times 7}{1^2 + 0^2 + 0.48^2 + 0.42^2 + 0.41^2 + 0.47^2} \right)$$

$$= (8.47, 5.12)$$

and

$$c_2 = \left( \frac{0^2 \times 3 + 1^2 \times 4 + 0.52^2 \times 9 + 0.58^2 \times 14 + 0.59^2 \times 18 + 0.53^2 \times 21}{0^2 + 1^2 + 0.52^2 + 0.58^2 + 0.59^2 + 0.53^2}, \frac{0^2 \times 3 + 1^2 \times 10 + 0.52^2 \times 6 + 0.58^2 \times 8 + 0.59^2 \times 11 + 0.53^2 \times 7}{0^2 + 1^2 + 0.52^2 + 0.58^2 + 0.59^2 + 0.53^2} \right)$$

$$= (10.42, 8.99).$$

We repeat the iterations, where each iteration contains an E-step and an M-step. Table 9.3 shows the results from the first three iterations. The algorithm stops when the cluster centers converge or the change is small enough.

“How can we apply the EM algorithm to compute probabilistic model-based clustering?” Let us use a univariate Gaussian mixture model (Example 9.6) to illustrate.

**Example 9.8. Using the EM algorithm for mixture models.** Given a set of objects,  $\mathbf{O} = \{o_1, \dots, o_n\}$ , we want to mine a set of parameters,  $\Theta = \{\Theta_1, \dots, \Theta_k\}$ , such that  $P(\mathbf{O}|\Theta)$  in Eq. (9.11) is maximized, where  $\Theta_j = (\mu_j, \sigma_j)$  are the mean and standard deviation, respectively, of the  $j$ th univariate Gaussian distribution, ( $1 \leq j \leq k$ ).

We can apply the EM algorithm. We assign random values to parameters  $\Theta$  as the initial values. We then iteratively conduct the E-step and the M-step as follows until the parameters converge or the change is sufficiently small.

In the **E-step**, for each object,  $o_i \in \mathbf{O}$  ( $1 \leq i \leq n$ ), we calculate the probability that  $o_i$  belongs to each distribution, that is,

$$P(\Theta_j|o_i, \Theta) = \frac{P(o_i|\Theta_j)}{\sum_{l=1}^k P(o_i|\Theta_l)}. \quad (9.13)$$

In the **M-step**, we adjust the parameters  $\Theta$  so that the expected likelihood  $P(\mathbf{O}|\Theta)$  in Eq. (9.11) is maximized. This can be achieved by setting

$$\mu_j = \frac{1}{k} \sum_{i=1}^n o_i \frac{P(\Theta_j|o_i, \Theta)}{\sum_{l=1}^n P(\Theta_j|o_l, \Theta)} = \frac{1}{k} \frac{\sum_{i=1}^n o_i P(\Theta_j|o_i, \Theta)}{\sum_{i=1}^n P(\Theta_j|o_i, \Theta)} \quad (9.14)$$

and

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^n P(\Theta_j|o_i, \Theta)(o_i - \mu_j)^2}{\sum_{i=1}^n P(\Theta_j|o_i, \Theta)}}. \quad (9.15)$$

□

In many applications, probabilistic model-based clustering has been shown to be effective because it is more general than partitioning methods and fuzzy clustering methods. A distinct advantage is that appropriate statistical models can be used to capture latent clusters. The EM algorithm is commonly used to handle many learning problems in data mining and statistics due to its simplicity. Note that, in general, the EM algorithm may not converge to the optimal solution. It may instead converge to a local maximum. Many heuristics have been explored to avoid this. For example, we could run the EM process multiple times using different random initial values. Furthermore, the EM algorithm can be very costly if the number of distributions is large or the data set contains many observed data points.

---

## 9.2 Clustering high-dimensional data

The clustering methods we have studied so far work well when the dimensionality of a data set is not high, that is, having, say, less than 10 attributes. There are, however, important applications of high dimensionality. “*How can we conduct cluster analysis on high-dimensional data?*”

In this section, we study approaches to clustering high-dimensional data. We first discuss why clustering high-dimensional data is challenging and categorize the existing methods in clustering high-dimensional data. Then we explain several representative methods, including some methods clustering in axis-parallel subspaces and those in arbitrarily oriented subspaces. Section 9.3 explores biclustering methods, which are popularly used in biological data analysis where there are many high-dimensional data sets.

### 9.2.1 Why is clustering high-dimensional data challenging?

Before we present any specific methods for clustering high-dimensional data, let us first demonstrate the needs of cluster analysis on high-dimensional data using some examples. We examine the challenges that call for new methods. We then categorize the major ideas and methods in clustering high-dimensional data.

#### ***Motivations of clustering analysis on high-dimensional data***

In some applications, a data object may be described by many attributes. Such objects are referred to as in a high-dimensional data space.



| Customer | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ | $P_9$ | $P_{10}$ |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| Ada      | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0        |
| Bob      | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1        |
| Cathy    | 1     | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 1        |

**Example 9.9. High-dimensional data and clustering.** Consider an e-commerce company that keeps track of the products purchased by every customer. As a customer-relationship manager, you want to cluster customers into groups according to what they purchased from the company.

The customer purchase data are of very high dimensionality. The company carries tens of thousands of products. Therefore a customer's purchase profile, which is a vector of the products carried by the company, has tens of thousands of dimensions.

“Are the traditional distance measures, which are frequently used in low-dimensional cluster analysis, also effective on high-dimensional data?” Consider the customers in Table 9.4, where 10 products,  $P_1, \dots, P_{10}$ , are used in demonstration. If a customer purchases a product, a 1 is set at the corresponding bit; otherwise, a 0 appears. Let us calculate the Euclidean distances among Ada, Bob, and Cathy. It is easy to see that

$$\text{dist}(\text{Ada}, \text{Bob}) = \text{dist}(\text{Bob}, \text{Cathy}) = \text{dist}(\text{Ada}, \text{Cathy}) = \sqrt{2}.$$

According to Euclidean distance, the three customers are equivalently similar (or dissimilar) to each other. However, a close look tells us that Ada should be more similar to Cathy than to Bob because Ada and Cathy share one common purchased item,  $P_1$ .  $\square$

As shown in Example 9.9, the traditional distance measures can be ineffective on high-dimensional data. Such distance measures may be dominated by the noise in many dimensions. Therefore clusters in the full, high-dimensional original data space may be unreliable, and finding such clusters may not be meaningful.

In general, when the dimensionality increases, the data space allows more and more complicated relations among objects. Such relations are harder and harder to detect using the traditional pairwise distance measurements. Moreover, high-dimensional spaces naturally allow more noise. This is known as the “*curse of dimensionality*.” More concretely, the curse of dimensionality is caused by the following four reasons.

- **Many irrelevant or correlated attributes.** Many attributes in a high-dimensional data set may not be relevant to an analytic task. For example, customer profiles may contain tens or even hundreds of attributes. Many attributes are or should be irrelevant to the task of credit card preapproval, such as gender, preferred language in communication, ethnic, purchases of dairy products, and distance to the store. Moreover, ideally the attributes in a data set are independent from each other, so that each attribute provides some independent and nonredundant information. However, in a high-dimensional data set, more often than not some attributes are correlated. Those correlated attributes provide redundant information in analysis and thus may lead to biases. For example, suppose microphones and webcams are often purchased together and thus have strong correlation. Then, when we include those two as independent attributes in clustering analysis, such as calculating the similarity between

two customers, the correlated information on those two attributes may lead to bias against other independent attributes, such as purchases of sport watches. In other words, the intrinsic dimensionality of a high-dimensional data set may be substantially lower than the embedding dimensionality, that is, the dimensionality of the data space. The existence of many irrelevant or correlated attributes in high-dimensional data sets leaves a large space for random noise as false positive signals and biases and thus raises a lot of challenges to clustering analysis.

- **Data sparsity.** Although a high-dimensional data set has many attributes, an object, however, often has nontrivial values on only a small number of attributes. For example, consider the scenario in Example 9.9, although the company carries tens of thousands of products and thus the customer purchase data set has tens of thousands of attributes correspondingly, one customer may only purchase tens or several hundreds of different products. In other words, a customer in the data set may not have nontrivial values on most of the attributes. The sparsity posts a great challenge to clustering high-dimensional data. When two objects are compared in the full space, they look like similar to each other on the majority of the attributes where they both take trivial values. The inherent similarity and the clusters are manifested by a very small number of attributes. Search for those critical attributes becomes a key to finding meaningful clusters in high-dimensional data.
- **Distance concentration effect of similarity measures.** Euclidean distance and  $L_p$ -norms are frequently used in clustering low-dimensional data. Recall the general formula of  $L_p$ -norm distance

$$\|\mathbf{x} - \mathbf{y}\|_p = \sqrt[p]{\sum_{i=1}^d |\mathbf{x}[i] - \mathbf{y}[i]|^p}, \quad (9.16)$$

where  $d$  is the dimensionality, and  $\mathbf{x}$  and  $\mathbf{y}$  are  $d$ -dimensional vectors. In high-dimensional data, the distance concentration effect takes control. That is, as dimensionality increases, the distances between far and close neighbors become very similar.<sup>1</sup> Due to the distance concentration effect, in high-dimensional data sets, the  $L_p$ -norm based distance measures lose the capability of distinguishing close neighbors from far away neighbors, and thus finding the groups of objects inherently close to each other using  $L_p$ -norm based distance measures becomes very difficult.

- **Difficulty in optimization.** Most of the formulations of clustering analysis tasks try to optimize an objective function. For example, the  $k$ -means clustering problem tries to minimize the sum of squared errors of all objects. In general, the difficulty of optimizing an objective function increases exponentially when the number of attributes, which are the number of variables involved in the function, increases. In other words, the optimization process in clustering high-dimensional data is more demanding.

### High-dimensional clustering models

A major challenge is how to create appropriate models for clusters in high-dimensional data. Unlike conventional clusters in low-dimensional spaces, clusters hidden in high-dimensional data are often significantly smaller, and are manifested by a dramatically smaller subset of attributes. For example, when clustering customer-purchase data, we would not expect many users to have similar purchase

<sup>1</sup> Mathematically, if  $\lim_{d \rightarrow \infty} \text{var}\left(\frac{|x|}{E[|x|]}\right) = 0$ ,  $\lim_{d \rightarrow \infty} \frac{D_{max} - D_{min}}{D_{min}} = 0$ , where  $E[|x|]$  is the length of the mean point vector in a data set, and  $D_{max}$  and  $D_{min}$  are the farthest point and closest point distances, respectively.

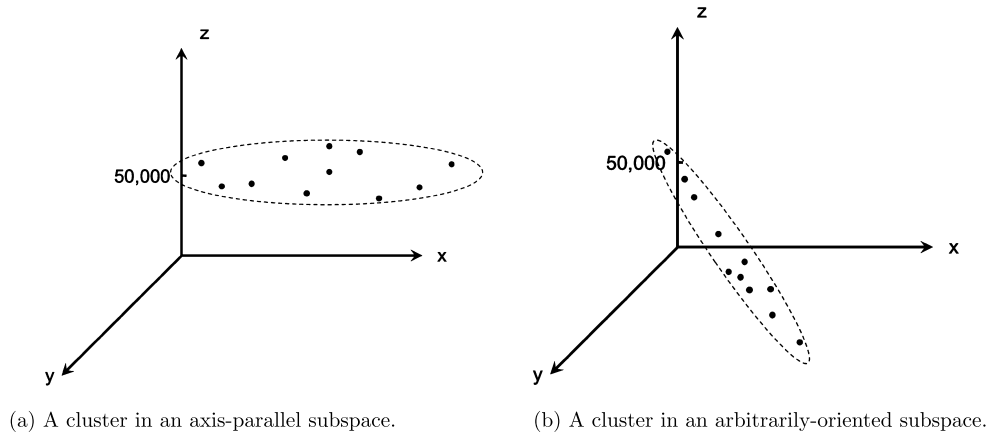
**FIGURE 9.3**

Illustration of clusters in axis-parallel and arbitrarily-oriented subspaces.

patterns. Those purchase patterns are manifested by only tens of products against tens of thousands of products carried by the store. Searching for such small but meaningful clusters is like finding needles in a haystack.

One essential principle in clustering high-dimensional data is to find clusters in subspaces. There are two different kinds of subspaces that clustering methods may target at.

First, a method may find clusters in **axis-parallel subspaces**. An axis-parallel subspace for a cluster is a subset of attributes that are relevant, and the cluster is full-dimensional in the subspace. All objects of the cluster can be projected into the subspace and form a hyperplane parallel to the irrelevant axes. Fig. 9.3(a) illustrates the idea. A cluster consists of several objects having values around 50,000 on attributes  $z$ . Attributes  $x$  and  $y$  are irrelevant to the cluster. Indeed, those objects spread widely on attributes  $x$  and  $y$ . Thus the subspace  $z = 5$  is a hyperplane parallel to both  $x$  and  $y$ .

Second, a method may find clusters in **arbitrarily oriented subspaces**. When two or more attributes are linearly correlated for a set of objects in a cluster, those objects are scattered along a hyperplane defined by the linear dependencies among those correlated attributes. Consequently, the subspace of the cluster is orthogonal to the hyperplane and is not parallel to those correlated attributes. For example, in Fig. 9.3(b), due to the correlation between attributes  $x$  and  $y$ , that is,  $x = y$ , the objects in the cluster are spread in the hyperplane  $x = y$ . The objects in the cluster have values around 50,000 on attribute  $z$ . The subspace is orthogonal to the hyperplane  $x = y$ .

### ***Categorization of high-dimensional clustering methods***

The clustering analysis methods on high-dimensional data can be divided into two categories.

The first category is the **clustering approaches**, which try to identify clusters in subspaces. These methods build on the basis of the traditional clustering methods we discussed in the last chapter, such as  $k$ -means, and incorporate with new techniques to contend high-dimensional data. Those techniques specific for tackling high-dimensional data can be further divided into three groups.

- The **subspace clustering methods** find all clusters in all subspaces of the entire data space. A data object may belong to zero, one, or multiple clusters simultaneously. Clusters may also substantially overlap in different subspaces. A representative method in this group is CLIQUE, which is briefly introduced in Section 8.4.3 as a grid-based method.
- The **projected clustering methods** partition a given data set into nonoverlapping subsets. In other words, a data object belongs to exactly one cluster. For each cluster, the methods search for the corresponding subspace. PROCLUS is a pioneering and representative projected clustering method.
- The **bi-clustering methods** cluster attributes and objects simultaneously. In other works, attributes and objects are treated in a symmetric manner in bi-clustering methods. This group of methods have been extensively used in biological data analysis and recommender systems. We will introduce bi-clustering in Section 9.3.

The second category is the **dimensionality-reduction methods**. As we discussed, in a high-dimensional data set, the intrinsic dimensionality may be much lower than the embedding dimensionality. Thus the dimension-reduction methods construct new attributes to approach the intrinsic dimensions and transform the objects from the original embedding data space to a constructed space of much lower dimensionality. Clustering analysis is then conducted in the constructed space. Dimensionality-reduction methods for clustering analysis will be discussed in Section 9.4.

## 9.2.2 Axis-parallel subspace approaches

Now, let us look at some representative subspace clustering methods and projected clustering methods. We use three methods, CLIQUE, PROCLUS, and LAC, to illustrate the essential ideas.

### ***CLIQUE: a subspace clustering method***

Subspace clustering methods find all clusters in all subspaces of the entire data space. CLIQUE is a representative. Recall that we introduce CLIQUE in Section 8.4.3 as a grid-based method. Let us recap the two major ideas in CLIQUE tackling high-dimensional data.

First, CLIQUE adopts a *bottom-up strategy*. It starts from low-dimensional subspaces and searches higher-dimensional subspaces only when there may be clusters in those higher-dimensional subspaces. Various pruning techniques are explored to reduce the number of higher-dimensional subspaces that need to be searched.

Second, CLIQUE allows overlaps among clusters in different subspaces. For example, in Fig. 8.22, the dense cell of salary \$30,000–40,000, age 35–40, and vacation 2–3 weeks in the 3-D subspace (salary, age, vacation) also belongs to the dense cell of salary \$30,000–40,000 and age 35–40, and the one of age 35–40 and vacation 2–3 weeks in the corresponding subspaces, and thus in turn belongs to the two clusters in the two subspaces. Every object in the 3-D dense cell also belongs to the two clusters.

### ***PROCLUS: a projected clustering method***

PROCLUS is a *k*-medoid–like method for projected clustering on high-dimensional data. PROCLUS works in three phases.

- In the *initialization phase*, PROCLUS generates *k* potential cluster centers, that is, medoids, using a greedy sample of the data set.

- In the *iterative phase*, PROCLUS progressively improves the quality of the medoids. To evaluate the quality of the clustering defined by the current set of  $k$  medoids, PROCLUS conducts two steps.
    - In the first step, PROCLUS finds the attributes for each medoid. For each medoid  $m_i$ , let  $\delta_i$  be the minimum distance from any other medoid to  $m_i$ . Then, the locality  $L_i$  is the set of objects in the data set that are within distance  $\delta_i$  from  $m_i$ . PROCLUS examines, on each attribute  $A_j$ , the average distance  $X_{m_i, A_j}$  from those objects in  $L_i$  to  $m_i$ . Let  $Y_i = \frac{\sum_{j=1}^d X_{m_i, A_j}}{d}$  be the average of the Manhattan segmental distances between the objects in  $L_i$  and  $m_i$  relative to the whole space, and  $\sigma_i = \sqrt{\frac{\sum_{j=1}^d (X_{m_i, A_j} - Y_i)^2}{d-1}}$  be the standard deviation, where  $d$  is the dimensionality. Then,  $Z_{m_i, A_j} = \frac{X_{m_i, A_j} - Y_i}{\sigma_i}$  indicates how the objects in  $L_i$  are correlated to the medoid  $m_i$  on attribute  $A_j$ , the smaller the better. PROCLUS picks  $k \cdot l$  attributes that have the smallest  $Z_{m_i, A_j}$  values associated with the current set of  $k$  medoids, so that each medoid is associated with at least two attributes. The attributes associated with a medoid are the subspace of the medoid.
    - In the second step, PROCLUS forms clusters. It assigns every data object to the medoid that has the smallest Manhattan segmental distance relative to the subspace of the medoid.
- The quality of a set of medoids can be assessed using the average Manhattan segmental distance from the data objects to the corresponding centroids of the clusters. Iteratively PROCLUS improves the quality until the best set of medoids is found.
- In the *refinement phase*, PROCLUS adjusts the subspaces of the clusters using only the data objects assigned to a cluster instead of the objects in  $L_i$ . In this way, the attributes after refinement approach the cluster subspaces better.

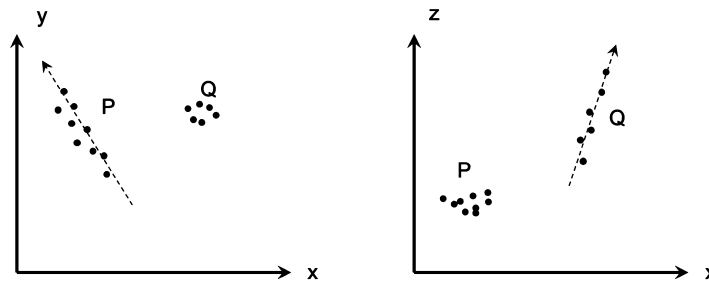
Comparing PROCLUS and CLIQUE, we can see two major differences between projected clustering (represented by PROCLUS) and subspace clustering (represented by CLIQUE). First, PROCLUS assigns one object to only one cluster. Thus clusters are exclusive from each other. Instead, CLIQUE allows overlap among clusters in different subspaces. One object may belong to multiple clusters. Second, PROCLUS searches subspaces for clusters in a top-down manner. It starts from the full space and ranks attributes according to their locality with respect to clusters. CLIQUE proceeds in a bottom-up manner.

### Soft projected clustering methods

PROCLUS and some projected clustering methods assign a “hard” subspace to each cluster. That is, an attribute either belongs to the subspace of the cluster or is excluded. Moreover, for all attributes included, their weights are the identical in distance calculation. However, in many applications, different attributes may carry different scale in units and different importance in similarity measurement. To reflect this intuition and provide the flexibility of normalization of attributes per cluster, **soft projected clustering** can be used.

The LAC (for *locally adaptive clustering*) method is a soft projected clustering approach. In LAC, for each attribute  $A_i$ , a weight  $w_i$  is introduced in the distance calculation. That is, Eq. (9.16) is revised to

$$|\mathbf{x} - \mathbf{y}|_p^w = \sqrt[p]{\sum_{i=1}^d w_i \cdot |\mathbf{x}[i] - \mathbf{y}[i]|^p}, \quad (9.17)$$



Two subspace clusters.

**FIGURE 9.4**

Clustering in arbitrarily oriented subspaces.

where  $\mathbf{w} = \langle w_1, \dots, w_d \rangle$  is the weight vector. Soft projected clustering can be computed using the EM framework similar in spirit as introduced in Section 9.1.3.

### 9.2.3 Arbitrarily oriented subspace approaches

As mentioned, when clusters do not align well with attributes in an embedding data space, the arbitrarily oriented subspace approaches are needed. Let us demonstrate the intuition using two clusters in Fig. 9.4. The figure shows the projections of a 3-D data set on two 2-D subspaces,  $X$ - $Y$  and  $X$ - $Z$ . We can see that the objects form two clusters,  $P$  and  $Q$ . However, those two clusters do not align with the attributes  $X$ ,  $Y$ , or  $Z$ . In order to find such clusters that are in subspaces arbitrarily oriented, a method has to construct subspaces according to the data on the fly.

ORCLUS is such a method. The general idea of ORCLUS is indeed quite similar to that in PROCLUS. In other words, ORCLUS also partitions the data objects into  $k$  clusters in a way similar to  $k$ -means and finds for each cluster the subspace. The critical difference is that instead of using a subset of attributes as the subspace for a cluster, ORCLUS constructs the subspace of a cluster based on the eigensystem of the objects assigned to the cluster and uses the weak eigenvectors to construct new attributes. Here, weak eigenvectors are used because on those dimensions, the objects assigned to the cluster are dense and thus indistinguishable, which meet the requirement of being clustered. The eigensystem is iteratively improved until clustering of good quality is achieved.

## 9.3 Biclustering

In the cluster analysis discussed so far, we cluster objects according to their attribute values. Objects and attributes are not treated in the same way. However, in some applications, objects and attributes are defined in a symmetric way, where data analysis involves searching data matrices for submatrices that show unique patterns as clusters. This kind of clustering technique belongs to the category of biclustering.

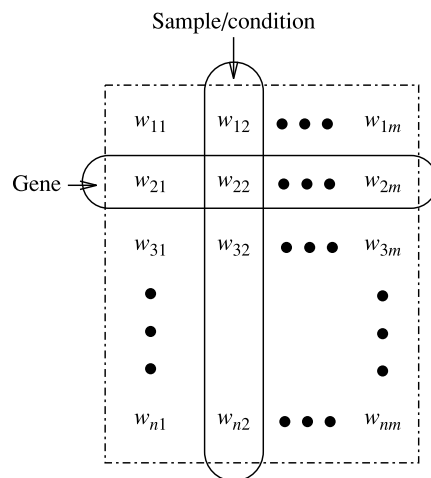
This section first introduces two motivating application examples of biclustering—gene expression and recommender systems (Section 9.3.1). You will then learn about the different types of biclusters (Section 9.3.2). Last, we present biclustering methods (Sections 9.3.3 and 9.3.4).

### 9.3.1 Why and where is biclustering useful?

Biclustering techniques were first proposed to address the needs for analyzing gene expression data. A *gene* is a unit of the passing-on of traits from a living organism to its offspring. Typically, a gene resides on a segment of DNA. Genes are critical for all living things because they specify all proteins and functional RNA chains. They hold the information to build and maintain a living organism's cells and pass genetic traits to offspring. Synthesis of a functional gene product, either RNA or protein, relies on the process of gene expression. A *genotype* is the genetic makeup of a cell, an organism, or an individual. *Phenotypes* are observable characteristics of an organism. *Gene expression* is the most fundamental level in genetics in that genotypes cause phenotypes.

Using *DNA chips* (also known as *DNA microarrays*) and other biological engineering techniques, we can measure the expression level of a large number (possibly all) of an organism's genes, in a number of different experimental conditions. Such conditions may correspond to different time points in an experiment or samples from different organs. Roughly speaking, the *gene expression data* or *DNA microarray data* are conceptually a gene-sample/condition matrix, where each row corresponds to one gene, and each column corresponds to one sample or condition. Each element in the matrix is a real number and records the expression level of a gene under a specific condition. Fig. 9.5 shows an illustration.

From the clustering viewpoint, an interesting issue is that a gene expression data matrix can be analyzed in two dimensions—the gene dimension and the sample/condition dimension.



**FIGURE 9.5**

Microarray data matrix.

- When analyzing in the *gene dimension*, we treat each gene as an object and treat the samples/conditions as attributes. By mining in the gene dimension, we may find patterns shared by multiple genes, or cluster genes into groups. For example, we may find a group of genes that express themselves similarly, which is highly interesting in bioinformatics, such as in finding pathways.
- When analyzing in the *sample/condition dimension*, we treat each sample/condition as an object and treat the genes as attributes. In this way, we may find patterns of samples/conditions, or cluster samples/conditions into groups. For example, we may find the differences in gene expression by comparing a group of tumor samples and nontumor samples.

**Example 9.10. Gene expression.** Gene expression matrices are popular in bioinformatics research and development. For example, an important task is to classify a new gene using the expression data of the gene and that of other genes in known classes. Symmetrically, we may classify a new sample (e.g., a new patient) using the expression data of the sample and that of samples in known classes (e.g., tumor and nontumor). Such tasks are invaluable in understanding the mechanisms of diseases and in clinical treatment. □

As can be seen, many gene expression data mining problems are highly related to cluster analysis. However, a challenge here is that, instead of clustering in one dimension (e.g., gene or sample/condition), in many cases we need to cluster in two dimensions simultaneously (e.g., both gene and sample/condition). Moreover, unlike the clustering models we have discussed so far, a cluster in a gene expression data matrix is a *submatrix* and usually has the following characteristics:

- Only a small set of genes participate in the cluster.
- The cluster involves only a small subset of samples/conditions.
- A gene may participate in multiple clusters, or may not participate in any cluster. This is especially helpful to handle noise, since we do not have to “force” noise points into a cluster.
- A sample/condition may be involved in multiple clusters, or may not be involved in any cluster.

To find clusters in gene-sample/condition matrices, we need new clustering techniques that meet the following requirements for *biclustering*:

- A cluster of genes is defined using only a subset of samples/conditions.
- A cluster of samples/conditions is defined using only a subset of genes.
- The clusters are neither *exclusive* (e.g., where one gene can participate in multiple clusters) nor *exhaustive* (e.g., where a gene may not participate in any cluster).

Biclustering is useful not only in bioinformatics, but also in other applications as well. Consider recommender systems as an example.

**Example 9.11. Using biclustering for a recommender system.** Imagine that an e-commerce company collects data from customers’ evaluations of products and uses the data to recommend products to customers. The data can be modeled as a customer-product matrix, where each row represents a customer, and each column represents a product. Each element in the matrix represents a customer’s evaluation of a product, which may be a score (e.g., like, like somewhat, not like) or purchase behavior (e.g., buy or not). Fig. 9.6 illustrates the structure.

The customer-product matrix can be analyzed in two dimensions: the *customer* dimension and the *product* dimension. Treating each customer as an object and products as attributes, the company can



|           |  | Products |          |          |          |
|-----------|--|----------|----------|----------|----------|
|           |  | $w_{11}$ | $w_{12}$ | $\cdots$ | $w_{1m}$ |
| Customers |  | $w_{21}$ | $w_{22}$ | $\cdots$ | $w_{2m}$ |
|           |  | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
|           |  | $w_{n1}$ | $w_{n2}$ | $\cdots$ | $w_{nm}$ |

FIGURE 9.6

---

Customer-product matrix.

find customer groups that have similar preferences or purchase patterns. Using products as objects and customers as attributes, the company can mine product groups that are similar in customer interest.

Moreover, the company can mine clusters in both customers and products simultaneously. Such a cluster contains a subset of customers and involves a subset of products. For example, the company may be highly interested in finding a group of customers who all like the same group of products. Such a cluster is a submatrix in the customer-product matrix, where all elements have a high value. Using such a cluster, the company can make recommendations in two directions. First, the company can recommend products to new customers who are similar to the customers in the cluster. Second, the company can recommend to customers new products that are similar to those involved in the cluster.

□

As with biclusters in a gene expression data matrix, the biclusters in a customer-product matrix usually have the following characteristics:

- Only a small set of customers participate in a cluster.
- A cluster involves only a small subset of products.
- A customer can participate in multiple clusters, or may not participate in any cluster.
- A product may be involved in multiple clusters, or may not be involved in any cluster.

Biclustering can be applied to customer-product matrices to mine clusters satisfying these requirements.

### 9.3.2 Types of biclusters

*“How can we model biclusters and mine them?”* Let us start with some basic notations. For the sake of simplicity, we will use “genes” and “conditions” to refer to the two dimensions in our discussion. Our discussion can easily be extended to other applications. For example, we can simply replace “genes” and “conditions” by “customers” and “products” to tackle the customer-product biclustering problem.

Let  $A = \{a_1, \dots, a_n\}$  be a set of genes and  $B = \{b_1, \dots, b_m\}$  be a set of conditions. Let  $\mathbf{E} = [e_{ij}]$  be a gene expression data matrix, that is, a gene-condition matrix, where  $1 \leq i \leq n$  and  $1 \leq j \leq m$ . A submatrix  $I \times J$  is defined by a subset  $I \subseteq A$  of genes and a subset  $J \subseteq B$  of conditions. For example, in the matrix shown in Fig. 9.7,  $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$  is a submatrix.

A bicluster is a submatrix where genes and conditions follow consistent patterns. We can define different types of biclusters based on such patterns.

- As the simplest case, a submatrix  $I \times J$  ( $I \subseteq A$ ,  $J \subseteq B$ ) is a **bicluster with constant values** if for any  $i \in I$  and  $j \in J$ ,  $e_{ij} = c$ , where  $c$  is a constant. For example, the submatrix  $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$  in Fig. 9.7 is a bicluster with constant values.

|          |     |       |     |          |     |          |     |          |     |
|----------|-----|-------|-----|----------|-----|----------|-----|----------|-----|
|          | ... | $b_6$ | ... | $b_{12}$ | ... | $b_{36}$ | ... | $b_{99}$ | ... |
| $a_1$    | ... | 60    | ... | 60       | ... | 60       | ... | 60       | ... |
| ...      | ... | ...   | ... | ...      | ... | ...      | ... | ...      | ... |
| $a_{33}$ | ... | 60    | ... | 60       | ... | 60       | ... | 60       | ... |
| ...      | ... | ...   | ... | ...      | ... | ...      | ... | ...      | ... |
| $a_{86}$ | ... | 60    | ... | 60       | ... | 60       | ... | 60       | ... |
| ...      | ... | ...   | ... | ...      | ... | ...      | ... | ...      | ... |

**FIGURE 9.7**

Gene-condition matrix, a submatrix, and a bicluster.

|    |    |    |    |    |
|----|----|----|----|----|
| 10 | 10 | 10 | 10 | 10 |
| 20 | 20 | 20 | 20 | 20 |
| 50 | 50 | 50 | 50 | 50 |
| 0  | 0  | 0  | 0  | 0  |

**FIGURE 9.8**

Bicluster with constant values on rows.

|    |    |    |     |    |
|----|----|----|-----|----|
| 10 | 50 | 30 | 70  | 20 |
| 20 | 60 | 40 | 80  | 30 |
| 50 | 90 | 70 | 110 | 60 |
| 0  | 40 | 20 | 60  | 10 |

**FIGURE 9.9**

Bicluster with coherent values.

- A bicluster is interesting if each row has a constant value, though different rows may have different values. A **bicluster with constant values on rows** is a submatrix  $I \times J$  such that for any  $i \in I$  and  $j \in J$ ,  $e_{ij} = c + \alpha_i$ , where  $\alpha_i$  is the adjustment for row  $i$ . For example, Fig. 9.8 shows a bicluster with constant values on rows.

Symmetrically, a **bicluster with constant values on columns** is a submatrix  $I \times J$  such that for any  $i \in I$  and  $j \in J$ ,  $e_{ij} = c + \beta_j$ , where  $\beta_j$  is the adjustment for column  $j$ .

- More generally, a bicluster is interesting if the rows change in a synchronized way with respect to the columns and vice versa. Mathematically, a **bicluster with coherent values** (also known as a **pattern-based cluster**) is a submatrix  $I \times J$  such that for any  $i \in I$  and  $j \in J$ ,  $e_{ij} = c + \alpha_i + \beta_j$ , where  $\alpha_i$  and  $\beta_j$  are the adjustment for row  $i$  and column  $j$ , respectively. For example, Fig. 9.9 shows a bicluster with coherent values.

It can be shown that  $I \times J$  is a bicluster with coherent values if and only if for any  $i_1, i_2 \in I$  and  $j_1, j_2 \in J$ , then  $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$ . Moreover, instead of using addition, we can define a bicluster with coherent values using multiplication, that is,  $e_{ij} = c \cdot (\alpha_i \cdot \beta_j)$ . Clearly, biclusters with constant values on rows or columns are special cases of biclusters with coherent values.

- In some applications, we may only be interested in the up- or down-regulated changes across genes or conditions without constraining the exact values. A **bicluster with coherent evolutions on rows** is a submatrix  $I \times J$  such that for any  $i_1, i_2 \in I$  and  $j_1, j_2 \in J$ ,  $(e_{i_1 j_1} - e_{i_1 j_2})(e_{i_2 j_1} - e_{i_2 j_2}) \geq 0$ . For

|    |     |    |      |    |
|----|-----|----|------|----|
| 10 | 50  | 30 | 70   | 20 |
| 20 | 100 | 50 | 1000 | 30 |
| 50 | 100 | 90 | 120  | 80 |
| 0  | 80  | 20 | 100  | 10 |

**FIGURE 9.10**

Bicluster with coherent evolutions on rows.

example, Fig. 9.10 shows a bicluster with coherent evolutions on rows. Symmetrically, we can define biclusters with coherent evolutions on columns.

Next, we study how to mine biclusters.

### 9.3.3 Biclustering methods

The previous specification of the types of biclusters only considers ideal cases. In real data sets, such perfect biclusters rarely exist. When they do exist, they are usually very small. Instead, random noise can affect the readings of  $e_{ij}$  and thus prevent a bicluster in nature from appearing in a perfect shape.

There are two major types of methods for discovering biclusters in data that may come with noise. **Optimization-based methods** conduct an iterative search. At each iteration, the submatrix with the highest significance score is identified as a bicluster. The process terminates when a user-specified condition is met. Due to cost concerns in computation, greedy search is often employed to find local optimal biclusters. **Enumeration methods** use a tolerance threshold to specify the degree of noise allowed in the biclusters to be mined and then try to enumerate all submatrices of biclusters that satisfy the requirements. We use the  $\delta$ -Cluster and MaPle algorithms as examples to illustrate these ideas.

#### *Optimization using the $\delta$ -cluster algorithm*

For a submatrix,  $I \times J$ , the mean of the  $i$ th row is

$$e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{ij}. \quad (9.18)$$

Symmetrically, the mean of the  $j$ th column is

$$e_{Ij} = \frac{1}{|I|} \sum_{i \in I} e_{ij}. \quad (9.19)$$

The mean of all elements in the submatrix is

$$e_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} e_{ij} = \frac{1}{|I|} \sum_{i \in I} e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{Ij}. \quad (9.20)$$

The quality of the submatrix as a bicluster can be measured by the *mean-squared residue* value as

$$H(I \times J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \quad (9.21)$$

Submatrix  $I \times J$  is a  $\delta$ -**bicluster** if  $H(I \times J) \leq \delta$ , where  $\delta \geq 0$  is a threshold. When  $\delta = 0$ ,  $I \times J$  is a perfect bicluster with coherent values. By setting  $\delta > 0$ , a user can specify the tolerance of average noise per element against a perfect bicluster, because in Eq. (9.21) the residue on each element is

$$\text{residue}(e_{ij}) = e_{ij} - e_{iJ} - e_{Ij} + e_{IJ}. \quad (9.22)$$

A *maximal  $\delta$ -bicluster* is a  $\delta$ -bicluster  $I \times J$  such that there does not exist another  $\delta$ -bicluster  $I' \times J'$ , and  $I \subseteq I'$ ,  $J \subseteq J'$ , and  $I = I'$  and  $J = J'$  do not hold simultaneously. Finding the maximal  $\delta$ -bicluster of the largest size is computationally costly. Therefore we can use a heuristic greedy search method to obtain a local optimal cluster. The algorithm works in two phases.

- In the *deletion phase*, we start from the whole matrix. While the mean-squared residue of the matrix is over  $\delta$ , we iteratively remove rows and columns. At each iteration, for each row  $i$ , we compute the *mean-squared residue* as

$$d(i) = \frac{1}{|J|} \sum_{j \in J} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \quad (9.23)$$

Moreover, for each column  $j$ , we compute the *mean-squared residue* as

$$d(j) = \frac{1}{|I|} \sum_{i \in I} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \quad (9.24)$$

We remove the row or column of the largest mean-squared residue. At the end of this phase, we obtain a submatrix  $I \times J$  that is a  $\delta$ -bicluster. However, the submatrix may not be maximal.

- In the *addition phase*, we iteratively expand the  $\delta$ -bicluster  $I \times J$  obtained in the deletion phase as long as the  $\delta$ -bicluster requirement is maintained. At each iteration, we consider rows and columns that are not involved in the current bicluster  $I \times J$  by calculating their mean-squared residues. A row or column of the smallest mean-squared residue is added into the current  $\delta$ -bicluster.

This greedy algorithm can find one  $\delta$ -bicluster only. To find multiple biclusters that do not have heavy overlaps, we can run the algorithm multiple times. After each execution where a  $\delta$ -bicluster is output, we can replace the elements in the output bicluster by random numbers. Although the greedy algorithm may find neither the optimal biclusters nor all biclusters, it is very fast even on large matrices.

### 9.3.4 Enumerating all biclusters using MaPle

As mentioned, a submatrix  $I \times J$  is a bicluster with coherent values if and only if for any  $i_1, i_2 \in I$  and  $j_1, j_2 \in J$ ,  $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$ . For any  $2 \times 2$  submatrix of  $I \times J$ , we can define a *p-score* as

$$p\text{-score} \begin{pmatrix} e_{i_1 j_1} & e_{i_1 j_2} \\ e_{i_2 j_1} & e_{i_2 j_2} \end{pmatrix} = |(e_{i_1 j_1} - e_{i_2 j_1}) - (e_{i_1 j_2} - e_{i_2 j_2})|. \quad (9.25)$$

A submatrix  $I \times J$  is a  $\delta$ -**pCluster** (for *pattern-based cluster*) if the *p-score* of every  $2 \times 2$  submatrix of  $I \times J$  is at most  $\delta$ , where  $\delta \geq 0$  is a threshold specifying a user's tolerance of noise against a perfect bicluster. Here, the *p-score* controls the noise on every element in a bicluster, whereas the mean-squared residue captures the average noise.

An interesting property of  $\delta$ -pCluster is that if  $I \times J$  is a  $\delta$ -pCluster, then every  $x \times y$  ( $x, y \geq 2$ ) submatrix of  $I \times J$  is also a  $\delta$ -pCluster. This monotonicity enables us to obtain a succinct representation of nonredundant  $\delta$ -pClusters. A  $\delta$ -pCluster is maximal if no more rows or columns can be added into the cluster while maintaining the  $\delta$ -pCluster property. To avoid redundancy, instead of finding all  $\delta$ -pClusters, we only need to compute all maximal  $\delta$ -pClusters.

**MaPle** is an algorithm that enumerates all maximal  $\delta$ -pClusters. It systematically enumerates every combination of conditions using a set enumeration tree and a depth-first search. This enumeration framework is the same as the pattern-growth methods for frequent pattern mining (Chapter 4). Consider gene expression data. For each condition combination,  $J$ , MaPle finds the maximal subsets of genes,  $I$ , such that  $I \times J$  is a  $\delta$ -pCluster. If  $I \times J$  is not a submatrix of another  $\delta$ -pCluster, then  $I \times J$  is a maximal  $\delta$ -pCluster.

There may be a huge number of condition combinations. MaPle prunes many unfruitful combinations using the monotonicity of  $\delta$ -pClusters. For a condition combination,  $J$ , if there does not exist a set of genes,  $I$ , such that  $I \times J$  is a  $\delta$ -pCluster, then we do not need to consider any superset of  $J$ . Moreover, we should consider  $I \times J$  as a candidate of a  $\delta$ -pCluster only if for every  $(|J| - 1)$ -subset  $J'$  of  $J$ ,  $I \times J'$  is a  $\delta$ -pCluster. MaPle also employs several pruning techniques to speed up the search while retaining the completeness of returning all maximal  $\delta$ -pClusters. For example, when examining a current  $\delta$ -pCluster,  $I \times J$ , MaPle collects all the genes and conditions that may be added to expand the cluster. If these candidate genes and conditions together with  $I$  and  $J$  form a submatrix of a  $\delta$ -pCluster that has already been found, then the search of  $I \times J$  and any superset of  $J$  can be pruned. Interested readers may refer to the bibliographic notes for additional information on the MaPle algorithm.

An interesting observation here is that the search for maximal  $\delta$ -pClusters in MaPle is somewhat similar to mining frequent closed itemsets. Consequently, MaPle borrows the depth-first search framework and ideas from the pruning techniques of pattern-growth methods for frequent pattern mining. This is an example where frequent pattern mining and cluster analysis may share similar techniques and ideas.

An advantage of MaPle and the other algorithms that enumerate all biclusters is that they guarantee the completeness of the results and do not miss any overlapping biclusters. However, a challenge for such enumeration algorithms is that they may become very time consuming if a matrix becomes very large, such as a customer-purchase matrix of hundreds of thousands of customers and millions of products.

---

## 9.4 Dimensionality reduction for clustering

Subspace clustering methods try to find clusters in subspaces of the original data space. In some situations, it is more effective to construct a new space instead of using subspaces of the original data. This is the motivation behind dimensionality reduction methods for clustering high-dimensional data. This section explores dimensionality reduction for clustering.

We start from the traditional linear dimensionality reduction methods and use principal component analysis (PCA) as an example. Then, we look at the general dimensionality and matrix decomposition methods. We use nonnegative matrix factorization (NMF) methods as an example and the general framework, and explain the relation between NMF and the traditional  $k$ -means clustering. Last, we

change the angle and discuss spectral clustering, which rebuilds a new feature space based on the similarity graph and conduct clustering.

### 9.4.1 Linear dimensionality reduction methods for clustering

In many applications, the challenges in clustering analysis on high dimensional data come from two obstacles in data sets.

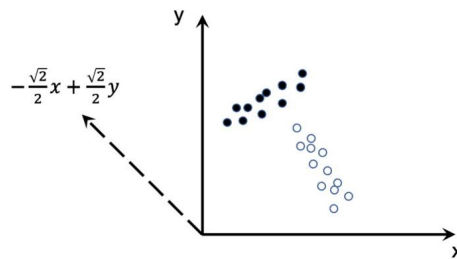
First, when a high-dimensional data set is collected, attributes may be correlated. For example, if we use three cameras to detect the spatial positions of objects, that is, the locations of objects are projected into three 2-D spaces. If each camera captures the 2-D coordinates of an object, then each object is recorded in a 6-D space. As the objects are in a 3-D space, the underlying location information should be described sufficiently using three intrinsic dimensions. Some correlation and thus redundancy may exist among the data collected using the three cameras. In other words, although a data set may have many dimensions, the underlying structures and relations may have a substantially lower dimensionality, which are often deeply hidden.

Second, observations on different attributes may be recorded using different scales and are not normalized properly. For example, different cameras may record the coordinates using different units, some may measure in metric and the others may measure in imperial. Moreover, those three cameras may not be set up orthogonally, two may be close to each other and one may be far away. In other words, the data recorded may be stretched.

**Dimensionality reduction**, or dimension reduction, transforms a high-dimensional data set into a low-dimensional space so that the low-dimensional representation retains meaningful properties of the original data, ideally approaching the intrinsic dimensions of the underlying structures. To understand the intuition, let us consider the following example.

**Example 9.12. Clustering in a derived space.** Consider the two clusters of points in Fig. 9.11. It is not possible to cluster these points in any subspace of the original space,  $X \times Y$ , because both clusters would end up being projected onto overlapping areas in the  $x$  and  $y$  axes.

What if, instead, we construct a new dimension,  $-\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y$  (shown as a dashed line in the figure)? By projecting the points onto this new dimension, the two clusters become apparent.  $\square$



**FIGURE 9.11**

Clustering in a derived space may be more effective.

Although Example 9.12 involves only two dimensions, the idea of constructing a new space (so that any clustering structure hidden in the data becomes well manifested) can be extended to high-dimensional data. Preferably, the newly constructed space should have low dimensionality.

There are many dimensionality reduction methods. **Principal component analysis (PCA)** is frequently used to identify the most meaningful basis to re-express a data set. Before conducting clustering analysis, one may first apply PCA to reduce the dimensionality of a data set. Applying PCA can help to remove or reduce the correlation among attributes and normalize the attributes.

Let us take a look at the intuition behind PCA. Suppose we have  $n$  data objects, each of  $m$  dimensions. To represent the original data set, we can write an  $n \times m$  matrix  $\mathbf{X}$ , where each row represents an object, and each column represents a dimension. In general, a **linear transformation** tries to find an  $m \times m$  matrix  $\mathbf{P}$  such  $\mathbf{XP} = \mathbf{Y}$ , where  $\mathbf{Y}$  is another  $m \times n$  matrix. Intuitively, matrix  $\mathbf{P}$  is a rotation and a stretch that transforms  $\mathbf{X}$  to  $\mathbf{Y}$ , and the columns of  $\mathbf{P}$  are the set of new basis vectors re-expressing the rows of  $\mathbf{X}$ . To further illustrate the intuition, let  $\mathbf{x}_1, \dots, \mathbf{x}_n$  be the rows of  $\mathbf{X}$ , that is, the row vectors of the objects, and  $\mathbf{p}_1, \dots, \mathbf{p}_m$  be the columns of  $\mathbf{P}$ . Then, we have

$$\mathbf{XP} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_m \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \mathbf{p}_1 & \cdots & \mathbf{x}_1 \mathbf{p}_m \\ \vdots & \ddots & \vdots \\ \mathbf{x}_n \mathbf{p}_1 & \cdots & \mathbf{x}_n \mathbf{p}_m \end{bmatrix} = \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_n \end{bmatrix},$$

where  $\mathbf{y}_1, \dots, \mathbf{y}_n$  are the rows of  $\mathbf{Y}$ . Then, we have

$$\mathbf{y}_i = [\mathbf{x}_i \mathbf{p}_1 \quad \cdots \quad \mathbf{x}_i \mathbf{p}_m].$$

As can be seen, each data object  $\mathbf{x}_i$  is transformed to  $\mathbf{y}_i$  by a dot-product with the corresponding column of  $\mathbf{P}$ . In other words,  $\mathbf{y}_i$  is the projection of  $\mathbf{x}_i$  on to the basis of  $\mathbf{p}_1, \dots, \mathbf{p}_m$ . Through this transformation, the data set  $\mathbf{X}$  is re-expressed as  $\mathbf{Y}$  using the new basis of  $\mathbf{p}_1, \dots, \mathbf{p}_m$ .

In a nutshell, PCA is a method to choose a good base  $\mathbf{P}$  that best re-express  $\mathbf{X}$  into  $\mathbf{Y}$  so that signals are maximized and noise is minimized. Let us measure the correlation between two attributes  $A$  and  $B$  can be measured by the covariance  $\sigma_{AB}^2 = \frac{1}{n} \sum_{i=1}^n a_i b_i$ , the larger the more correlation. To measure the correlation between every two dimensions in  $\mathbf{X}$ , PCA computes the covariance between every pair of dimensions by constructing the covariance matrix  $\mathbf{C}_X = \frac{1}{n} \mathbf{X}^T \mathbf{X}$ . The covariance matrix  $\mathbf{C}_X$  reflects the noise and redundancy in the original representation  $\mathbf{X}$ .

By transforming  $\mathbf{X}$  to  $\mathbf{Y}$ , PCA tries to remove or reduce redundancy caused by correlation among dimensions in the original data space. Thus the covariance matrix  $\mathbf{C}_Y$  of  $\mathbf{Y}$  should have two properties. First, all off-diagonal terms in  $\mathbf{C}_Y$  should be zero, that is, the dimensions in the new basis are not correlated with each other. Second, the dimensions in  $\mathbf{Y}$  should be ordered by variance, the larger the variance, the more signals the dimension carries, and thus the more important the dimension.

We can build the connection between the covariance matrices  $\mathbf{C}_X$  and  $\mathbf{C}_Y$  as follows.

$$\begin{aligned} \mathbf{C}_Y &= \frac{1}{n} \mathbf{Y}^T \mathbf{Y} = \frac{1}{n} (\mathbf{XP})^T (\mathbf{XP}) = \frac{1}{n} \mathbf{P}^T \mathbf{X}^T \mathbf{X} \mathbf{P} = \mathbf{P}^T \left( \frac{1}{n} \mathbf{X}^T \mathbf{X} \right) \mathbf{P} \\ &= \mathbf{P}^T \mathbf{C}_X \mathbf{P}. \end{aligned} \tag{9.26}$$

From linear algebra, we know that every symmetric matrix is diagonalized by an orthogonal matrix of its eigenvectors. The covariance matrix  $\mathbf{C}_X$  is obviously symmetric. The matrix  $\mathbf{P}$  where each column

$\mathbf{p}_i$  is an eigenvector of  $\mathbf{C}_X = \frac{1}{n} \mathbf{X}^T \mathbf{X}$  is exactly the transformation matrix serving our purpose. The eigenvectors are ranked in the eigenvalue descending order.

To summarize the above rationale, computing PCA in practice of a data set  $\mathbf{X}$  can be conducted in three steps.

- 1. Normalize  $\mathbf{X}$ .** For each dimension in  $\mathbf{X}$ , we calculate the mean of each column, subtract off the mean of every observation on the dimension and normalize by the standard deviation. That is, entry  $x_{ij}$  is normalized to  $\frac{x_{ij} - \mu_j}{\sigma_j}$ , where  $\mu_j$  and  $\sigma_j$  are the mean and the standard deviation of the  $j$ th column, respectively. For the sake of simplicity, let us still denote by  $\mathbf{X}$  the normalized matrix.
- 2. Compute the eigenvectors of the covariance matrix  $\mathbf{C}_X$ .** The eigenvectors form the new basis.
- 3. Choose the top- $k$  eigenvectors and transform the original data in the new space of reduced dimensionality.** The eigenvalues reflect the variances on the corresponding eigenvectors. We can choose the top- $k$  eigenvectors as the new basis and reduce the dimensionality so that the cumulated eigenvalue dominates the sum of eigenvalues.

**Example 9.13.** Consider a 3-D data set that has four objects:

$$\mathbf{X} = \begin{bmatrix} 5 & 9 & 3 \\ 4 & 10 & 6 \\ 3 & 8 & 11 \\ 6 & 3 & 7 \end{bmatrix}.$$

After normalization, the matrix is

$$\mathbf{X} = \begin{bmatrix} 0.387 & 0.482 & -1.135 \\ -0.387 & 0.804 & -0.227 \\ -1.162 & 0.161 & 1.286 \\ 1.162 & -1.447 & 0.076 \end{bmatrix}.$$

The covariance matrix is

$$\mathbf{C}_X = \begin{bmatrix} 0.750 & -0.411 & -0.439 \\ -0.411 & 0.549 & -0.152 \\ -0.439 & -0.152 & 0.750 \end{bmatrix}.$$

The three eigenvectors are  $[-1.339, 0.567, 1]^T$  with eigenvalue 1.252,  $[0.282, -1.098, 1]^T$  with eigenvalue 0.793, and  $[1.270, 1.237, 1]^T$  with eigenvalue 0.004. In other words, the transformation matrix is

$$\mathbf{P} = \begin{bmatrix} -1.339 & 0.282 & 1.270 \\ -0.567 & -1.098 & 1.237 \\ 1 & 1 & 1 \end{bmatrix}.$$

Correspondingly,  $\mathbf{X}$  is transformed to

$$\mathbf{Y} = \mathbf{X}\mathbf{P} = \begin{bmatrix} -8.798 & -5.472 & 20.483 \\ -5.026 & -3.852 & 23.45 \\ 2.447 & 3.062 & 24.706 \\ -2.735 & 5.398 & 18.331 \end{bmatrix}.$$



Since the last eigenvalue 0.004 is very small comparing to the first two eigenvalues, we can reduce the dimensionality from 3 to 2 by dropping the last dimension in  $\mathbf{Y}$ . That is, the data set can be represented in the 2-D space using the first two eigenvectors as the basis. The representation is

$$\begin{bmatrix} -8.798 & -5.472 \\ -5.026 & -3.852 \\ 2.447 & 3.062 \\ -2.735 & 5.398 \end{bmatrix}.$$

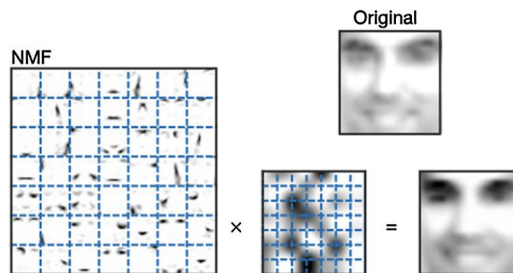
□

To recap, the core idea of PCA is to assume that the variance along a small number of principal components can provide a good characterization of a high-dimensional data set. PCA is parameter-free. One can apply PCA on any numeric data sets. On the one hand, this is an advantage since it is easy to use and thus is employed in many applications. On the other hand, the agnostic nature of PCA to data sources also comes as a weakness, since it cannot take advantage of any background knowledge to conduct dimensionality reduction.

### 9.4.2 Nonnegative matrix factorization (NMF)

*What is the intuitive idea behind the popularly used nonnegative matrix factorization methods?* We know that a data set  $\mathbf{X}$  of  $n$  objects in an  $m$ -dimensional space can be represented using an  $n \times m$  matrix, where each row represents an object and each column represents a dimension. Intuitively, the problem of clustering the objects in  $\mathbf{X}$  into  $k$  clusters can be modeled as factorizing  $\mathbf{X}$  into two matrices  $\mathbf{W}$  and  $\mathbf{H}$  such that  $\mathbf{X} \approx \mathbf{HW}$ , where  $\mathbf{H}$  is an  $n \times k$  matrix representing how the  $n$  objects are assigned to the  $k$  clusters, and  $\mathbf{W}$  is a  $k \times m$  matrix and each row in  $\mathbf{W}$  represents the “center” of a cluster. An object may be similar to more than one cluster. Therefore the entries in  $\mathbf{H}$  are nonnegative. Here, we do not want to have a negative entry in  $\mathbf{H}$ , since a negative weight between an object and a cluster is hard to explain.

For example, consider a database of 2429 facial images, each consisting of  $19 \times 19$  pixels. A  $2429 \times 361$  matrix is constructed. Fig. 9.12 shows the  $7 \times 7$  montages. The example face shown at top right



**FIGURE 9.12**

An example of using NMF in clustering images. Extracted from D. D. Lee and H. S. Seung, *Nature* Vol. 401, October 21, 1999.

is approximately represented by a linear superposition of some basic images in an additive manner learned by NMF. The weights are shown in the  $7 \times 7$  grid, where the positive values are shown with black pixels. The resulting superpositions are shown on the other side of the equality sign.

Formally, assume an  $n \times m$  nonnegative matrix  $\mathbf{X}$  containing a set of  $n$  data row vectors. We factorize  $\mathbf{X}$  into two matrices

$$\mathbf{X} \approx \mathbf{H}\mathbf{W} \quad (9.27)$$

where  $\mathbf{X} \in \mathcal{R}^{n \times m}$ ,  $\mathbf{H} \in \mathcal{R}^{n \times k}$ ,  $\mathbf{W} \in \mathcal{R}^{k \times m}$ , and matrices  $\mathbf{X} \geq 0$ ,  $\mathbf{H} \geq 0$ , and  $\mathbf{W} \geq 0$  are all nonnegative. Generally, the rank of matrices  $\mathbf{H}$  and  $\mathbf{W}$  is much lower than the rank of  $\mathbf{X}$ , that is,  $k \ll \min\{m, n\}$ .

There may exist many different matrices satisfying Eq. (9.27) given  $\mathbf{X}$ . **Nonnegative matrix factorization** (NMF) minimizes an objective cost function that measures the quality of factorization, that is, how well the factorization represents the original data. The most common cost function is the sum of squared errors, that is,

$$J_{SSE} = |\mathbf{X} - \mathbf{H}\mathbf{W}|^2 = \sum_{1 \leq i \leq n, 1 \leq j \leq m} |\mathbf{x}_{ij} - \sum_{l=1}^k \mathbf{h}_{il} \mathbf{w}_{lj}|^2,$$

where  $\mathbf{x}_{ij}$ ,  $\mathbf{h}_{ij}$ , and  $\mathbf{w}_{ij}$  are the entry at the  $i$  row and the  $j$ th column of matrices  $\mathbf{X}$ ,  $\mathbf{H}$ , and  $\mathbf{W}$ , respectively. The smaller  $J_{sse}$ , the better  $\mathbf{H}\mathbf{W}$  approximates  $\mathbf{X}$ .

Another cost function often used is the information divergence or Kullback-Leibler information number<sup>2</sup>

$$J_{ID} = \sum_{1 \leq i \leq n, 1 \leq j \leq m} [\mathbf{x}_{ij} \log \frac{\mathbf{x}_{ij}}{(\mathbf{H}\mathbf{W})_{ij}} - \mathbf{x}_{ij} + (\mathbf{H}\mathbf{W})_{ij}].$$

NMF has close connections with many clustering problems we discussed before. For example, if we constrain that  $\mathbf{H}^T \mathbf{H} = \mathbf{I}$ , that is, the assignments to clusters are orthogonal, then NMF is mathematically equivalent to  $k$ -means clustering. To understand the rationale, let  $\mathbf{c}_1, \dots, \mathbf{c}_k$  be the centers of the  $k$  clusters. Since  $\mathbf{H}$  is the indicators of how data objects are assigned to clusters,  $\mathbf{h}_{il} = 1$  if data object  $\mathbf{x}_i$  belongs to cluster  $\mathbf{c}_l$ ;  $\mathbf{h}_{il} = 0$  otherwise. The  $k$ -means clustering objective function is

$$J = \sum_{i=1}^n \sum_{l=1}^k \mathbf{h}_{il} |\mathbf{x}_i - \mathbf{c}_l|^2 = |\mathbf{X} - \mathbf{H}\mathbf{W}|^2.$$

By composing different constraints and objective cost functions, NMF can be shown equivalent to some other kinds of clustering problems, such as probabilistic latent semantic indexing and kernel  $k$ -means.

*How can we compute the factorization in NMF?* Unfortunately, unlike PCA, NMF does not allow exact solution in efficient time. Here, we introduce Lee and Seung's *multiplicative update rule* approach, which is simple in implementation. The algorithm works as follows.

1. We randomly initialize  $\mathbf{W}$  and  $\mathbf{H}$  as  $k \times m$  and  $n \times k$  nonnegative matrices, respectively.

<sup>2</sup> In calculating  $I$ -divergence, we adopt the conventions  $\frac{0}{0} = 0$ ,  $0 \log 0 = 0$ , and  $\frac{v}{0} = \infty$  for  $v > 0$ .

2. At the  $l$ th iteration, we update the elements in  $\mathbf{W}$  and  $\mathbf{H}$  by

$$\mathbf{h}_{ij}^{l+1} = \mathbf{h}_{ij}^l \frac{(\mathbf{X}(\mathbf{W}^l)^T)_{ij}}{(\mathbf{H}^l(\mathbf{W}^l)^T \mathbf{W}^l)_{ij}}$$

and

$$\mathbf{w}_{ij}^{l+1} = \mathbf{w}_{ij}^l \frac{(\mathbf{X}(\mathbf{H}^{l+1})^T)_{ij}}{((\mathbf{H}^{l+1})^T \mathbf{H}^{l+1} \mathbf{W}^l)_{ij}}.$$

3. Repeat Step 2 until  $\mathbf{W}$  and  $\mathbf{H}$  are stable.

In a nutshell, this is an EM method (Section 9.1.3). In each iteration, we first use the current clusters  $\mathbf{W}$  to update the membership assignment matrix  $\mathbf{H}$ , which is the E-step. Then, we update the cluster centers in matrix  $\mathbf{M}$ , which is the M-step.

NMF provides a flexible and powerful tool to express a wide spectrum of clustering problems by using different objective functions and composing different constraints. At the same time, the flexibility and expression power come with a cost. There are a series of challenges in using NMF in practice. First, NMF results may often be sensitive to initialization. Many methods have been proposed for initialization in NMF. For example, we may use the clustering results of some simple methods as the initialization of NMF. We can also use different random initializations and select the best estimates from multiple runs. Second, setting an appropriate stopping criterion for NMF is far from trivial. In practice we can set a maximum number of iterations or a maximum amount of runtime. We can also use some heuristics, such as setting a threshold on the objective cost function or the improvements. Third, scalability is often a concern when NMF is applied on a large and high-dimensional data set.

### 9.4.3 Spectral clustering

*You may hear that spectral clustering methods have been used in many applications. What is the major idea behind spectral clustering?* As discussed in Section 9.2.1, due to the curse of dimensionality, measuring pairwise distance in the original data space of a high-dimensional data set may not be reliable for clustering analysis. Fig. 9.13 illustrates the intuition. The figure shows two clusters of points, one in white and the other in black. Points  $a$  and  $b$  belong to two different clusters. However, in the original data space, the distance between  $a$  and  $b$  is shorter than that between  $a$  and some other points within the white cluster, such as  $c$ . Therefore if we rely on the pairwise distances in the original data space to construct clusters directly, the results may not be meaningful.

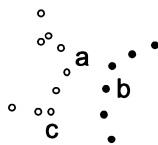
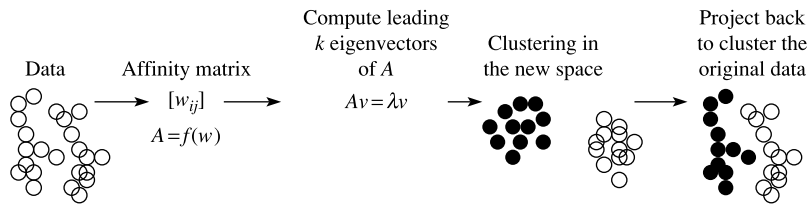


FIGURE 9.13

Pairwise distances in the original data space may not be reliable in clustering.



**FIGURE 9.14**

The framework of spectral clustering approaches. *Source:* Adapted from Slide 8 at [http://videlectures.net/micued08\\_azran\\_mcl/](http://videlectures.net/micued08_azran_mcl/).

To tackle the curse of dimensionality, spectral clustering relies on the intuition that, when the curse of dimensionality is present, the close neighborhood relation is more reliable than pairwise distances among all possible pairs of data objects. Spectral clustering conducts in three steps. First, spectral clustering transforms the original data set of objects into a similarity graph, where each object is a node and the edges connect close neighbors. Then, spectral clustering tries to embed the data points in a space where the clusters stand out. Last, a classical clustering algorithm, such as  $k$ -means, can be used to extract the clusters. Fig. 9.14 shows the general framework for spectral clustering approaches.

Let us explain the ideas of spectral clustering step by step and use the Ng-Jordan-Weiss algorithm as an example.

### Similarity graph

Spectral clustering connects objects and builds clusters by identifying objects that are close neighbors with each other. In other words, only when two objects are close to each other, they are regarded similar. This is implemented by constructing a similarity graph. Mathematically, let  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  be a set of objects in an  $m$ -dimensional space, that is, each data object  $\mathbf{x}_i$  ( $1 \leq i \leq n$ ) is an  $m$ -dimensional row vector. We assume a distance measure  $d(\mathbf{x}_i, \mathbf{x}_j) \geq 0$  between all pairs of objects  $\mathbf{x}_i$  and  $\mathbf{x}_j$  ( $1 \leq i, j \leq n$ ). As the first step, spectral clustering forms a similarity graph  $G = (V, E)$ , where each object  $\mathbf{x}_i$  becomes a vertex. There are three typical ways to form edges in the similarity graph.

- The  $\epsilon$ -neighborhood graph connects all points whose pairwise distances are smaller than or equal to  $\epsilon$ , where  $\epsilon$  is a parameter. Since  $\epsilon$  is often a small number, all the edges are treated as unweighted.
- The nearest neighbor graphs connect every vertex with its  $k$  nearest neighbors. Please note that the  $k$ -nearest neighbor relationship is not symmetric; that is, it is possible that  $\mathbf{x}_i$  is one of the  $k$  nearest neighbors of  $\mathbf{x}_j$ , but  $\mathbf{x}_j$  is not one of the  $k$  nearest neighbors of  $\mathbf{x}_i$ . To make the nearest neighbor graphs undirected and unweighted, there are several ways. For example, the  $k$ -nearest neighbor graph simply connects  $\mathbf{x}_i$  and  $\mathbf{x}_j$  with an unweighted edge if either  $\mathbf{x}_i$  is one of the  $k$  nearest neighbors of  $\mathbf{x}_j$  or the other way. As an alternation, the mutual  $k$ -nearest neighbor graph connects  $\mathbf{x}_i$  and  $\mathbf{x}_j$  with an unweighted edge if  $\mathbf{x}_i$  is one of the  $k$  nearest neighbors of  $\mathbf{x}_j$  and vice versa.
- The fully connected graph derives a new similarity function such that only pairs of nodes in close neighborhoods have positive similarity values. For example, the Gaussian similarity function

$$\text{sim}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{d(\mathbf{x}_i, \mathbf{x}_j)^2}{2\sigma^2}}$$

serves the purpose, where  $\sigma$  is a parameter controlling the width of the neighborhoods, playing a role similar to the parameter  $\epsilon$  in the  $\epsilon$ -neighborhood graph approach.

Denote by  $W$  the similarity graph. For vertices  $\mathbf{x}_i$  and  $\mathbf{x}_j$ , set  $w_{ij} > 0$  if they are connected in the similarity graph; 0 otherwise. Moreover,  $d_i = \sum_{j=1}^n w_{ij}$  is the degree of vertex  $\mathbf{x}_i$ . The *degree matrix*  $D$  is the diagonal matrix with the degrees  $d_1, \dots, d_n$  on the diagonal.

The Ng-Jordan-Weiss algorithm uses the distance measure and constructs a fully connected graph. Particularly, it calculates a similarity matrix  $\mathbf{W}$  such that

$$w_{ij} = e^{-\frac{\text{dist}(o_i, o_j)^2}{2\sigma^2}},$$

where  $\sigma$  is a scaling parameter. In the Ng-Jordan-Weiss algorithm,  $W_{ii}$  is set to 0.

### Finding a new space

After we transform a set of objects into a similarity graph, where objects are connected by neighbor relationship, intuitively, we can explore how flow may be propagated among vertices in the similarity graph. The sources are the centers of the clusters. Following this intuition, the gradient describes the direction of flow, and the divergence of the gradient provides a quantity of the vector field's source at each point. Mathematically, the Laplace operator<sup>3</sup> is what we need, which is a differential operator computing the divergence of the gradient of a function on Euclidean space.

There are two common ways to apply the Laplace operator on the similarity graph. The most commonly used approach is the *unnormalized graph Laplacian*, which is defined by  $L = D - W$ . Alternatively, some methods use the *normalized graph Laplacian*, which is defined by  $L = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ .

In order to embed the nodes in the similarity graph in a space where the clusters stand out, we can extract the first  $k$  eigenvectors of the Laplacian matrix, that is, the  $k$  eigenvectors of the smallest eigenvalues for unnormalized graph Laplacian or the  $k$  eigenvectors of the largest eigenvalues for normalized graph Laplacian. The nodes collapse into clusters in the space represented using those  $k$  eigenvectors as the base.

The Ng-Jordan-Weiss algorithm employs the normalized graph Laplacian, and uses the first  $k$  eigenvectors. It stacks the first  $k$  eigenvectors  $\mathbf{u}_1, \dots, \mathbf{u}_k$  in columns to form a matrix  $\mathbf{U} \in \mathbb{R}^{n \times k}$  and normalizes the rows of  $\mathbf{U}$  to norm 1 and form matrix  $\mathbf{T}$ , that is,  $t_{ij} = \frac{u_{ij}}{\sqrt{\sum_{l=1}^k u_{il}^2}}$ . Let  $\mathbf{y}_i$  be the  $i$ th row of

$\mathbf{T}$ , which is the embedding of object  $\mathbf{x}_i$  in the new space.

### Extracting clusters

We can employ a classical clustering algorithm, such as  $k$ -means, to extract the clusters in the new space formed by the first  $k$  eigenvectors in the previous step.

The Ng-Jordan-Weiss algorithm applies  $k$ -means on  $\mathbf{y}_1, \dots, \mathbf{y}_n$ , the row vectors in matrix  $\mathbf{Y}$ , to form clusters  $C_1, \dots, C_k$ . Then, it assigns the original data points to clusters according to how the

<sup>3</sup> If  $f$  is a twice-differentiable real-valued function, then the Laplacian of  $f$  is  $\Delta f = \nabla^2 f = \nabla \cdot \nabla f$ , where  $\nabla f = (\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n})$ . Equivalently,  $\Delta f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}$ .

transformed points are assigned in the clusters obtained in clustering  $\mathbf{Y}$ . In other words, the original object  $o_i$  is assigned to the  $j$ th cluster if and only if the row  $i$  of matrix  $\mathbf{Y}$  is assigned to the  $j$ th cluster  $C_j$ .

In spectral clustering methods, the dimensionality of the new space is often set to the desired number of clusters. This setting expects that each new dimension should be able to manifest a cluster.

Spectral clustering is effective in high-dimensional applications such as image processing. Theoretically, it works well when certain conditions apply. Scalability, however, is a challenge. Computing eigenvectors on a large matrix is costly. Spectral clustering can be combined with other clustering methods, such as biclustering.

## 9.5 Clustering graph and network data

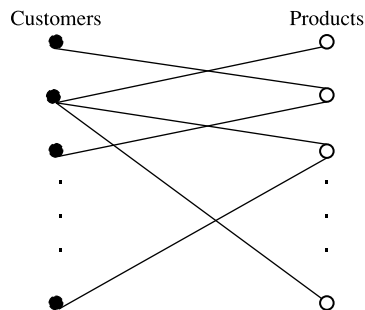
Cluster analysis on graph and network data extracts valuable knowledge and information. Such data are increasingly popular in many applications. We discuss applications and challenges of clustering graph and network data in Section 9.5.1. Similarity measures for this form of clustering are given in Section 9.5.2. You will learn about graph clustering methods in Section 9.5.3.

In general, the terms *graph* and *network* can be used interchangeably. In the rest of this section, we mainly use the term *graph*.

### 9.5.1 Applications and challenges

As a customer relationship manager at a big company, you notice that a lot of data relating to customers and their purchase behavior can be preferably modeled using graphs.

**Example 9.14. Bipartite graph.** The customer purchase behavior at a company can be represented in a *bipartite graph*. In a bipartite graph, vertices can be divided into two disjoint sets so that each edge connects a vertex in one set to a vertex in the other set. For the customer purchase data, one set of vertices represents customers, with one customer per vertex. The other set represents products, with one product per vertex. An edge connects a customer to a product, representing the purchase of the product by the customer. Fig. 9.15 shows an illustration.



**FIGURE 9.15**

Bipartite graph representing customer-purchase data.

“What kind of knowledge can we obtain by a cluster analysis of the customer-product bipartite graph?” By clustering the customers such that those customers buying similar sets of products are placed into one group, a customer relationship manager can make product recommendations. For example, suppose Ada belongs to a customer cluster in which most of the customers purchased a popular game in the last 6 months, but Ada has yet to purchase one. As the manager, you can recommend the game to her.

Alternatively, we can cluster products such that those products purchased by similar sets of customers are grouped together. This clustering information can also be used for product recommendations. For example, if a robot vacuum and an electric smart cooker belong to the same product cluster, then when a customer purchases a robot vacuum, we can recommend the smart cooker. □

Bipartite graphs are widely used in many applications. Consider another example.

**Example 9.15. Web search engines.** In web search engines, search logs are archived to record user queries and the corresponding *click-through information*. (The click-through information tells us on which pages, given as a result of a search, the user clicked.) The query and click-through information can be represented using a bipartite graph, where the two sets of vertices correspond to queries and web pages, respectively. An edge links a query to a web page if a user clicks the web page when asking the query. Valuable information can be obtained by the cluster analysis on the query-web page bipartite graph. For instance, we may identify queries posed in different languages, but mean the same thing, if the click-through information for each query is similar.

As another example, all the web pages on the Web form a directed graph, also known as the *web graph*, where each web page is a vertex, and each hyperlink is an edge pointing from a source page to a destination page. Cluster analysis on the web graph can disclose communities, find hubs and authoritative web pages, and detect web spams. □

In addition to bipartite graphs, cluster analysis can also be applied to other types of graphs, including general graphs, as elaborated in Example 9.16.

**Example 9.16. Social network.** A *social network* is a social structure and can be represented as a graph, where the vertices are individuals or organizations, and the links are interdependencies between the vertices, representing, for example, friendship, common interests, or collaborative activities. For a company, the customers may form a social network, where each customer is a vertex, and an edge links two customers if they know each other.

As a customer relationship manager, you are interested in finding useful information and knowledge that can be derived from the customer social network through clustering analysis. You obtain clusters from the network, where customers in a cluster know each other or have friends in common. Customers within a cluster may influence one another regarding purchase decision making. Moreover, communication channels can be designed to inform the “heads” of clusters (i.e., the “best” connected people in the clusters), so that promotional information can be spread out quickly. Thus you may use customer clustering to promote sales for the company.

As another example, the authors of scientific publications form a social network, where the authors are vertices and two authors are connected by an edge if they coauthored a publication. The network is, in general, a weighted graph because an edge between two authors can carry a weight representing the strength of the collaboration, such as how many publications the two authors (as the end vertices) coauthored. Clustering the coauthor network provides insight as to communities of authors and patterns

of collaboration. Indeed, in the context of (social) network analysis, clustering as finding well connected subgraphs is also known as *community detection*.  $\square$

“Are there any challenges specific to clustering analysis on graph and network data?” In most of the clustering methods discussed so far, objects are represented using a set of attributes. A unique feature of graph and network data is that only objects (as vertices) and relationships between them (as edges) are given. No dimensions or attributes are explicitly defined. To conduct cluster analysis on graph and network data, there are two major new challenges.

- “How can we measure the similarity between two objects on a graph accordingly?” Typically, we cannot use conventional distance measures, such as Euclidean distance. Instead, we need to develop new measures to quantify the similarity. Such measures often are not metric and thus raise new challenges regarding the development of efficient clustering methods. Similarity measures for graphs are discussed in Section 9.5.2.
- “How can we design clustering models and methods that are effective on graph and network data?” Graph and network data are often complicated, carrying topological structures that are more sophisticated than traditional cluster analysis applications. Many graph data sets are large, such as the web graph containing at least tens of billions of web pages in the publicly indexable Web. Graphs can also be sparse where, on average, a vertex is connected to only a small number of other vertices in the graph. To discover accurate and useful knowledge hidden deep in the data, a good clustering method has to accommodate these factors. Clustering methods for graph and network data are introduced in Section 9.5.3.

## 9.5.2 Similarity measures

“How can we measure the similarity or distance between two vertices in a graph?” In our discussion, we examine two types of measures: *geodesic distance* and *distance based on random walk*.

### **Geodesic distance**

A simple measure of the distance between two vertices in a graph is the shortest path between the vertices. Formally, the **geodesic distance**  $d(u, v)$  between two vertices  $u$  and  $v$  is the length in terms of the number of edges of the shortest path between the vertices. For two vertices that are not connected in a graph, the geodesic distance is defined as infinite, that is,  $d(u, v) = +\infty$  if  $u$  and  $v$  are not connected.

Using geodesic distance, we can define several other useful measurements for graph analysis and clustering. Given an undirected graph  $G = (V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of edges, we define the following:

- For a vertex  $v \in V$ , the **eccentricity** of  $v$ , denoted  $eccen(v)$ , is the largest geodesic distance between  $v$  and any other vertex  $u \in V - \{v\}$ . That is,

$$eccen(v) = \max_{u \in V - \{v\}} \{d(u, v)\}.$$

The eccentricity of  $v$  captures how far away  $v$  is from its remotest vertex in the graph.

- The **radius** of graph  $G$  is the minimum eccentricity of all vertices. That is,

$$r = \min_{v \in V} \{eccen(v)\}. \tag{9.28}$$



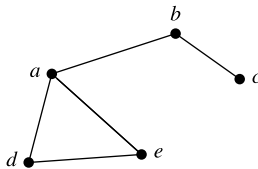


FIGURE 9.16

A graph,  $G$ , where vertices  $c$ ,  $d$ , and  $e$  are peripheral.

The radius captures the distance between the “most central point” and the “farthest border” of the graph.

- The **diameter** of graph  $G$  is the maximum eccentricity of all vertices. That is,

$$l = \max_{v \in V} \{eccen(v)\} = \max_{u, v \in V} \{dist(u, v)\}. \quad (9.29)$$

The diameter represents the largest distance between any pair of vertices. A **peripheral vertex** is a vertex  $v$  that achieves the diameter.

**Example 9.17. Measurements based on geodesic distance.** Consider graph  $G$  in Fig. 9.16. The eccentricity of  $a$  is 2. It is easy to verify that  $eccen(a) = 2$ ,  $eccen(b) = 2$ , and  $eccen(c) = eccen(d) = eccen(e) = 3$ . Thus the radius of  $G$  is 2, and the diameter is 3. Note that it is not necessary that diameter =  $2 \times$  radius. Vertices  $c$ ,  $d$ , and  $e$  are peripheral vertices.  $\square$

### **SimRank: similarity based on random walk and structural context**

For some applications, geodesic distance may be inappropriate in measuring the similarity between vertices in a graph. Here we introduce SimRank, a similarity measure based on random walk and on the structural context of the graph. In mathematics, a *random walk* is a trajectory that consists of taking successive random steps.

**Example 9.18. Similarity between people in a social network.** Let us consider measuring the similarity between two vertices in the customer social network of a company, as discussed in Example 9.16. Here, similarity can be explained as the closeness between two participants in the network, that is, how close two people are in terms of the relationship represented by the social network.

“How well can the geodesic distance measure similarity and closeness in such a network?” Suppose Ada and Bob are two customers in the network, and the network is undirected. The geodesic distance (i.e., the length of the shortest path between Ada and Bob) is the shortest path that a message can be passed from Ada to Bob and vice versa. However, this information is not useful for the customer relationship management of the company, because the company typically does not want to send a specific message from one customer to another. Therefore geodesic distance does not suit the application.

“What does similarity mean in a social network?” We consider two ways to define similarity:

- Two customers are considered similar to each other if they have similar neighbors in the social network. This heuristic is intuitive because, in practice, two people receiving recommendations from

a good number of common friends often make similar decisions. This kind of similarity is based on the local structure (i.e., the *neighborhoods*) of the vertices, and thus is called *structural context-based similarity*.

- Suppose the company sends promotional information to both Ada and Bob in the social network. Ada and Bob may randomly forward such information to their friends (or *neighbors*) in the network. The closeness between Ada and Bob can then be measured by the likelihood that other customers simultaneously receive the promotional information that was originally sent to Ada and Bob. This kind of similarity is based on the random walk reachability over the network, and thus is referred to as *similarity based on random walk*.  $\square$

Let us have a closer look at what is meant by similarity based on structural context and that based on random walk.

The intuition behind similarity based on structural context is that two vertices in a graph are similar if they are connected to similar vertices. To measure such similarity, we need to define the notion of individual neighborhood. In a directed graph  $G = (V, E)$ , where  $V$  is the set of vertices and  $E \subseteq V \times V$  is the set of edges, for a vertex  $v \in V$ , the *individual in-neighborhood* of  $v$  is defined as

$$I(v) = \{u | (u, v) \in E\}. \quad (9.30)$$

Symmetrically, we define the *individual out-neighborhood* of  $v$  as

$$O(v) = \{w | (v, w) \in E\}. \quad (9.31)$$

Following the intuition illustrated in Example 9.18, we define SimRank, a structural-context similarity, with a value that is between 0 and 1 for any pair of vertices. For any vertex,  $v \in V$ , the similarity between the vertex and itself is  $s(v, v) = 1$  because the neighborhoods are identical. For vertices  $u, v \in V$  such that  $u \neq v$ , we can define

$$s(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s(x, y), \quad (9.32)$$

where  $C$  is a constant between 0 and 1. A vertex may not have any in-neighbors. Thus we define Eq. (9.32) to be 0 when either  $I(u)$  or  $I(v)$  is  $\emptyset$ . Parameter  $C$  specifies the rate of decay as similarity is propagated across edges.

SimRank can also be represented using matrices. Let  $\mathbf{A}$  be the column normalized adjacency matrix, where  $\mathbf{A}_{ij} = \frac{1}{|I(v_j)|}$  if there is an edge from  $v_i$  to  $v_j$ , and 0 otherwise. Let  $\mathbf{S}$  be the SimRank matrix, where  $\mathbf{S}_{ij}$  is the SimRank between vertices  $v_i$  and  $v_j$ . Then,

$$\mathbf{S} = \max\{C(\mathbf{A}^T \mathbf{S} \mathbf{A}), \mathbf{I}\},$$

where  $\mathbf{I}$  is an identity matrix.

“How can we compute SimRank?” A straightforward method iteratively evaluates Eq. (9.32) until a fixed point is reached. Let  $s_i(u, v)$  be the SimRank score calculated at the  $i$ th round. To begin, we set

$$s_0(u, v) = \begin{cases} 0 & \text{if } u \neq v \\ 1 & \text{if } u = v. \end{cases} \quad (9.33)$$

We use Eq. (9.32) to compute  $s_{i+1}$  from  $s_i$  as

$$s_{i+1}(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s_i(x, y). \quad (9.34)$$

It can be shown that  $\lim_{i \rightarrow \infty} s_i(u, v) = s(u, v)$ . Additional methods for approximating SimRank are given in the bibliographic notes (Section 9.9).

Now, let us consider similarity based on random walk. A directed graph is *strongly connected* if, for any two nodes  $u$  and  $v$ , there is a path from  $u$  to  $v$  and another path from  $v$  to  $u$ . In a strongly connected graph,  $G = (V, E)$ , for any two vertices,  $u, v \in V$ , we can define the *expected distance* from  $u$  to  $v$  as

$$d(u, v) = \sum_{t: u \rightsquigarrow v} P[t]l(t), \quad (9.35)$$

where  $u \rightsquigarrow v$  is a path starting from  $u$  and ending at  $v$  that may contain cycles but does not reach  $v$  until the end. For a path,  $t = w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_k$ , its length is  $l(t) = k - 1$ . The probability of the path is defined as

$$P[t] = \begin{cases} \prod_{i=1}^{k-1} \frac{1}{|O(w_i)|} & \text{if } l(t) > 0 \\ 0 & \text{if } l(t) = 0. \end{cases} \quad (9.36)$$

To measure the probability that a vertex  $w$  receives a message that is originated simultaneously from  $u$  and  $v$ , we extend the expected distance to the notion of *expected meeting distance*, that is,

$$m(u, v) = \sum_{t: (u, v) \rightsquigarrow (x, x)} P[t]l(t), \quad (9.37)$$

where  $(u, v) \rightsquigarrow (x, x)$  is a pair of paths  $u \rightsquigarrow x$  and  $v \rightsquigarrow x$  of the same length. Using a constant  $C$  between 0 and 1, we define the *expected meeting probability* as

$$p(u, v) = \sum_{t: (u, v) \rightsquigarrow (x, x)} P[t]C^{l(t)}, \quad (9.38)$$

which is a similarity measure based on random walk. Here, the parameter  $C$  specifies the probability of continuing the walk at each step of the trajectory.

It has been shown that  $s(u, v) = p(u, v)$  for any two vertices,  $u$  and  $v$ . That is, SimRank is based on both structural context and random walk.

### **Personalized PageRank and topical PageRank**

In a large network like the web, message passing and user accessing can be modeled by random walks. A user starts from one node  $u$ , and randomly chooses one of the outgoing edges of  $u$  to proceed and reach the next hop. It is also assumed that a user takes a probability  $\alpha$  ( $0 \leq \alpha \leq 1$ ) to terminate a random walk. Random walks can be used to model the distance or similarity from one node to another.<sup>4</sup> Let us look at two similarity measures based on random walks.

<sup>4</sup> In Section 7.6.3, we discuss how to use random walks to model proximity between nodes in graphs.

Given a source node  $u$  and a target node  $v$ , the **personalized PageRank** (PPR) models the similarity of  $v$  to  $u$  by the probability that a random walk starting from  $u$  ends at  $v$ . Formally, the personalized PageRank of  $v$  with respect to  $u$  is

$$PPR(u, v) = P[t \text{ ends at } v | \text{random walk } t \text{ starting from } u]. \quad (9.39)$$

It is easy to see that personalized PageRank is not symmetric. That is, in general,  $PPR(u, v) \neq PPR(v, u)$ . It can be shown that

$$\sum_{u \in V} PPR(u, v) = |V| \cdot PR(v),$$

where  $PR(v)$  is the PageRank of  $v$ .

Now let us consider a more sophisticated scenario, where some nodes may carry some topics. For example, in the web graph, a page may have some predefined topics, such as “politics,” “sports,” “arts,” and so on. Imagine a user interested in a selected topic, say “politics,” conducts biased random walks. The reason we call it a biased random walk because the user chooses the starting node and the out-links following a distribution proportional to the probabilities that the nodes are relevant to the selected topic. In this way, each node in the graph has a probability that a biased random walk with respect to a selected topic may reach the node, which is the **topical PageRank** of the selected topic.

Let  $T_1, \dots, T_l$  be the topics that are under consideration. For a node  $u$ , let  $TPR(u, T_i)$  ( $1 \leq i \leq l$ ) be the probability that  $u$  is the end of a biased random walk with respect to topic  $T_i$ . Then, vector  $\mathbf{T}_u = \langle TPR(u, T_1), \dots, TPR(u, T_l) \rangle$  is called the **topical PageRank** (TPR) of  $u$  and captures the importance of  $u$  with respect to various topics. For two nodes  $u$  and  $v$ , we can measure the similarity between  $u$  and  $v$  by calculating the similarity between their topical PageRank vectors. For example, we can measure the similarity between  $u$  and  $v$  by the cosine similarity between  $\mathbf{T}_u$  and  $\mathbf{T}_v$ .

Please note that personalized PageRank can be regarded as a special case of topical PageRank, where only the source node  $u$  belongs to the selected topic, and every restart jumps back to  $u$  exclusively. The selection of the outgoing edge follows the uniform distribution.

Mathematically, for a topic  $T_i$ , let  $\mathbf{TPR} = [TPR(u_1, T_i) \ \cdots \ TPR(u_n, T_i)]^T$  be the topical PageRank vector recording the topical PageRank of all nodes in the graph. Let  $\mathbf{P}$  be the transition matrix such that

$$\mathbf{P}_{ij}^T = \begin{cases} \frac{1}{|o(u_i)|} & \text{if } (u_i, u_j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Let  $\alpha$  be the damping factor, such that at each step it takes a probability of  $\alpha$  to terminate the current random walk and restart. Then we have

$$\mathbf{TPR} = (1 - \alpha) \cdot \mathbf{P} \cdot \mathbf{TPR} + \alpha \cdot \mathbf{S}, \quad (9.40)$$

where  $\mathbf{S}$  is the distribution of topic  $T_i$  on all nodes in the graph.

A straightforward approach to compute topical PageRank with respect to a topic is to randomly initialize  $\mathbf{TPR}$  in Eq. (9.40) and iteratively update the equation.

### 9.5.3 Graph clustering methods

Let us consider how to conduct clustering on a graph. We first describe the intuition behind graph clustering. We then discuss two general categories of graph clustering methods.

The intuition of finding clusters in a graph is to cut the graph into pieces, each piece being a cluster, such that the vertices within a cluster are well connected, and the vertices in different clusters are connected in a much weaker way. Formally, for a graph,  $G = (V, E)$ , a **cut**,  $C = (S, T)$ , is a partitioning of the set of vertices  $V$  in  $G$ , that is,  $V = S \cup T$  and  $S \cap T = \emptyset$ . The *cut set* of a cut is the set of edges,  $\{(u, v) \in E \mid u \in S, v \in T\}$ . The *size* of the cut is the number of edges in the cut set. For weighted graphs, the size of a cut is the sum of the weights of the edges in the cut set.

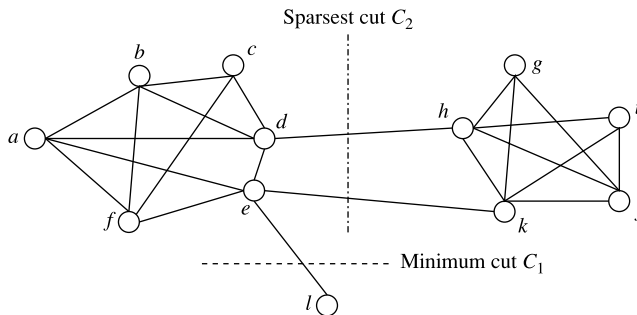
“What kinds of cuts are good for deriving clusters in graphs?” In graph theory and some network applications, a minimum cut is of importance. A cut is *minimum* if the size of the cut is not greater than the size of any other cut. There are polynomial time algorithms to compute minimum cuts of graphs. Can we use those algorithms in graph clustering?

**Example 9.19. Cuts and clusters.** Consider graph  $G$  in Fig. 9.17. The graph has two clusters:  $\{a, b, c, d, e, f\}$  and  $\{g, h, i, j, k\}$ , and one outlier vertex,  $l$ .

Consider cut  $C_1 = (\{a, b, c, d, e, f, g, h, i, j, k\}, \{l\})$ . Only one edge, namely,  $(e, l)$ , crosses the two partitions created by  $C_1$ . Therefore the cut set of  $C_1$  is  $\{(e, l)\}$ , and the size of  $C_1$  is 1. (Note that the size of any cut in a connected graph cannot be smaller than 1.) As a minimum cut,  $C_1$  does not lead to a good clustering because it only separates the outlier vertex,  $l$ , from the rest of the graph.

Cut  $C_2 = (\{a, b, c, d, e, f, l\}, \{g, h, i, j, k\})$  leads to a much better clustering than  $C_1$ . The edges in the cut set of  $C_2$  are those connecting the two “natural clusters” in the graph. Specifically, for edges  $(d, h)$  and  $(e, k)$  that are in the cut set, most of the edges connecting  $d, h, e$ , and  $k$  belong to one cluster.  $\square$

Example 9.19 indicates that using a minimum cut is unlikely to lead to a good clustering. We are better off choosing a cut where, for each vertex  $u$  that is involved in an edge in the cut set, most of the edges connecting to  $u$  belong to one cluster. Formally, let  $\text{deg}(u)$  be the degree of  $u$ , that is, the number



**FIGURE 9.17**

A graph  $G$  and two cuts.

of edges connecting to  $u$ . The *sparsity* of a cut  $C = (S, T)$  is defined as

$$\Phi = \frac{\text{cut size}}{\min\{|S|, |T|\}}. \quad (9.41)$$

A cut is *sparsest* if its sparsity is not greater than the sparsity of any other cut. There may be more than one sparsest cut.

In Example 9.19 and Fig. 9.17,  $C_2$  is a sparsest cut. Using sparsity as the objective function, a sparsest cut tries to minimize the number of edges crossing the partitions and balance the partitions in size.

Consider a clustering on a graph  $G = (V, E)$  that partitions the graph into  $k$  clusters. The **modularity** of a clustering assesses the quality of the clustering and is defined as

$$Q = \sum_{i=1}^k \left( \frac{l_i}{|E|} - \left( \frac{d_i}{2|E|} \right)^2 \right), \quad (9.42)$$

where  $l_i$  is the number of edges between vertices in the  $i$ th cluster, and  $d_i$  is the sum of the degrees of the vertices in the  $i$ th cluster. The modularity of a clustering of a graph is the difference between the fraction of all edges that fall into individual clusters and the fraction that would do so if the graph vertices were randomly connected. The optimal clustering of graphs maximizes the modularity.

Theoretically, many graph clustering problems can be regarded as finding good cuts, such as the sparsest cuts, on the graph. In practice, however, a number of challenges exist:

- **High computational cost:** Many graph cut problems are computationally expensive. The sparsest cut problem, for example, is NP-hard. Therefore finding the optimal solutions on large graphs is often impossible. A good trade-off between efficiency/scalability and quality has to be achieved.
- **Sophisticated graphs:** Graphs can be more sophisticated than the ones described here, involving weights and/or cycles.
- **High dimensionality:** A graph can have many vertices. In a similarity matrix, a vertex is represented as a vector (a row in the matrix) with a dimensionality that is the number of vertices in the graph. Vertices and edges may carry various attributes, too. Therefore graph clustering methods must handle high dimensionality.
- **Sparsity:** A large graph is often sparse, meaning each vertex on average connects to only a small number of other vertices. A similarity matrix from a large sparse graph can also be sparse.

There are different kinds of methods for clustering graph data, which address these challenges. We introduce some representative ones here.

### ***Generic high-dimensional clustering methods on graphs***

The first group of methods are based on generic clustering methods for high-dimensional data. They extract a similarity matrix from a graph using a similarity measure such as those discussed in Section 9.5.2. A generic clustering method can then be applied on the similarity matrix to discover clusters. Clustering methods for high-dimensional data are typically employed. For example, in many scenarios, once a similarity matrix is obtained, spectral clustering methods (Section 9.4) can be applied. Spectral clustering can approximate optimal graph cut solutions. For additional information, please refer to the bibliographic notes (Section 9.9).

### **Specific clustering methods by searching graph structures**

The second group of methods are specific to graphs. They search the graph to find well-connected components as clusters. Let us look at a method called **SCAN** (Structural Clustering Algorithm for Networks) as an example.

Given an undirected graph,  $G = (V, E)$ , for a vertex,  $u \in V$ , the neighborhood of  $u$  is  $\Gamma(u) = \{v \mid (u, v) \in E\} \cup \{u\}$ . Using the idea of structural-context similarity, SCAN measures the similarity between two vertices,  $u, v \in V$ , by the normalized common neighborhood size, that is,

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)||\Gamma(v)|}}. \quad (9.43)$$

The larger the value computed, the more similar the two vertices. SCAN uses a similarity threshold  $\varepsilon$  to define the cluster membership. For a vertex,  $u \in V$ , the  $\varepsilon$ -neighborhood of  $u$  is defined as  $N_\varepsilon(u) = \{v \in \Gamma(u) \mid \sigma(u, v) \geq \varepsilon\}$ . The  $\varepsilon$ -neighborhood of  $u$  contains all neighbors of  $u$  with a structural-context similarity to  $u$  that is at least  $\varepsilon$ .

In SCAN, a *core vertex* is a vertex inside of a cluster. That is,  $u \in V$  is a core vertex if  $|N_\varepsilon(u)| \geq \mu$ , where  $\mu$  is a popularity threshold. SCAN grows clusters from core vertices. If a vertex  $v$  is in the  $\varepsilon$ -neighborhood of a core  $u$ , then  $v$  is assigned to the same cluster as  $u$ . This process of growing clusters continues until no cluster can be further grown. The process is similar to the density-based clustering method, DBSCAN (Section 8.4.1).

Formally, a vertex  $v$  can be *directly reached* from a core  $u$  if  $v \in N_\varepsilon(u)$ . Transitively, a vertex  $v$  can be *reached* from a core  $u$  if there exist vertices  $w_1, \dots, w_n$  such that  $w_1$  can be reached from  $u$ ,  $w_i$  can be reached from  $w_{i-1}$  for  $1 < i \leq n$ , and  $v$  can be reached from  $w_n$ . Moreover, two vertices,  $u, v \in V$ , which may or may not be cores, are said to be *connected* if there exists a core  $w$  such that both  $u$  and  $v$  can be reached from  $w$ . All vertices in a cluster are connected. A cluster is a maximum set of vertices such that every pair in the set is connected.

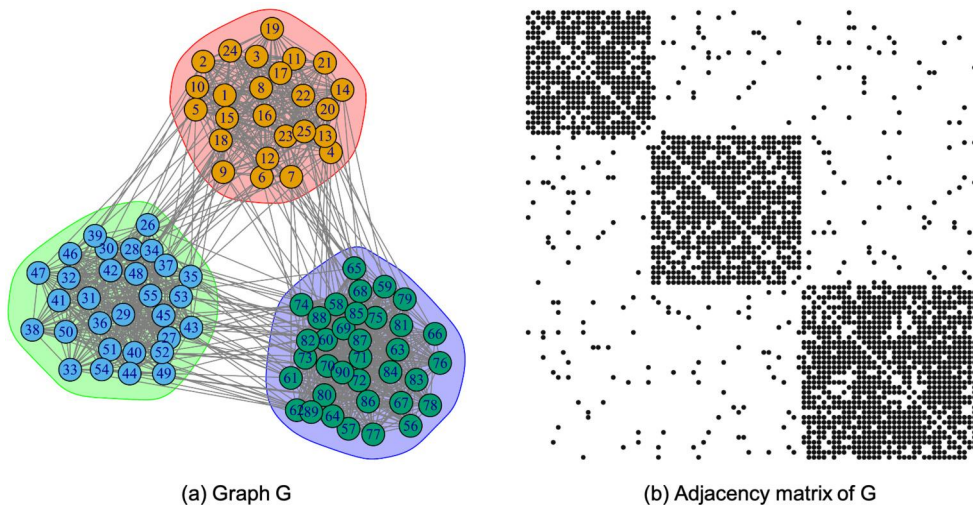
Some vertices may not belong to any cluster. Such a vertex  $u$  is a *hub* if the neighborhood  $\Gamma(u)$  of  $u$  contains vertices from more than one cluster. If a vertex does not belong to any cluster, and is not a hub, it is an *outlier*.

The search framework in SCAN closely resembles the cluster-finding process in DBSCAN. SCAN finds a cut of the graph, where each cluster is a set of vertices that are connected based on the transitive similarity in a structural context. An advantage of SCAN is that its time complexity is linear with respect to the number of edges. In very large and sparse graphs, the number of edges is in the same scale of the number of vertices. Therefore SCAN is expected to have good scalability on clustering large graphs.

### **Probabilistic graphical model-based methods**

Probabilistic graphical model-based methods regard a graph as a set of observations generated by a probabilistic model. Consequently, probabilistic graphical model-based methods use some heuristics on community generation to find clusters as communities in a graph. They often use some probabilistic models of graph structures to describe the dependencies among vertices via the edges.

Three major types of probabilistic graphical models are often used, directed graphical models, undirected graphical models and hybrid graphical models. First, directed graphical models are mainly based on latent variables and leverage similarity of vertices and cluster structures to generate edges that are observable in a graph. Second, undirected graphical models are often based on field structures and use

**FIGURE 9.18**

A graph and the similarity matrix showing the block/cluster structure. *Source:* adapted from Lee and Wilkinson *Applied Network Science* (2019) 4:122.

the cluster label agreement between neighbor vertices to identify clusters. Last, hybrid graphical models transform directed graphical models and undirected graphical models into a unified factor graph and detect clusters.

Let us use the well-known stochastic block model (SBM)-based methods as an example to illustrate the essential ideas. To understand the intuition, consider the graph  $G$  in Fig. 9.18(a), where there are three clusters. Fig. 9.18(b) shows the adjacency matrix, where the nodes are sorted according to the clusters to which they belong. A black dot in the adjacency matrix represents an edge connecting two vertices. As it shows, if we assign the nodes into clusters properly, then each cluster is a dense block in the adjacency matrix, since the nodes in a cluster are densely connected, whereas the edges crossing different clusters (i.e., blocks) are sparse. SBM tries to learn a generative model describing graph block structures from an observed graph, where a block corresponds to a cluster.

Let us describe the generation process of a graph  $G = (V, E)$ , where  $V = \{u_1, \dots, u_n\}$  is the set of vertices, and  $E \subseteq V \times V$  is the set of edges. Denote by  $\mathbf{A}$  the adjacency matrix. Suppose we want to form  $K$  clusters in the graph. To model the assignments of vertices to clusters, for a vertex  $u_i$  ( $1 \leq i \leq n$ ), define a  $K$ -vector  $\mathbf{Z}_i$ , where all elements are 0 except the one that takes value 1 and represents the cluster to which  $u_i$  belongs. For example, in Fig. 9.18(a),  $\mathbf{Z}_1 = [1 \ 0 \ 0]^T$ . Putting all  $\mathbf{Z}_i$  together, we have an  $n \times K$  cluster membership matrix  $\mathbf{Z} = [\mathbf{Z}_1 \ \dots \ \mathbf{Z}_n]^T$ . To model the generation of edges, we define a  $K \times K$  block matrix  $\mathbf{C}$ , where  $\mathbf{C}_{ij}$  ( $1 \leq i, j \leq K$ ) is the probability of occurrence of an edge between a vertex in cluster  $i$  and a vertex in cluster  $j$ . Clearly,  $\mathbf{C}_{ii}$  is the probability of occurrence of an edge within cluster  $i$ . Matrix  $\mathbf{C}$  does not have to be symmetric. Indeed, an asymmetric block matrix can model directed edges between blocks.



SBM assumes that an edge that happens to two vertices  $u_i$  and  $u_j$  is conditionally independent given the cluster memberships  $\mathbf{Z}$ . That is,  $\mathbf{A}_{ij}$  follows the Bernoulli distribution with success probability  $\mathbf{Z}_i^T \mathbf{C} \mathbf{Z}_j$  and is independent from  $\mathbf{A}_{i'j'}$  for  $(i, j) \neq (i', j')$ . This assumption is based on the concept of *stochastic equivalence*. Therefore the total number of edges between two blocks  $i$  and  $j$  is a random variable following the Binomial distribution with mean  $\mathbf{C}_{ij} \cdot b_i \cdot b_j$ , where  $b_i$  and  $b_j$  are the number of vertices assigned to clusters  $i$  and  $j$ , respectively.

Based on the above assumption, given a cluster memberships matrix  $\mathbf{Z}$  and a block matrix  $\mathbf{C}$ , the likelihood of a graph adjacency matrix  $\mathbf{A}$  is given by

$$\pi(\mathbf{A}|\mathbf{Z}, \mathbf{C}) = \prod_{1 \leq i < j \leq n} \pi(\mathbf{A}_{ij}|\mathbf{Z}, \mathbf{C}) = \prod_{1 \leq i < j \leq n} [(\mathbf{Z}_i^T \mathbf{C} \mathbf{Z}_j)^{\mathbf{A}_{ij}} (1 - \mathbf{Z}_i^T \mathbf{C} \mathbf{Z}_j)^{1 - \mathbf{A}_{ij}}]. \quad (9.44)$$

If the graph is directed, we can simply replace the index in Eq. (9.44) from  $1 \leq i < j \leq n$  to  $1 \leq i, j \leq n, i \neq j$ . Eq. (9.44) is also known as the *Bernoulli stochastic block model*.

To apply Eq. (9.44) to find clusters in a graph, we need to initialize  $\mathbf{Z}$  and  $\mathbf{C}$ . Often, we assume that the assignments of different vertices into clusters are independent; that is,  $\mathbf{Z}_i$  and  $\mathbf{Z}_j$  are independent a priori. Moreover, by assuming a prior distribution of clusters  $\boldsymbol{\theta} = [\theta_1 \ \dots \ \theta_K]^T$  such that  $\sum_{i=1}^K \theta_i = 1$ , we have  $Pr(\mathbf{Z}_{ij}) = \theta_j$ , that is, a vertex  $u_i$  is assigned to cluster  $j$  with probability  $\theta_j$ . In other words, the cluster to which  $u_i$  is assigned follows the multinomial distribution with probabilities  $\boldsymbol{\theta}$  and thus

$$\pi(\mathbf{Z}|\boldsymbol{\theta}) = \prod_{i=1}^n \mathbf{Z}_i^T \boldsymbol{\theta} = \prod_{i=1}^n \boldsymbol{\theta}^T \mathbf{Z}_i = \prod_{i=1}^n \theta_i^{b_i}. \quad (9.45)$$

In the above SBM model, the adjacency matrix  $\mathbf{A}$  is the input, the block matrix  $\mathbf{C}$  and the cluster prior distribution  $\boldsymbol{\theta}$  are parameters, and the cluster memberships matrix  $\mathbf{Z}$  is the result. To inference  $\mathbf{Z}$ , we need to assign priors to  $\mathbf{C}$  and  $\boldsymbol{\theta}$  before we can conduct inference. We can assume that  $\mathbf{C}$  has an independent Beta prior, that is,  $\mathbf{C}_{ij} \sim \text{Beta}(\mathbf{P}_{ij}, \mathbf{Q}_{ij})$ , where  $\mathbf{P}$  and  $\mathbf{Q}$  are  $K \times K$  matrices of all positive hyperparameters. We also assume that  $\boldsymbol{\theta}$  is from the Dirichlet( $\alpha \mathbf{1}_K$ ) distribution, where parameter  $\alpha$  is from a Gamma( $a, b$ ) prior. Then, the joint posterior of  $\mathbf{Z}$ ,  $\boldsymbol{\theta}$ ,  $\mathbf{C}$ , and  $\alpha$  satisfy the following.

$$\begin{aligned} \pi(\mathbf{Z}, \boldsymbol{\theta}, \mathbf{C}, \alpha|\mathbf{A}) &\propto \pi(\mathbf{A}, \mathbf{Z}, \boldsymbol{\theta}, \mathbf{C}, \alpha) \\ &= \pi(\mathbf{A}|\mathbf{Z}, \boldsymbol{\theta}, \mathbf{C}, \alpha) \cdot \pi(\mathbf{Z}|\boldsymbol{\theta}, \mathbf{C}, \alpha) \cdot \pi(\boldsymbol{\theta}|\mathbf{C}, \alpha) \cdot \pi(\mathbf{C}|\alpha) \\ &= \pi(\mathbf{A}|\mathbf{Z}, \mathbf{C}) \cdot \pi(\mathbf{Z}|\boldsymbol{\theta}) \cdot \pi(\boldsymbol{\theta}|\alpha) \cdot \pi(\alpha) \\ &\propto \prod_{1 \leq i < j \leq n} [(\mathbf{Z}_i^T \mathbf{C} \mathbf{Z}_j)^{\mathbf{A}_{ij}} (1 - \mathbf{Z}_i^T \mathbf{C} \mathbf{Z}_j)^{1 - \mathbf{A}_{ij}}] \cdot \prod_{i=1}^n \mathbf{Z}_i^T \boldsymbol{\theta} \\ &\quad \cdot [\Gamma(K\alpha) \mathbf{1}\{\sum_{i=1}^K \theta_i = 1\}] \prod_{i=1}^K \frac{\theta_i^{\alpha-1}}{\Gamma(\alpha)} \\ &\quad \cdot \prod_{1 \leq i \leq j \leq K} [\mathbf{C}_{ij}^{\mathbf{P}_{ij}-1} (1 - \mathbf{C}_{ij}^{\mathbf{Q}_{ij}-1})] \cdot \alpha^{a-1} e^{-b\alpha} \end{aligned} \quad (9.46)$$

We can use a Markov chain Monte Carlo method to conduct inference according to Eq. (9.46).

There are many variances of stochastic block models that incorporate different constraints and background knowledge. Please refer to the bibliography section for the related references.

## 9.6 Semisupervised clustering

In the conventional setting of clustering analysis, the data objects to be clustered are not labeled. However, in some application scenarios, we may have some domain knowledge that may help us in clustering analysis. Let us consider an example.

**Example 9.20. Clustering with partially labeled data.** Imagine that you want to build a malware detector. You collect a large number of images of host systems. You and some of your colleagues label a small number of such images into two categories: malware and benign. However, there is no hope that all or even a fair portion of such images are labeled properly and timely. What should you make good use of the data?

An immediate idea is to use the labeled data to build a classifier. However, the proportion of labeled data is very small. A classifier built on such a small amount of data may not be effective and cannot use a large amount of unlabeled data. In other words, the vast majority of the data available cannot be used by a classification method.  $\square$

The semisupervised clustering approaches address the clustering analysis scenarios where some domain knowledge is available. Those methods build on the clustering methods and incorporate different types of domain knowledge to strengthen the clustering results. In this section, we discuss different types of semisupervised clustering methods according to different types of domain knowledge available in addition to the unlabeled data.

### 9.6.1 Semisupervised clustering on partially labeled data

Example 9.20 illustrates the situation where a small portion of data to be clustered comes with labels. How can we make good use of the small amount of labeled data to strengthen an existing clustering method, such as  $k$ -means?

Let  $D = \{x_1, \dots, x_n\}$  be a set of objects to be clustered and  $K$  be the target number of clusters. In addition, let  $S_1, \dots, S_K \subset D$  be  $K$  exclusive subsets of labeled objects such that  $x_i \in S_j$  ( $1 \leq i \leq n$ ,  $1 \leq j \leq K$ ) implies that object  $x_i$  is known to belong to cluster  $C_j$ , and  $S_j \cap S_{j'} = \emptyset$  ( $1 \leq j < j' \leq K$ ). Let  $S = \bigcup_{j=1}^K S_j$ .

The *constrained  $k$ -means* method extends the classical  $k$ -means algorithm to make good use of the labeled data. The central idea is to use the labeled data to produce the initial means of the clusters. The constrained  $k$ -means algorithm works in the following steps.

1. For each cluster  $j$  ( $1 \leq j \leq K$ ), compute the mean  $c_j$  of the labeled subset  $S_j$ . That is,  $c_j = \frac{\sum_{x_i \in S_j} x_i}{|S_j|}$ .
2. Assign all objects in  $D$  to clusters. For each object  $x_i \in D$ , there are two cases. If  $x_i \in S$ , then there exists a labeled subset  $S_j$  ( $1 \leq j \leq K$ ) such that  $x_i \in S_j$ . We assign  $x_i$  to cluster  $C_j$ . If  $x_i \notin S$ , then we assign  $x_i$  to the cluster  $C_j$  whose mean is closest to  $x_i$ , that is,  $j = \arg \min_k \{dist(x_i, c_k)\}$ .

3. Update the means of the clusters  $C_1, \dots, C_K$ . That is, for  $1 \leq j \leq K$ , we update the mean  $c_j = \frac{\sum_{x_i \in C_j} x_i}{|C_j|}$ .
4. Repeat Steps 2 and 3 until the algorithm converges (similar to the convergence condition for the  $k$ -means algorithm).

The constrained  $k$ -means algorithm fully trusts the labels. As shown in Step 2, it always assigns a labeled object to the cluster of the label. However, what if the labeled data could contain some errors? The possible errors in the labels can be tackled by another semisupervised clustering method, *seeded  $k$ -means*, which is the same as constrained  $k$ -means except for Step 2. In Step 2, seeded  $k$ -means always assigns an object to the cluster whose mean is the closest, no matter whether or what the object is labeled.

## 9.6.2 Semisupervised clustering on pairwise constraints

Domain knowledge that can be useful for clustering analysis can come not only from labeling individual objects but also the knowledge about the pairwise relations between some objects.

**Example 9.21. Clustering with pairwise constraints.** Suppose as a customer relationship manager of a wholesale company, you want to use a clustering method to divide the representatives of your customer companies into several groups so that a marketing campaign event is run for a group. You may have multiple representatives from the same customer company. They should be invited to the same event. In order to express such domain knowledge, you want to specify must-link constraints between every pair of representatives who should be invited together.

Moreover, you may have two customer companies that are head-to-head competitors. You may want to make sure that representatives from two head-to-head competitors are not invited to the same event. Accordingly, you want to specify cannot-link constraints between every pair of representatives who should not be invited to the same event.  $\square$

A *pairwise constraint* specifies how a pair or a set of instances should be grouped in the clustering analysis. Two common types of pairwise constraints are as follows:

- **Must-link constraints.** If a must-link constraint is specified on two objects  $x$  and  $y$ , then  $x$  and  $y$  should be grouped into one cluster in the output of the cluster analysis. These must-link constraints are transitive. That is, if  $\text{must-link}(x, y)$  and  $\text{must-link}(y, z)$ , then  $\text{must-link}(x, z)$ .
- **Cannot-link constraints.** Cannot-link constraints are the opposite of must-link constraints. If a cannot-link constraint is specified on two objects,  $x$  and  $y$ , then in the output of the cluster analysis,  $x$  and  $y$  should belong to different clusters. Cannot-link constraints can be entailed. That is, if  $\text{cannot-link}(x, y)$ ,  $\text{must-link}(x, x')$ , and  $\text{must-link}(y, y')$ , then  $\text{cannot-link}(x', y')$ .

*How can we incorporate pairwise constraints in clustering methods?* COP- $k$ -means is a revision of the classical  $k$ -means algorithm to accommodate pairwise constraints. Essentially, COP- $k$ -means assigns objects to clusters in a way fully respecting the constraints. Let  $D = \{x_1, \dots, x_n\}$  be the set of objects to be clustered.

The algorithm works as follows.

1. Arbitrarily choose  $k$  objects in  $D$  as the initial cluster centers  $c_1, \dots, c_K$ , such that there is no must-link between any two of those  $k$  objects.

- Assign each object  $x_i$  ( $1 \leq i \leq n$ ) to cluster  $C_j$  ( $1 \leq j \leq K$ ) whose mean is the closest, and there is no violation of the pairwise constraints. That is, let

$$H(x_i) = \{C_j | 1 \leq k \leq K, \text{ no constraints are violated} \\ \text{when } x_i \text{ is assigned to cluster } C_j\}$$

be the set of clusters that may host  $x_i$  without violating any constraints. Then,

$$j = \arg \min_{k \in H(x_i)} \{dist(x_i, c_k)\}.$$

- Update the means of the clusters  $C_1, \dots, C_K$ . That is, for  $1 \leq j \leq K$ , we update the mean  $c_j = \frac{\sum_{x_i \in C_j} x_i}{|C_j|}$ .
- Repeat Steps 2 and 3 until the algorithm converges (similar to the convergence condition for the  $k$ -means algorithm).

Because COP- $k$ -means ensures that no constraints are violated at every step, it does not require any backtracking. It is a greedy algorithm for generating a clustering that satisfies all constraints, provided that no conflicts exist among the constraints.

While COP- $k$ -means strictly respects the specified must-link and cannot-link constraints as *hard constraints*, in some application scenarios, one may want to treat those as *soft constraints*. When a clustering violates a soft constraint, a penalty is imposed on the clustering. Therefore the optimization goal of the clustering contains two parts: optimizing the clustering quality and minimizing the constraint violation penalty. The overall objective function is a combination of the clustering quality score and the penalty score.

For example, the PCKmeans algorithm extends the  $k$ -means algorithm and handles soft constraints by revising the objective function. Let  $ML$  be the set of must-link constraints. That is, for any  $(x_i, x_j) \in ML$ , there is a must-link constraint between  $x_i$  and  $x_j$ . Let  $CL$  be the set of cannot-link constraints. That is, for any  $(x_i, x_j) \in CL$ , there is a cannot-link constraint between  $x_i$  and  $x_j$ . PCKmeans minimizes the following objective function:

$$\sum_{k=1}^K \sum_{x_i, x_j \in C_k} dist(x_i, x_j)^2 + \sum_{x_i, x_j \in ML} p_{x_i, x_j}^{ML} I(C(x_i) \neq C(x_j)) + \sum_{x_i, x_j \in CL} p_{x_i, x_j}^{CL} I(C(x_i) = C(x_j)),$$

where  $I(\cdot)$  returns 1 if the parameter is true and 0 otherwise,  $C(x_i)$  identifies the cluster-id that  $x_i$  is assigned to, and  $p_{x_i, x_j}^{ML}$  and  $p_{x_i, x_j}^{CL}$  are the penalties for violating the must-link and cannot-link constraints between  $x_i$  and  $x_j$ , respectively.

The first term in this objective function is the sum of squared error for all objects that is also the objective in  $k$ -means. The second and third terms, respectively, are the penalties on the violations of must-link and cannot-link constraints.

### 9.6.3 Other types of background knowledge for semisupervised clustering

We use partially labeled data and pairwise constraints as examples to illustrate how background knowledge may be present and help clustering analysis, and how the traditional clustering methods can be

extended to make good use of the available background knowledge. In practice, the background knowledge that can be used in clustering analysis is not limited by the two types we discussed above. Here, we use some examples to demonstrate more types of background knowledge. We will not go deep into the corresponding clustering algorithms. Instead, interested readers are encouraged to explore the related literature. Our bibliographic notes provide a series of pointers.

### ***Semisupervised hierarchical clustering***

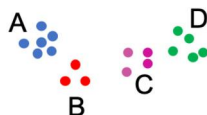
Hierarchical clustering returns a dendrogram instead of just one partitioning of objects. Thus hierarchical clustering can entertain richer background knowledge. At the same time, some constraints used in the partitioning clustering approaches may need to be extended or revised.

For example, since a full dendrogram has all objects in the data set at the root, that is, the most general level, and each object in a separate cluster at the finest level, every must-link constraint is satisfied at the root and every cannot-link constraint is satisfied at the finest level. Must-link and cannot-link constraints for hierarchical clustering have to come with more context information. As a concrete example, when conducting hierarchical clustering on animal species, a taxonomist may specify a *must-link before* constraint between platypus and echidna, meaning these two species should be clustered together before they are clustered with other species. The taxonomist may also provide more domain knowledge by specifying an *ordering constraint* (platypus, echidna, beaver). The constraint means that, in the resulting hierarchical clustering dendrogram, platypus and echidna should be clustered together before they join beaver in a cluster at a higher level.

### ***Clusters associated with outcome variables***

In some applications, we may wish to find clusters that are associated with one or multiple given variables. The outcome variables are often a “noisy surrogate” for some unobserved clusters that are interesting to users.

For example, a marketing manager may conduct clustering analysis to partition customers into groups. The clustering method may use the observed features from the customer profiles, such as address, household income, gender, and age. The manager may wish to find clusters that are associated with an outcome variable of annual spending amount on products sold by the company. Fig. 9.19 illustrates the idea. The color of a data point reflects the annual spending amount, the warmer the higher. Without considering this output variable, we may form four clusters: *A*, *B*, *C*, and *D*. If hierarchical clustering is applied, we may combine *A* and *B* into one second-level cluster and *C* and *D* into another second-level cluster. However, when the output variable is considered, since the points in *B* and *C* have more similar spending amounts, we may want to combine *B* and *C* into a second-level cluster. The background knowledge by the output variable provides us with some useful guidance to form clusters.



**FIGURE 9.19**

Clusters associated with an outcome variable.

### ***Active and interactive learning for semisupervised clustering***

Background knowledge like must-link and cannot-link constraints is helpful for semisupervised clustering. However, a user may have difficulty in specifying must-link and cannot-link constraints on a large set of objects to be clustered. To tackle this challenge, active and interactive learning for semisupervised clustering is explored.

*Active learning* can be conducted to obtain important must-link and cannot-link constraints. For example, one idea is to get at least one object from each cluster with a small number of requests to users for providing pairwise relationship information. One possible approach is to pick an object that is farthest from all the other points and another object in an existing cluster and ask a user to judge whether a must-link constraint should be put on those two objects. If so, then the farthest point is assigned to the cluster. If not, a new cluster is created for the farthest point. This process continues until we have enough initial clusters.

User feedback can be accommodated not only at the beginning of the clustering, but can be taken interactively during the whole iterative clustering process. For example, a user can interactively provide feedback about the quality of clusters in the intermediate results. The types of feedback may include (1) an object is put in a wrong cluster; (2) an object should be moved to a more appropriate cluster; (3) two objects should be put in the same cluster (similar to a must-link constraint); and (4) two objects should be put in different clusters (similar to a cannot-link constraint). A clustering algorithm can take the feedback and update the clusters by either directly adjusting the cluster assignment or adjusting the similarity measure and rerunning the clustering process.

---

## 9.7 Summary

- In conventional cluster analysis, an object is assigned to one cluster exclusively. However, in some applications, there is a need to assign an object to one or more clusters in a fuzzy or probabilistic way. **Fuzzy clustering** and **probabilistic model-based clustering** allow an object to belong to one or more clusters. A **partition matrix** records the membership degree of objects belonging to clusters.
- **Probabilistic model-based clustering** assumes that a cluster is a parameterized distribution. Using the data to be clustered as the observed samples, we can estimate the parameters of the clusters.
- A **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters. Conceptually, each observed object is generated independently by first choosing a probabilistic cluster according to the probabilities of the clusters, and then choosing a sample according to the probability density function of the chosen cluster.
- An **expectation-maximization algorithm** is a framework for approaching maximum likelihood or maximum a posteriori estimates of parameters in statistical models. Expectation-maximization algorithms can be used to compute fuzzy clustering and probabilistic model-based clustering.
- **High-dimensional data** poses grand challenges for cluster analysis, including how to model high-dimensional clusters and how to search for such clusters. The **curse of dimensionality** is mainly caused by *many irrelevant or correlated attributes, data sparsity, distance concentration effect of similarity measures, and difficulty in optimization.*
- There are two different kinds of subspaces that clustering methods may target at, namely **axis-parallel subspaces** and **arbitrarily-oriented subspaces**. There are two major categories of clustering methods for high-dimensional data: **clustering methods** and **dimensionality reduction meth-**

**ods. Clustering methods** search for clusters in subspaces of the original space and can be further categorized into **subspace clustering methods**, **projected clustering methods**, and **biclustering methods**. **Dimensionality reduction methods** create a new space of lower dimensionality and search for clusters there.

- **Biclustering methods** cluster objects and attributes simultaneously. Types of biclusters include biclusters with **constant values**, **constant values on rows/columns**, **coherent values**, and **coherent evolutions on rows/columns**. Two major types of biclustering methods are **optimization-based methods** and **enumeration methods**.
- **Dimensionality reduction** transforms a high-dimensional data set into a low-dimensional space so that the low-dimensional representation retains meaningful properties of the original data, ideally approaching the intrinsic dimensions of the underlying structures.
- There are many dimensionality reduction methods. **Principal component analysis (PCA)** is frequently used to identify the most meaningful basis to re-express a data set. **Nonnegative matrix factorization (NMF)** decomposes a data set  $\mathbf{X} \approx \mathbf{HW}$ , where the entries in  $\mathbf{H}$  and  $\mathbf{W}$  are non-negative.  $\mathbf{H}$  and  $\mathbf{W}$  represent how objects are assigned to clusters and the “centers” of clusters, respectively. **Spectral clustering** constructs new dimensions using an affinity matrix.
- **Clustering graph and network data** have many applications, such as social network analysis and web search. The major challenges include how to measure the similarity between objects in a graph and how to design clustering models and methods for graph and network data.
- **Geodesic distance** is the number of edges between two vertices on a graph. It can be used to measure similarity. Alternatively, similarity in graphs, such as social networks, can be measured using structural context and random walk. **SimRank** is a similarity measure that is based on both structural context and random walk. Some other similarity measures for graphs include **personalized PageRank** and **topical PageRank**.
- Graph clustering can be modeled as computing **graph cuts**. A **sparsest cut** may lead to a good clustering, whereas **modularity** can be used to measure the clustering quality. Computing graph cuts on large graphs faces several challenges, including *high computational cost*, *sophisticated graphs*, *high dimensionality*, and *sparsity*.
- The **generic graph clustering methods** extract a similarity matrix from a graph using a similarity measure and then apply a generic clustering method on the similarity matrix to discover clusters. **SCAN** is a graph clustering algorithm that searches graph structures to identify well-connected components as clusters. **Probabilistic graphical model-based methods** regard a graph as a set of observations generated by a probabilistic model and use some heuristics on community generation to find clusters as communities in a graph. The **stochastic block model (SBM)** is an example.
- **Semisupervised clustering** uses domain knowledge to improve clustering results. The domain knowledge may take the form of partially labeled data, pairwise constraints on objects, and some other types. Some classical clustering algorithms can be extended to accommodate those constraints.

---

## 9.8 Exercises

- 9.1.** Traditional clustering methods are rigid in that they require each object to belong exclusively to only one cluster. Explain why this is a special case of fuzzy clustering. You may use *k*-means as an example.

- 9.2.** An e-commerce company carries 1000 products,  $P_1, \dots, P_{1000}$ . Consider customers Ada, Bob, and Cathy such that Ada and Bob purchase three products in common,  $P_1, P_2$ , and  $P_3$ . For the other 997 products, Ada and Bob independently purchase seven of them randomly. Cathy purchases 10 products, randomly selected from the 1000 products. In Euclidean distance, what is the probability that  $\text{dist}(\text{Ada}, \text{Bob}) > \text{dist}(\text{Ada}, \text{Cathy})$ ? What if Jaccard similarity (Chapter 2) is used? What can you learn from this example?
- 9.3.** Can you show that the  $k$ -medoids method can also be implemented in the EM algorithm framework?
- 9.4.** In the EM algorithm for mixture models, if for all univariate Gaussian distributions  $\Theta_j$  ( $1 \leq j \leq k$ ),  $\sigma_j = \sigma$ , that is, all have the standard deviation. Can you simplify the calculation in the E-step and the M-step accordingly?
- 9.5.** Show that  $I \times J$  is a bicluster with coherent values if and only if, for any  $i_1, i_2 \in I$  and  $j_1, j_2 \in J$ ,  $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$ .
- 9.6.** In soft projected clustering method LAC, explain how the weights and the clusters can be computed using the EM algorithm.
- 9.7.** Compare the MaPle algorithm (Section 9.3) with the frequent closed itemset mining algorithm, CLOSET (Pei, Han, and Mao [PHM00]). What are the major similarities and differences?
- 9.8.** Given 20 data points in 2-D space whose first principal component is  $u = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})'$ .
- Suppose we add one more data point at  $(2, 2)'$ ; how would that affect the first principal component?
  - Suppose we add one more data point at  $(3, 0)'$ ; how would that affect the first principal component?
  - Suppose we add an *infinite* number of data points at  $(5, 0)'$ ; how would that affect the first principal component?
- 9.9.** Given  $n$  data tuples in  $d$ -dimensional space, we can represent them as an  $n \times d$  matrix  $X$ , where the rows of  $X$  are for different data tuples and columns are for different features. NMF introduced in Section 9.4.2 performs matrix low-rank approximation with the constraint that both low-rank matrices must be nonnegative. In this exercise, we will learn that  $k$ -Means clustering can also be viewed as a special form of matrix low-rank approximation. Based on that, we will compare the similarity and difference between these two clustering methods.
- Prove that K-Means clustering can be viewed as a special form of matrix low-rank approximation. That is, the optimization objective of K-Means is equivalent to

$$\{\mathbf{W}^*, \mathbf{H}^*\} = \operatorname{argmin}_{\mathbf{W}, \mathbf{H}} \|\mathbf{X} - \mathbf{H}\mathbf{W}\|_{fro}^2, \quad (9.47)$$

where  $\|\cdot\|_{fro}$  is the matrix Frobenius norm,  $\mathbf{H}$  and  $\mathbf{W}$  are two low-rank matrices with appropriate constraints. In particular, what is the size constraint on  $\mathbf{H}$  and  $\mathbf{W}$ , respectively? What are additional constraints we need to impose on  $\mathbf{H}$  and/or  $\mathbf{W}$ , so that Eq. (9.47) is equivalent to the optimization objective of K-Means?

- Based on the above analysis, what are the commonalities between NMF and K-Means? What are the difference between them?
- 9.10.** In this exercise, we will learn about the mathematical details underlying many spectral clustering methods (Section 9.4.3). Given an  $n \times n$  similarity matrix  $W$  whose elements are the similarities between the corresponding data tuples (i.e.,  $W(i, j)$  measures the similarity between tuples  $i$  and



$j$ ). We wish to partition the data tuples into two clusters. Let  $q$  be a cluster membership vector of length  $n$ :  $q(i) = 1$  if data tuple  $i$  belongs to Cluster A; and  $q(i) = -1$  if it belongs to Cluster B. One way to find these two clusters is to minimize the so-called *cut-size*, which measures the total similarities across different clusters:

$$q^* = \arg \min_{q \in \{-1, 1\}^n} J = \frac{1}{4} \sum_{i, j=1}^n (q(i) - q(j))^2 W(i, j). \quad (9.48)$$

- a. Prove that the *cut-size*  $J = \frac{1}{2} q^T (D - W) q$ , where  $D$  is the degree matrix of  $W$ :  $D(i, i) = \sum_{j=1}^n W(i, j)$  and  $D(i, j) = 0$  for  $j \neq i$ ; and  $T$  is the vector transpose.
- b. It is very difficult to directly optimize Eq. (9.48) since the cluster membership vector  $q$  is a binary vector. In practice, we relax  $q$  and allow it to take real numbers, and aim to solve the following optimization problem instead. Prove that the optimal solution of Eq. (9.49) is given by the eigenvector of  $D - W$  that corresponds to the second smallest eigenvalue.

$$\begin{aligned} q^* = \arg \min_{q \in \mathbb{R}^n} q^T (D - W) q \\ \text{s.t. } \sum_{i=1}^n q(i)^2 = n \end{aligned} \quad (9.49)$$

- 9.11. SimRank is a similarity measure for clustering graph and network data.
  - a. Prove  $\lim_{i \rightarrow \infty} s_i(u, v) = s(u, v)$  for SimRank computation.
  - b. Show  $s(u, v) = p(u, v)$  for SimRank.
- 9.12. In a large sparse graph where on average each node has a low degree, is the similarity matrix using SimRank still sparse? If so, in what sense? If not, why? Deliberate on your answer.
- 9.13. Compare the SCAN algorithm (Section 9.5.3) with DBSCAN (Section 8.4.1). What are their similarities and differences?
- 9.14. Consider partitioning clustering and the following constraint on clusters: The number of objects in each cluster must be between  $\frac{n}{k}(1 - \delta)$  and  $\frac{n}{k}(1 + \delta)$ , where  $n$  is the total number of objects in the data set,  $k$  is the number of clusters desired, and  $\delta$  in  $[0, 1)$  is a parameter. Can you extend the  $k$ -means method to handle this constraint? Discuss situations where the constraint is hard and soft.

---

## 9.9 Bibliographic notes

Höppner Klawonn, Kruse, and Runkler [HKKR99] provide a thorough discussion of fuzzy clustering. The fuzzy  $c$ -means algorithm (on which Example 9.7 is based) is proposed by Bezdek [Bez81]. Fraley and Raftery [FR02] give a comprehensive overview of model-based cluster analysis and probabilistic models. McLachlan and Basford [MB88] present a systematic introduction to mixture models and applications in cluster analysis.

Dempster, Laird, and Rubin [DLR77] are recognized as the first to introduce the EM algorithm and give it its name. However, the idea of the EM algorithm has been “proposed many times in special circumstances” before, as admitted in Dempster, Laird, and Rubin [DLR77]. Wu [Wu83] gives the correct analysis of the EM algorithm.

Mixture models and EM algorithms are used extensively in many data mining applications. Introductions to model-based clustering, mixture models, and EM algorithms can be found in recent textbooks on machine learning and statistical learning—for example, Bishop [Bis06a], Marsland [Mar09], and Alpaydin [Alp11].

The increase of dimensionality has severe effects on distance functions, as indicated by Beyer et al. [BGRS99]. It also has had a dramatic impact on various techniques for classification, clustering, and semisupervised learning (Radovanović, Nanopoulos, and Ivanović [RNI09]).

Kriegel, Kröger, and Zimek [KKZ09] present a comprehensive survey on methods for clustering high-dimensional data. Parsons, Haque, and Liu [PHL04] review subspace clustering for high-dimensional data. The CLIQUE algorithm is developed by Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98]. The PROCLUS algorithm is proposed by Aggarwal et al. [APW<sup>+</sup>99].

The technique of biclustering is initially proposed by Hartigan [Har72]. The term *biclustering* is coined by Mirkin [Mir98]. Cheng and Church [CC00] introduce biclustering into gene expression data analysis. There are many studies on biclustering models and methods. Madeira and Oliveira [MO04] and Tanay, Sharan, and Shamir [TSS04] present surveys on biclustering for biological data analysis. The notion of  $\delta$ -pCluster is introduced by Wang, Wang, Yang, and Yu [WWYY02]. In this chapter, we introduce the  $\delta$ -cluster algorithm by Cheng and Church [CC00] and MaPle by Pei et al. [PZC<sup>+</sup>03] as examples of optimization-based methods and enumeration methods for biclustering, respectively.

Dimensionality reduction is a topic involving many areas. For informative surveys, please see [CG15]. Nonnegative matrix factorization is developed by Paatero and Tapper [PT94] and Lee and Seung [LS99]. Donath and Hoffman [DH73] and Fiedler [Fie73] pioneer spectral clustering. In this chapter, we use an algorithm proposed by Ng, Jordan, and Weiss [NJW01] as an example. For a thorough tutorial on spectral clustering, see Luxburg [Lux07] and Filippone et al. [FCMR08].

Clustering graph and network data is an important and fast-growing topic. Schaeffer [Sch07], Nascimento and de Carvalho [Nd11], Aggarwal and Wang [AW10], and Malliaros and Vazirgianis [MV13] provide several surveys. The SimRank measure of similarity is developed by Jeh and Widom [JW02]. PageRank and personalized PageRank are proposed by Page, Brin, Motwani, and Winograd [PBMW98]. Bahmani, Chowdhury, and Geol [BCG10] discuss fast incremental and personalized PageRank. Xu et al. [XYFS07] propose the SCAN algorithm. Arora, Rao, and Vazirani [ARV09] discuss the sparsest cuts and approximation algorithms. Stochastic block models are first proposed by Holland, Laskey, and Leinhardt [HLL83]. Rohe, Chatterjee, and Yu [RCY11] discuss spectral clustering and high-dimensional stochastic block models.

Semisupervised clustering has been extensively studied [BBM02,BBM04,GCB04,Bai13]. Constrained kmeans is developed by Wagstaff, Cardie, Rogers, and Schrödl [WCRS01]. Basu, Banerjee, and Mooney [BBM02] develop seeded kmeans. The COP- $k$ -means algorithm is given by Wagstaff et al. [WCRS01]. Clustering with constraints has been extensively studied. Davidson, Wagstaff, and Basu [DWB06] propose the measures of informativeness and coherence. Zheng and Li [ZL11] propose a framework for semisupervised hierarchical clustering. Settles [Set10] present a thorough review on active learning literature. Cohn, Caruana, and McCallum [CCM03] discuss interactive learning for semisupervised clustering.

This page intentionally left blank

**In this chapter, you will learn** deep learning, a powerful family of techniques based on *artificial neural networks* with broad applications in computer vision, natural language processing, machine translation, social network analysis, and so on. It has been used in a variety of data mining tasks, including classification, clustering, outlier detection, and reinforcement learning. We start with the basic concepts (Section 10.1). Then, we introduce key algorithmic techniques for training an effective deep learning model (Section 10.2), and commonly used deep learning model architectures, including convolutional neural networks (Section 10.3), recurrent neural networks (Section 10.4), and graph neural networks (Section 10.5).

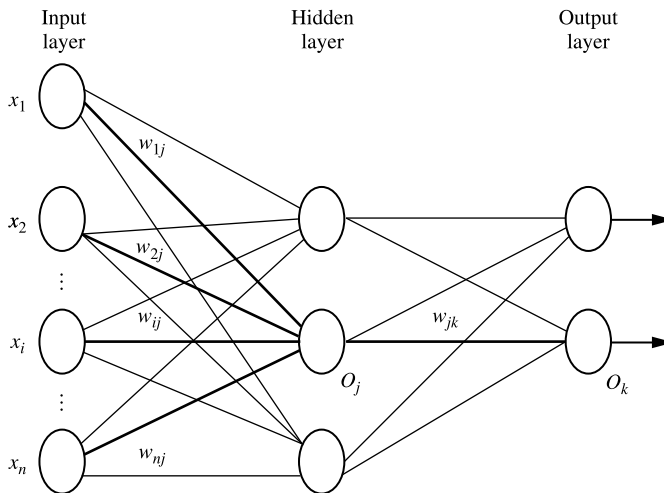
## 10.1 Basic concepts

### 10.1.1 What is deep learning?

Deep learning is based on **artificial neural networks** (ANNs) (or **neural networks** for short). Fig. 10.1 shows an illustrative example of neural network, and we will dive into details in a minute. Roughly speaking, a **neural network** is a set of connected input-output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights in order to predict the correct target values (e.g., class labels) of the input tuples.

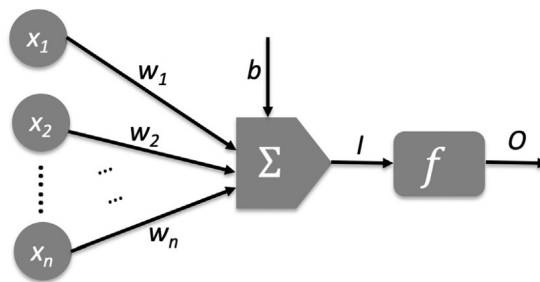
A neural network is made up of interconnected **units**. So, what is a unit? Actually, we have already seen it! Recall that in Chapter 6, we introduced some basic classifiers, such as perceptron and logistic regression. Consider a data tuple  $\mathbf{X}$  with  $n$  attributes:  $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n)$ . Both perceptron and logistic regression classifier first take a linear weighted sum of different attributes or features  $\sum_{i=1}^n \mathbf{X}_i w_i + b$ , where  $w_i$  ( $i = 1, \dots, n$ ) are the weights and  $b$  is the bias scalar. Then, perceptron predicts the class label based on the *sign* of the linear weighted sum  $\hat{y} = \text{sign}(\sum_{i=1}^n \mathbf{X}_i w_i + b)$ , where  $\text{sign}(z) = 1$  if  $z \geq 0$  and  $\text{sign}(z) = 0$  otherwise; logistic regression predicts the class posterior probability based on the sigmoid function of the linear weighted sum  $P(y = 1|\mathbf{X}) = \sigma(\sum_{i=1}^n \mathbf{X}_i w_i + b)$ , where  $\sigma(z) = \frac{1}{1+\exp(-z)}$  is the sigmoid function. In the neural network terminology, both perceptron and logistic regression can be viewed as a unit.

Formally, a unit is a mathematical function that (1) takes a linear weighted sum of the input, and then (2) passes the sum through an **activation function**  $f(\cdot)$ . The activation function  $f(\cdot)$  is typically a nonlinear function, such as the sign function in perceptron and the sigmoid function  $\sigma(\cdot)$  in logistic regression. Many other choices for the activation function exist, and we will introduce some of them in Section 10.2. For the deep learning algorithms, it is convenient to introduce two additional notations. The first one is the net input  $I = \sum_{i=1}^n \mathbf{X}_i w_i + b$  of the activation function. The second is the output of the unit  $O = f(I)$ . A pictorial illustration of a unit is presented in Fig. 10.2.



**FIGURE 10.1**

Multilayer feed-forward neural network.



**FIGURE 10.2**

An illustrative example of unit. Given the inputs  $X_1, \dots, X_n$ , a unit takes a linear weighted sum and then passes the sum through an activation function  $f(\cdot)$ .  $I = \sum_{i=1}^n X_i w_i + b$  is the net input of the activation function and  $O = f(I)$  is the output of the unit.  $w_i$  ( $i = 1, \dots, n$ ) are weights and  $b$  is the bias scalar.

A neural network is essentially a collection of interconnected units. Depending on how different units are organized with each other, there are many different kinds of neural networks. Among them, a very important and powerful type of neural network is called **multilayer feed-forward neural network**. Formally, a multilayer feed-forward neural network consists of an *input layer*, one or more *hidden layers*, and an *output layer*. An example of a multilayer feed-forward network is shown in Fig. 10.1.

Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the **input layer**. These

inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuronlike” units, known as a **hidden layer**. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers could be arbitrary (1 hidden layer in this example), outputs of the last hidden layer are input to units making up the **output layer**, which emits the network’s predictions for given tuples.

The units in the input layer are called **input units**. The units in the hidden layers and output layer are sometimes referred to as **neurodes** or **neurons**, due to their symbolic biological basis, or as **output units**. The multilayer feed-forward neural network shown in Fig. 10.1 has two layers of output units. Therefore we say that it is a **two-layer** neural network. Notice that the input layer is not counted because it serves only to pass the input values to the next layer. In other words, the activation function for an input unit is always an *identity function*:  $O = f(I) = I$ . Similarly, a network containing two hidden layers is called a *three-layer* neural network, and so on. It is a feed-forward network since none of the weights cycles back to an input unit or to a previous layer’s output unit. It is **fully connected** in the sense that each unit provides input to each unit in the next forward layer.

Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer (see Fig. 10.1). It applies a nonlinear activation function to the weighted sum of the inputs. If we know the weights connecting to each layer and the activation function of each unit, we can feed any data tuple into the input layer of the neural network and calculate its output from the output layer, which can be used for classification, clustering, and so on. Multilayer feed-forward neural networks are able to model the class prediction as a nonlinear combination of the inputs. From a statistical point of view, they perform nonlinear regression. Multilayer feed-forward networks with sufficient depth and width, given enough hidden units and enough training samples, can closely approximate any function. In a multilayer feed-forward network, the output of a hidden unit is used as the input to units in the next layer. This gives neural network an incredible ability to learn more complicated, often semantically more meaningful features from the simpler ones. Put it in another way, from left to right of Fig. 10.1, the outputs of units at each layer of a multilayer feed-forward neural networks form a hierarchy of learned features at increasingly more complex levels.

Compared with other data mining methods we have seen before, such as logistic regression and Support Vector Machines (SVMs), being able to learn *any* nonlinear function that maps the input data tuple to the output (e.g., the class label) and the automatic feature learning are two major advantages of neural networks. Let us use the classic XOR problem in Fig. 10.3 to further illustrate this. In Fig. 10.3(a), there are four training tuples, each of which is represented by two attributes  $X_1$  and  $X_2$ . Two positive training tuples are at (0, 1) and (1, 0), respectively, and two negative training tuples are at (0, 0) and (1, 1), respectively. This training set is linearly inseparable, meaning that there is no linear classifier (e.g., perceptron) that is able to separate the two positive tuples from the two negative ones. However, we can use two perceptrons (the two dashed lines in Fig. 10.3(a)) to separate the positive tuples from the negative tuples. The first perceptron is in the form of  $O_1 = \text{sign}(X_2 - X_1 - 0.5)$ . The second perceptron is in the form of  $O_2 = \text{sign}(X_2 - X_1 + 0.5)$ . By combining these two perceptrons (each of which is a linear classifier) together, we have a nonlinear classifier that perfectly separates the two positive tuples from the two negative tuples; that is, if  $X_2 - X_1 - 0.5 \geq 0$  or  $X_2 - X_1 + 0.5 < 0$ , predict it as a positive tuple; otherwise, predict it as a negative tuple.

This nonlinear classifier can be implemented as a two-layer feed-forward neural network in Fig. 10.3(b) marked by the weights and scalars of each unit. There are two units ( $U_3$  and  $U_4$ ) in the hidden layer, each of which takes the output of  $U_1$  and  $U_2$  (i.e.,  $X_1$  and  $X_2$ ) to build a per-

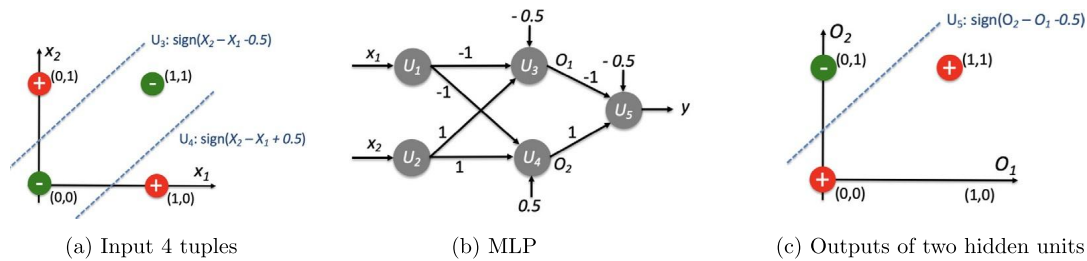


FIGURE 10.3

Solving XOR problem with a two-layer feed-forward neural network with *sign* as the activation function, which is also called multilayer perceptron (MLP). Note that in (c), both negative tuples share the same representation ( $O_1, O_2$ ), both being at (0, 1).

ceptron. In particular, the unit  $U_3$  corresponds to the first perceptron shown in Fig. 10.3(a):  $O_1 = \text{sign}(X_2 - X_1 - 0.5)$ ; and the unit  $U_4$  corresponds to the second perceptron shown in Fig. 10.3(a):  $O_2 = \text{sign}(X_2 - X_1 + 0.5)$ . In the output layer, there is only one unit  $U_5$ , which takes the outputs of the two hidden units ( $O_1$  and  $O_2$ ) to build another perceptron, that is,  $y = \text{sign}(O_2 - O_1 - 0.5)$  (shown as the dashed line in Fig. 10.3(c)). In this way, we are able to separate the two positive tuples from the two negative ones. This neural network has three layers. The input layer has two units ( $U_1$  and  $U_2$ ), each of which just passes through the corresponding input attributes ( $X_1$  and  $X_2$ ) to the hidden layer. In this example, we use the *sign* function as the activation function for all neurons ( $U_3$ ,  $U_4$ , and  $U_5$ ), and this kind of neural network is also called multilayer perceptron (MLP).

In this example, the third perceptron at unit  $U_5$  takes  $O_1$  and  $O_2$  (i.e., the outputs of unit  $U_3$  and  $U_4$ ), as its input features. In other words, both  $O_1$  and  $O_2$  can be viewed as the *learned* features by the two hidden units.  $O_1$  represents whether or not the input tuple is above ( $O_1 = 1$ ) or below ( $O_1 = 0$ ) of the decision boundary of the first perceptron (the upper dashed line in Fig. 10.3(a)). Likewise,  $O_2$  represents whether or not the input tuple is above ( $O_2 = 1$ ) or below ( $O_2 = 0$ ) of the decision boundary of the second perceptron (the lower dashed line in Fig. 10.3(a)). Compared with the original input attributes ( $X_1$  and  $X_2$ ), these two learned features are semantically more meaningful. As we can see from Fig. 10.3(c), the tuples with these two newly learned features are now linearly separable. In contrast, in the original feature space (Fig. 10.3(a)), these training tuples cannot be separated from each other by any linear classifier.

The ability of neural networks to automatically learn features naturally motivates to use neural networks with many hidden layers, namely, **deep neural networks**. Deep neural networks have an incredible capability to learn and represent features at different abstraction levels (e.g., one level at each hidden layer), where the more complicated features are learned based on the simpler ones. Let us look at two examples to elaborate on this. In computer vision, we can build a type of deep neural network called convolutional neural networks (it will be introduced in Section 10.3) to recognize different objects (e.g., car, person, animal) from the input image. The input layer of the neural network contains the raw pixels (e.g., one pixel for each input unit), which represent the feature at the lowest semantic level. The output (i.e., the learned feature from the raw pixels at the input layer) of the first hidden layer might correspond

to edges of the input image.<sup>1</sup> The output (i.e., the learned feature from the edges of the first hidden layer) of the second hidden layer might correspond to contours or corners of the input image. The output (i.e., the learned feature from the edges and contours of the second hidden layer) of the third hidden layer might correspond to parts of the objects (e.g., nose, car wheel) of the input image. Finally, the output layer can easily learn a classifier to recognize different objects based on the object parts at the output layer. For text mining, we can build a type of deep neural network called recurrent neural networks (it will be introduced in Section 10.4) to classify text documents into different categories. The first layer might take the raw characters as input and output the tokens (e.g., words, punctuation); the second layer might take the tokens as input and output phrases; and the final output layer might take the phrases as input and output category that the input document belongs to. While it is very difficult to directly train a classifier (e.g., logistic regression for object recognition or document categorization) based on the raw features of the input data (e.g., the raw pixels of images, the characters of documents), a deep neural network decomposes this task into a sequence of interdependent subtasks (one at each hidden or output layer), each of which learns a semantically more meaningful feature from the feature at the previous layer.

On the one hand, the core algorithmic framework to train a deep neural network, namely **backpropagation** algorithm, has largely remained the same as for the traditional feed-forward neural networks with only a few hidden layers. Therefore we will first introduce this algorithm in Section 10.1.2. On the other hand, it becomes significantly more challenging to train a deep neural network due to its increased number of layers. We will introduce such challenges in Section 10.1.3 and the solutions to handle these challenges in Section 10.2.

Deep learning is closely related to **representation learning**, which aims to automatically learn effective representation (i.e., features, attributes) from the input data to facilitate data mining tasks (e.g., classification, clustering). Notice that the scope of representation learning is broader than deep learning, in that there exist nonneural network methods to automatically learn representation from the input data. An example is principal component analysis (introduced in Chapter 2), where each principal component, a linear combination of the input attributes, can be viewed as a learned new feature.

### 10.1.2 Backpropagation algorithm

For the example in Fig. 10.3, the weights and bias values are given. However, how can we automatically learn such parameters, including the weight vectors and the bias scalars, from the training tuples? A foundational technique is called **backpropagation** algorithm. Backpropagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known *target* value. The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for numeric prediction). For each training tuple, the weights as well as the bias values are modified so as to minimize the difference or the disagreement (e.g., mean-squared loss) between the network's prediction and the actual target value. These modifications are made in the “backward” direction (i.e., from the output layer) through each hidden layer down to the first hidden layer (hence the name *backpropagation*). Although it is not guaranteed, in general, the weights will often eventually converge, when the learning process stops. The algorithm is summarized

---

<sup>1</sup> In computer vision, an edge is where the color or the gray intensity of a group of adjacent pixels suddenly change.



**Algorithm: Backpropagation.** Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

**Input:**

- $D$ , a data set consisting of the training tuples and their associated target values;
- $\eta$ , the learning rate;
- $network$ , a multilayer feed-forward network.

**Output:** A trained neural network (i.e., the weights  $w_{ij}$  and bias  $b_j$  for each hidden or output unit).

**Method:**

```

(1) Initialize all weights and biases in  $network$ ;
(2) while terminating condition is not satisfied {
(3)   for each training tuple  $X$  with target output  $T$  in  $D$  {
(4)     // Propagate the inputs forward:
(5)     for each input layer unit  $j$  {
(6)        $O_j = I_j$ ; // output of an input unit is its actual input value
(7)     for each hidden or output layer unit  $j$  {
(8)        $I_j = \sum_i w_{ij} O_i + b_j$ ; // compute the net input of unit  $j$  with respect to
           the units of the previous layer,  $i$ 
(9)        $O_j = \frac{1}{1+e^{-I_j}}$ ; // compute the output of each unit  $j$  (sigmoid activation)
(10)    // Backpropagate the errors:
(11)    for each unit  $j$  in the output layer
(12)       $\delta_j = O_j(1 - O_j)(T_j - O_j)$ ; // compute the error (mean-squared loss)
(13)    for each unit  $j$  in the hidden layers, from the last to the first hidden layer
(14)       $\delta_j = O_j(1 - O_j)(\sum_l \delta_l w_{jl})$ ; // compute the error with respect to
           the next higher layer,  $l$ 
(15)    for each weight  $w_{ij}$  in  $network$  {
(16)       $\Delta w_{ij} = \eta \delta_j O_i$ ; // weight increment
(17)       $w_{ij} = w_{ij} - \Delta w_{ij}$ ; // weight update
(18)    for each bias  $b_j$  in  $network$  {
(19)       $\Delta b_j = \eta \delta_j$ ; // bias increment
(20)       $b_j = b_j - \Delta b_j$ ; // bias update
(21)    } }

```

**FIGURE 10.4**

Backpropagation algorithm.

in Fig. 10.4. The steps involved are expressed in terms of inputs  $I_i$ , outputs  $O_i$ , and errors  $\delta_i$  and may seem awkward if this is your first look at neural network learning. However, once you become familiar with the process, you will see that each step is inherently simple. The steps are described next. Note that the algorithm described in Fig. 10.4 uses the mean-squared loss and sigmoid activation function. We can generalize the algorithm in Fig. 10.4 with other types of loss functions or activation functions.

**Initialize the weights:** The weights in the network are initialized to small random numbers (e.g., ranging from  $-1.0$  to  $1.0$  or  $-0.5$  to  $0.5$ ). Each unit has a *bias* associated with it, which are similarly initialized to small random numbers. An important aspect for initializing the model parameters is to *break symmetry*. For the example in Fig. 10.5, both  $U_3$  and  $U_4$  are connected to the same input units ( $U_1$  and  $U_2$ ). Therefore it is critical to make sure that the initial weights for  $U_3$  are different from those for  $U_4$ , i.e.,  $w_{13} \neq w_{14}$  and  $w_{23} \neq w_{24}$ . Otherwise,  $U_3$  and  $U_4$  become *unidentifiable*; that is, their weights will be updated in the same way and thus will always be the same. Other than random initial-

ization, in the next section, we will introduce an effective strategy, called “pretraining,” which presets the weights and scalars in a certain way.

Each training tuple,  $\mathbf{X}$ , has a target output  $T$ , which could be the actual class label for the classification task or the numerical value for the regression task. It is processed by the following steps. See Fig. 10.5 for a pictorial illustration.

**Propagate the inputs forward:** First, the training tuples are fed to the network’s input layer. The inputs pass through the input units, unchanged. That is, for an input unit,  $j$ , its output,  $O_j$ , is equal to its input value,  $I_j$ . Next, the net input and output of each unit in the hidden and output layers are computed. The net input to a unit in the hidden or output layers is computed as a linear weighted sum of its inputs. To help illustrate this point, a hidden layer or output layer unit is shown in Fig. 10.2. Each such unit has a number of inputs to it that are, in fact, the outputs of the units connected to it in the previous layer. Each connection has a weight. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit,  $j$  in a hidden or output layer, the net input,  $I_j$ , to unit  $j$  is

$$I_j = \sum_i w_{ij} O_i + b_j, \quad (10.1)$$

where  $w_{ij}$  is the weight of the connection from unit  $i$  in the previous layer to unit  $j$ ;  $O_i$  is the output of unit  $i$  from the previous layer; and  $b_j$  is the bias of the unit. The bias acts as a threshold to adjust the net input of the unit.<sup>2</sup>

Each unit in the hidden and output layers takes its net input and then applies an activation function to it, as illustrated in Fig. 10.2. The function symbolizes the activation of the neuron represented by the unit. Here, the sigmoid function is used. (Recall that in Chapter 6, we have used the sigmoid function to train the logistic regression classifier.) Given the net input  $I_j$  to unit  $j$ , then  $O_j$ , the output of unit  $j$ , is computed as

$$O_j = \frac{1}{1 + e^{-I_j}}. \quad (10.2)$$

This function is also referred to as a *squashing function*, because it maps a large input domain onto the smaller range of 0 to 1. The sigmoid function is nonlinear and differentiable, allowing the backpropagation algorithm to model classification problems that are linearly inseparable.

We compute the output values,  $O_j$ , for each hidden layer, up to and including the output layer, which gives the network’s prediction. In practice, it is a good idea to cache (i.e., save) the intermediate output values at each unit as they are required again later when backpropagating the error. This trick can substantially reduce the amount of computation required.

**Backpropagate the error:** The error is propagated backward to reflect the accuracy of the current network’s prediction, which is in turn used to update the weights and biases. For a unit  $j$  in the output layer, the error  $\delta_j$  is computed by

$$\delta_j = O_j(1 - O_j)(O_j - T_j), \quad (10.3)$$

<sup>2</sup> To see this, we can conceptually think of the bias  $b_j$  as the bias scalar in a linear regression model. In other words, it tells the default (i.e., biased) value of the net input  $I_j$ , without any output from the previous layer (i.e.,  $O_i = 0$ ).

where  $O_j$  is the actual output of unit  $j$ , and  $T_j$  is the known target value of the given training tuple. Note that for some data mining tasks, such as multiclass classification, there are multiple output units (one unit for each class label), and the index  $j$  is used for different output units. For simpler tasks (e.g., binary classification, regression), there is only one unit in the output layer. In that case, the index  $j$  can be omitted. In Eq. (10.3),  $O_j(1 - O_j)$  is the derivative of the sigmoid function with respect to the net input, i.e.  $\frac{\partial O_j}{\partial I_j} = O_j(1 - O_j)$ .

To compute the error of a hidden layer unit  $j$ , the weighted sum of the errors of the units connected to unit  $j$  in the next higher layer is considered. The error of a hidden layer unit  $j$  is

$$\delta_j = O_j(1 - O_j) \left( \sum_l \delta_l w_{jl} \right), \quad (10.4)$$

where  $w_{jl}$  is the weight of the connection from unit  $j$  to unit  $l$  in the next higher layer,  $\delta_l$  is the error of unit  $l$ , and  $O_j(1 - O_j)$  is the derivative of the sigmoid function with respect to the net input, i.e.  $\frac{\partial O_j}{\partial I_j} = O_j(1 - O_j)$ .

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where  $\Delta w_{ij}$  is the change in weight  $w_{ij}$ .

$$\Delta w_{ij} = \eta \delta_j O_i. \quad (10.5)$$

$$w_{ij} = w_{ij} - \Delta w_{ij}. \quad (10.6)$$

“What is  $\eta$  in Eq. (10.5)?”  $\eta$  is the **learning rate**, a constant typically having a value between 0.0 and 1.0. Backpropagation uses using a gradient descent method to search for a set of weights that fits the training data with the goal of minimizing the mean-squared loss between the network’s prediction and the known target value of the tuples.<sup>3</sup> The learning rate helps avoid getting stuck at a local minimum in the decision space (i.e., where the weights appear to converge but are not the optimum solution) and encourages finding a high-quality solution. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to  $1/t$ , where  $t$  is the number of iterations through the training set so far. We will talk more about the learning rate  $\eta$  in Section 10.2.2.

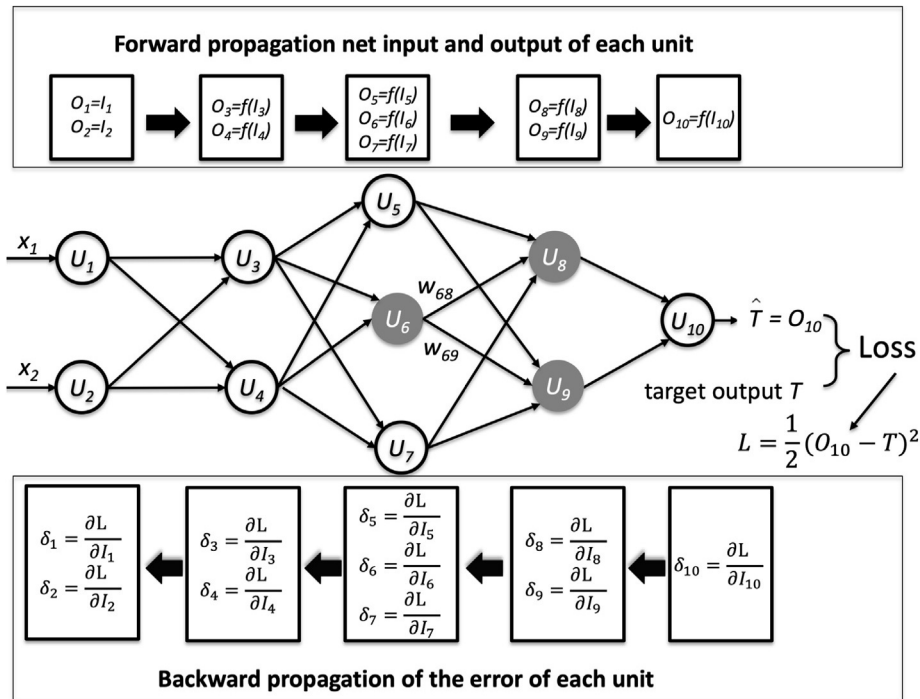
Biases are updated by the following equations, where  $\Delta b_j$  is the change in bias  $b_j$ :

$$\Delta b_j = \eta \delta_j. \quad (10.7)$$

$$b_j = b_j - \Delta b_j. \quad (10.8)$$

For a given tuple  $X$  with the target value  $T$ , let  $\hat{T}$  be the predicted output of  $X$ . The predicted value  $\hat{T}$  is the output of the output unit, which is a (complicated) function of the weights and bias scalars. For the algorithm in Fig. 10.4, we use the mean-squared loss  $L$  to measure the disagreement between the predicted and actual target values of training tuple  $X$ :  $L = \frac{1}{2}(T - \hat{T})^2$ . In some applications (e.g., multiclass classification), the training tuple  $X$  has multiple target values. In this case, both  $T$  and  $\hat{T}$  are vectors:  $T = (T_1, T_2, \dots, T_C)$  and  $\hat{T} = (\hat{T}_1, \hat{T}_2, \dots, \hat{T}_C)$ . The mean-squared loss can be defined in a similar way, that is,  $L = \frac{1}{2} \sum_{j=1}^C (T_j - \hat{T}_j)^2$ . Mathematically, the error term  $\delta_i$  in Eq. (10.3) and

<sup>3</sup> The gradient descent method was used for training logistic regression classifier in Chapter 7.

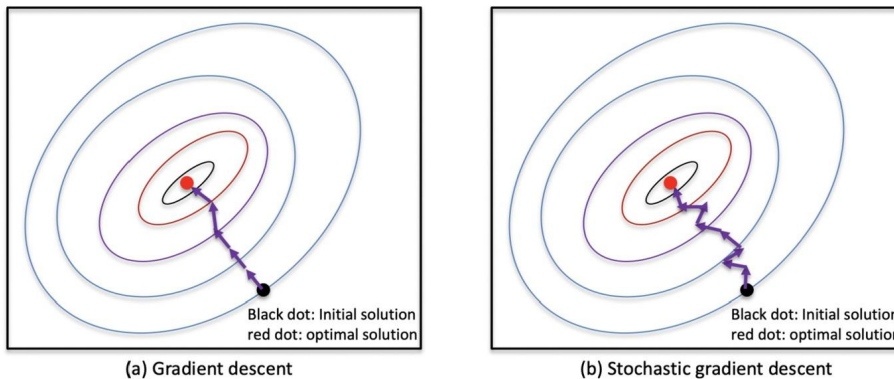


**FIGURE 10.5**

An illustration of the backpropagation algorithm. Given a training tuple, the algorithm first forward propagates the net input and output of each unit (the upper part of the figure). Then, it backward propagates the error of each unit (the bottom part of the figure). The output  $O_{10}$  of unit 10 gives the predicted target value  $\hat{T} = O_{10}$ , which is a function of the weights and bias scalars. The goal of backpropagation is to adjust the weights and bias scalars so that the predicted output matches the actual target output  $T$  as well as possible. That is, to minimize the loss  $L = \frac{1}{2}(\hat{T} - T)^2$ . The error of a given unit can be recursively computed based on the errors of the units it connects to in the next higher layer. For example, the error of unit 6 can be computed based on the errors of unit 8 and unit 9 (shaded in the figure). That is,  $\delta_6 = O_6(1 - O_6)(w_{68}\delta_8 + w_{69}\delta_9)$ . For clarity, the bias scalar or the weights between other units are not shown in the figure. Note that we actually do not need to calculate  $\delta_1 = \delta_2 = 0$ , since the input units ( $U_1$  and  $U_2$ ) just pass through the input attributes:  $O_1 = I_1 = X_1$  and  $O_2 = I_2 = X_2$ .

Eq. (10.4) is the derivative of the loss  $L$  with respect to the net input of unit  $i$ :  $\delta_i = \frac{\partial L}{\partial I_i}$ . Accordingly, in Eq. (10.6) and Eq. (10.8), we update the weights and bias based on gradient descent. The intuition is that we want to update the weights and bias so that the loss  $L$  will decrease most. In some literature, the error term  $\delta_i$  is defined as the derivative of the loss  $L$  with respect to the output of unit  $i$ :  $\delta_i = \frac{\partial L}{\partial O_i}$ . One can develop a similar algorithm as in Fig. 10.4 with this definition, although some mathematical details (e.g., the equations for updating the error terms) will be different. Fig. 10.5 presents an illustration of backpropagation algorithm.

Each iteration in the `while` loop of Fig. 10.4 is called an **epoch**. Note that in Fig. 10.4, we update the weights and bias values after the presentation of each tuple. Alternatively, the weight and bias



**FIGURE 10.6**

Comparison of gradient descent and stochastic gradient descent. Each ellipse is a contour of the function (e.g., the loss function in a neural network) to minimize. Gradient descent (the purple (mid gray in print version) arrows in (a)) finds the best direction to decrease the objective function value. It needs less epochs to find the optimal solution. However, in each epoch, it needs to use all the training tuples to find the best direction (i.e., the gradient). Stochastic gradient descent (the purple (mid gray in print version) arrows in (b)) finds a direction that approximates best direction to decrease the loss function value. It needs more epochs to find the optimal solution. However, in each epoch, it only needs a mini-batch of training tuples. Overall, stochastic gradient descent is often much more computationally efficient than gradient descent.

increments ( $\Delta w_{ij}$  and  $\Delta b_j$ ) could be accumulated in variables, so that the weights and biases are updated after all the tuples (called **full batch**) in the training set have been presented. In theory, the mathematical derivation of backpropagation employs the latter strategy, since the gradients computed in this way give the best direction to reduce the disagreement (or loss) between the actual target values and predicted values of training tuples. In practice, we often use another strategy called **stochastic gradient descent**, which works as follows. In each epoch, we randomly and independently sample a small number of training tuples (called **mini-batch**). The weight and bias increments of each sampled training tuple are accumulated, that is,  $\Delta b_j = \eta \sum_k \delta_j^k$  and  $\Delta w_{ij} = \eta \sum_k \delta_j^k O_j^k$ , where  $k$  is the index of sampled mini-batch. Compared with the standard gradient descent, which needs all training tuples to compute the exact gradient in one epoch, stochastic gradient descent needs more epochs to terminate, but it only needs a small number of sampled tuples to calculate an estimated gradient in each epoch. The benefit of being able to quickly estimate the gradient at each epoch often outweighs the fact that stochastic gradient descent needs more epochs. Therefore the overall running time of stochastic gradient descent is often much smaller than the standard gradient descent. See Fig. 10.6 for an illustration and comparison.

**Terminating condition:** Training stops when

- All  $\Delta w_{ij}$  in the previous epoch are so small as to be below some specified threshold, or
- The percentage of tuples misclassified in the previous epoch is below some threshold, or
- A prespecified number of epochs has expired.

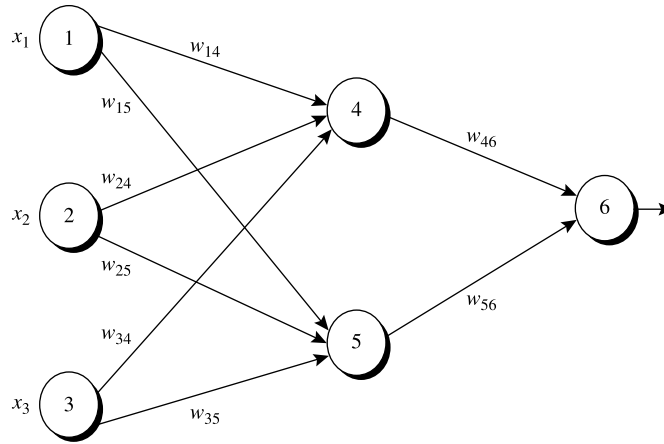


FIGURE 10.7

Example of a multilayer feed-forward neural network.

| $x_1$ | $x_2$ | $x_3$ | $w_{14}$ | $w_{15}$ | $w_{24}$ | $w_{25}$ | $w_{34}$ | $w_{35}$ | $w_{46}$ | $w_{56}$ | $b_4$ | $b_5$ | $b_6$ |
|-------|-------|-------|----------|----------|----------|----------|----------|----------|----------|----------|-------|-------|-------|
| 1     | 0     | 1     | 0.2      | -0.3     | 0.4      | 0.1      | -0.5     | 0.2      | -0.3     | -0.2     | -0.4  | 0.2   | 0.1   |

In practice, several hundreds of thousands of epochs may be required before the weights will converge.

“How efficient is backpropagation?” The computational efficiency depends on the time spent training the network. Given  $|D|$  tuples and  $w$  weights, each epoch requires  $O(|D| \times w)$  time. However, in the worst-case scenario, the number of epochs can be exponential in  $n$ , the number of inputs. In practice, the time required for the networks to converge is highly variable. A number of techniques exist that help speed up the training time. We will introduce some key algorithmic techniques to accelerate the computation in Section 10.2.

**Example 10.1. Sample calculations for learning by the backpropagation algorithm.** Fig. 10.7 shows a multilayer feed-forward neural network. Let the learning rate  $\eta = 0.9$ . The initial weight and bias values of the network are given in Table 10.1, along with the first training tuple,  $X = (1, 0, 1)$ , with a class label of 1.

This example shows the calculations for backpropagation, given the first training tuple,  $X$ . The tuple is fed into the network, and the net input and output of each unit are computed. These values are shown in Table 10.2. The error of each unit is computed and propagated backward. The error values are shown in Table 10.3. The weight and bias updates are shown in Table 10.4. □

“How can we classify an unknown tuple using a trained network?” To classify an unknown tuple,  $X$ , the tuple is input to the trained network, and the net input and output of each unit are computed. There is no need for computation or backpropagation of the error. If there is one output node per class, then the output node with the highest value determines the predicted class label for  $X$ . This is called  $\text{Max}_{\text{out}}$  in the sense that the maximum output value of all output units  $O^i (i = 1, \dots, C)$  is used as the

**Table 10.2** Net input and output calculations.

| Unit, $j$ | Net Input, $I_j$                              | Output, $O_j$               |
|-----------|-----------------------------------------------|-----------------------------|
| 4         | $0.2 + 0 - 0.5 - 0.4 = -0.7$                  | $1/(1 + e^{0.7}) = 0.332$   |
| 5         | $-0.3 + 0 + 0.2 + 0.2 = 0.1$                  | $1/(1 + e^{-0.1}) = 0.525$  |
| 6         | $(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$ | $1/(1 + e^{0.105}) = 0.474$ |

**Table 10.3** Calculation of the error at each node.

| Unit, $j$ | Error, $\delta_j$                            |
|-----------|----------------------------------------------|
| 6         | $(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$     |
| 5         | $(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$ |
| 4         | $(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$ |

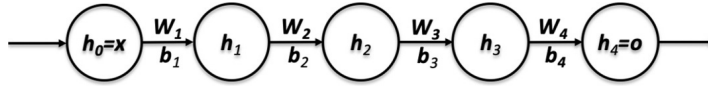
**Table 10.4** Calculations for weight and bias updating.

| Weight or Bias | New Value                              |
|----------------|----------------------------------------|
| $w_{46}$       | $-0.3 - (0.9)(0.1311)(0.332) = -0.339$ |
| $w_{56}$       | $-0.2 - (0.9)(0.1311)(0.525) = -0.262$ |
| $w_{14}$       | $0.2 - (0.9)(-0.0087)(1) = 0.208$      |
| $w_{15}$       | $-0.3 - (0.9)(-0.0065)(1) = -0.294$    |
| $w_{24}$       | $0.4 - (0.9)(-0.0087)(0) = 0.4$        |
| $w_{25}$       | $0.1 - (0.9)(-0.0065)(0) = 0.1$        |
| $w_{34}$       | $-0.5 - (0.9)(-0.0087)(1) = -0.492$    |
| $w_{35}$       | $0.2 - (0.9)(-0.0065)(1) = 0.206$      |
| $b_6$          | $0.1 - (0.9)(0.1311) = -0.180$         |
| $b_5$          | $0.2 - (0.9)(-0.0065) = 0.206$         |
| $b_4$          | $-0.4 - (0.9)(-0.0087) = -0.392$       |

network's prediction. Alternatively, we can use `SoftMax` to convert the outputs  $O^i$  ( $i = 1, \dots, C$ ) to the probabilities that the input tuple belongs to different class:  $\frac{e^{O_i}}{\sum_{i=1}^C e^{O_i}}$ . Here,  $O_i$  is the output of the  $i$ th output unit, and  $C$  is the total number of output units (e.g., the number of classes). If there is only one output node, then an output value greater than or equal to 0.5 may be considered as belonging to the positive class, while a value less than 0.5 may be considered negative.

### Matrix form representation of feed-forward neural networks

For an  $L$ -layer feed-forward neural network, we can alternatively represent it in a more concise form using matrix representation. To be specific, each data tuple is represented as an  $n_0$ -dimensional vector of attributes  $\mathbf{X} = (x_1, \dots, x_{n_0})$  where  $n_0$  is the number of attributes. Suppose each hidden or output layer has  $n_i$  units. We denote the output of units at each layer as a vector  $\mathbf{h}_i$  of length  $n_i$  ( $i = 0, \dots, L$ ),  $\mathbf{h}_0 = \mathbf{X}$ , and  $\mathbf{h}_L$  contains the output of all output units. We denote all the weights connecting layer  $(i - 1)$  to layer  $i$  as an  $n_{i-1} \times n_i$  weight matrix  $\mathbf{W}_i$  and all the bias values at a hidden or output layer as an  $n_i$ -



**FIGURE 10.8**

The matrix form representation of feed-forward neural network in Fig. 10.5.  $\mathbf{h}_0$  represents the input units (i.e.,  $\mathbf{h}_0 = \mathbf{x}$ ) and  $\mathbf{h}_4$  represents the output units (i.e.,  $\mathbf{h}_4 = \mathbf{o}$ ). For  $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ , each of them represents the hidden units at the corresponding layer.

dimensional vector  $\mathbf{b}_i$  ( $i = 1, \dots, L$ ). Then, each hidden or output layer essentially transforms the vector at the previous layer  $\mathbf{h}_{i-1}$  to another vector  $\mathbf{h}_i$ , that is,  $\mathbf{h}_i = f(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i)$ , where the activation function  $f(\cdot)$  operates element-wisely on a vector. What is inside the  $f(\cdot)$  function ( $\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i$ ) is called an *affine transformation*.<sup>4</sup> In a feed-forward neural network, each hidden or output unit is connected to *all* units in the previous layer. We call such (hidden or output) layers as *fully connected*. With such a matrix form representation, we represent an  $L$ -layer feed-forward neural network by a *chain graph* with length of  $L$ .

**Example 10.2.** For the four-layer neural network in Fig. 10.5, its equivalent matrix form representation is shown in Fig. 10.8, which is a chain graph of length 4. Since there are two input units in Fig. 10.5,  $\mathbf{h}_0$  is a vector of length 2. Likewise,  $\mathbf{h}_1$  is a vector of length 2 (two hidden units in the first hidden layer),  $\mathbf{h}_2$  is a vector of length 3 (three hidden units in the second hidden layer),  $\mathbf{h}_3$  is a vector of length 2 (two hidden units in the third hidden layer), and  $\mathbf{h}_4$  is a vector of length 1 (i.e., it is a scalar, since there is only one output unit in Fig. 10.5). The bias vector  $\mathbf{b}_i$  has the same length as the corresponding  $\mathbf{h}_i$  ( $i = 1, 2, 3, 4$ ). The size of the weight matrix  $\mathbf{W}_i$  is determined by the length of  $\mathbf{h}_{i-1}$  (number of rows) and the length of  $\mathbf{h}_i$  ( $i = 1, 2, 3, 4$ ) (number of columns). For instance, since both  $\mathbf{h}_0$  and  $\mathbf{h}_1$  have length 2, the weight matrix  $\mathbf{W}_1$  is of size  $2 \times 2$ ; since  $\mathbf{h}_2$  has length 3 but  $\mathbf{h}_3$  has length 2, the weight matrix  $\mathbf{W}_2$  is of size  $3 \times 2$ .  $\mathbf{h}_i$  is obtained by a nonlinear activation function  $f(\cdot)$  on the affine transformation of  $\mathbf{h}_{i-1}$  with the weight matrix  $\mathbf{W}_i$  and the bias vector  $\mathbf{b}_i$  ( $i = 1, 2, 3, 4$ ).  $\square$

Most deep learning models are trained in the matrix form, where a major computational bottleneck lies in matrix multiplication<sup>5</sup> in both forward and backward propagation stages. In order to accelerate this process, deep learning models are often trained with GPUs (graphics processing units) instead of CPUs (central processing unit). Compared with CPUs, GPUs offer two major advantages, which are critical in training deep learning models. First, GPUs are *bandwidth optimized*, which means that GPUs are good at fetching a large amount of memory needed for matrix multiplication. Second, optimizing the bandwidth comes at the potential cost of a high memory access latency. GPUs bypass this issue (memory access latency) using a technique called *thread parallelism*.

<sup>4</sup> Mathematically, an affine transformation is a linear method that changes one vector to another vector via translation, scale, shear, or rotation.

<sup>5</sup> Other computationally intensive components in training deep learning models include *convolution*, the key operation in a type of deep learning model called convolutional neural networks, which will be introduced in Section 10.3.



### 10.1.3 Key challenges for training deep learning models

As mentioned before, the key algorithmic framework to train a deep neural network, namely the backpropagation algorithm in Fig. 10.4, has largely remained the same since the 1980s. Let us first take a closer look at the backpropagation from the optimization perspective to identify the main algorithmic challenges.

Given a set of  $m$  training tuples  $\{(X^1, T^1), \dots, (X^m, T^m)\}$ , where  $X^l$  and  $T^l$  are the input attribute vector and target value of the  $l$ th tuple ( $l = 1, \dots, m$ ), respectively. We are given a feed-forward neural network with fixed architecture, including the number of layers, the number of units at each layer, and the activation function of each unit. However, we do not know the weights ( $w_{ij}$ ) and bias values ( $b_j$ ) of the network. Backpropagation learns such model parameters by adaptively adjusting them to minimize the (approximate) training error  $E$

$$E(\theta) = \frac{1}{m} \sum_{l=1}^m \text{Loss}(\hat{T}(X^l, \theta), T^l), \quad (10.9)$$

where  $\text{Loss}()$  is the loss function for an individual tuple, such as mean-squared loss in Fig. 10.4;  $\theta$  represents all the model parameters the algorithm aims to learn, including all the weights  $w_{i,j}$  and bias values  $b_j$ ;  $\hat{T}(X^l, \theta)$  is the predicted target value for the  $l$ th tuple that is a (complicated) function of the model parameter  $\theta$ . The overall training error  $E(\theta)$  is the average loss among all  $m$  training tuples. Backpropagation starts with some initial guess of the model parameters  $\theta_0$  and then iteratively updates them as the following equation in order to minimize the training error  $E(\theta)$ , until the algorithm terminates,

$$\theta_{t+1} = \theta_t - \eta g_t, \quad (10.10)$$

where  $t$  is the epoch number,  $\eta$  is the learning rate, and  $g_t$  is the gradient of the training error  $E(\theta)$  w.r.t. the model parameters  $\theta$ . In the full-batch gradient descent approach,  $g_t$  is calculated by all  $m$  training tuples; in the stochastic gradient descent approach,  $g_t$  is estimated by a mini-batch of training tuples. When the algorithm terminates, we use the  $\theta^*$  from the final epoch as the learned model parameters, including all the weights ( $w_{ij}$ ) and the bias values  $b_j$ .

The first challenge (**optimization**) we face is how to make sure that  $\theta^*$  is indeed a high-quality solution to minimize  $E(\theta)$  in Eq. (10.9). If the objective function we wish to minimize is convex (e.g., the one in Fig. 10.6), then the gradient descent in Eq. (10.10) guarantees that  $\theta^*$  is indeed the optimal solution regardless the initial solution  $\theta_0$ . With stochastic gradient descent,  $\theta_t$  will converge to the optimal solution in the probabilistic sense. However, the training error  $E(\theta)$  in a deep neural network is almost always *nonconvex*. As illustrated in Fig. 10.9, the (stochastic) gradient descent in Eq. (10.10) becomes much more challenging for a nonconvex function, even if there is only a single variable to optimize. Depending on where we start the search (i.e., the initial choice of  $\theta_0$ ), the algorithm might end up in a high-cost local minimal; it might be stuck in a plateau where the gradient  $g_t$  is almost 0; on the contrast the algorithm might jump over a desirable search area near a “cliff” where the gradient  $g_t$  is very large; or the algorithm might be deceived to stop at a saddle point where the gradient is zero but it neither a local minimum nor maximum. Note that in practice, due to the high complexity of the training error  $E(\theta)$  of a deep neural network, it is neither realistic nor necessary to search for its global minimal. Instead, most algorithms aim to find a *high-quality local minimum*, that is, a local minimal with a low training error  $E(\theta)$  in a computationally efficient way.

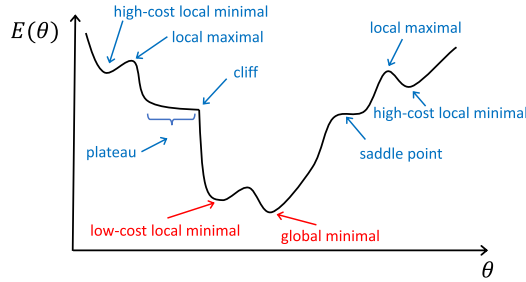


FIGURE 10.9

An illustration of the optimization challenge for training deep neural networks due to nonconvexity.

The second challenge (**generalization**) we face is overfitting.  $E(\theta)$  in Eq. (10.9) is the (approximated) training error. However, what we really want is to minimize the *generalization error*, that is, the classification error on future, unseen test tuples. From what we have learned from Chapter 6, we could end up with a deep neural network model whose parameters indeed minimize the training error  $E(\theta)$ , but the learned network performs poorly on test tuples. Such overfitting is likely to happen, especially given the high complexity of deep neural networks with many layers and model parameters and a limited amount of labeled training tuples.

In Section 10.2, we will learn some key algorithmic techniques to address these two challenges.

### 10.1.4 Overview of deep learning architecture

For an  $L$ -layer feed-forward neural network, it is primarily designed for *multidimensional* data. That is, each data tuple is represented as an  $n_0$ -dimensional vector of attributes  $\mathbf{X} = (x_1, \dots, x_{n_0})$  where  $n_0$  is the number of attributes. In addition to feed-forward neural networks, there are many kinds of deep neural networks, which are often designed for other types of input data, including convolutional neural networks (CNNs) for *grid-like* data, recurrent neural networks (RNNs) for *sequence* data, and graph neural networks (GNNs) for *relational* (i.e., graph) data. Table 10.5 gives an overview of these

Table 10.5 An overview of typical deep learning architectures.

| Data Type       | Multidimensional                                                                              | Grid    | Sequence                                | Graph   |
|-----------------|-----------------------------------------------------------------------------------------------|---------|-----------------------------------------|---------|
|                 | Features: credit rating, account balance<br>$x = (4.5, 500, 3, 5)$<br>#deposits, #withdrawals |         | $x = "I \text{ love watching movies.}"$ |         |
| DL Architecture | Feed-forward Network<br>                                                                      | CNN<br> | RNN<br>                                 | GNN<br> |

typical deep learning architectures. Details of these deep learning architectures will be introduced in Sections 10.3 (CNNs), 10.4 (RNNs), and 10.5 (GNNs), respectively.

## 10.2 Improve training of deep learning models

In this section, we introduce some key algorithmic techniques to address the two challenges (i.e., optimization and generalization) outlined in Section 10.1.3.

### 10.2.1 Responsive activation functions

For the backpropagation algorithm in Fig. 10.4, we have used the sigmoid function  $\sigma(I)$  as the activation function, where  $O = \sigma(I) = \frac{1}{1+e^{-I}} \in (0, 1)$  and its derivative is  $\frac{\partial O}{\partial I} = O(1 - O)$ . The error  $\delta_j$  for an output unit  $j$  is calculated as  $\delta_j = O_j(1 - O_j)(T_j - O_j)$ , where  $T_j$  and  $O_j$  are the actual and predicted target values for a training tuple, respectively. We can see that if  $O_j$  is close to either 1 or 0,  $O_j(1 - O_j)$  will be close to 0, which will in turn make the error  $\delta_j \approx 0$ . From Line 16 and Line 18 of Fig. 10.4, we can see that the weights increment  $\Delta w_{ij}$  and the bias increment  $\Delta b_j$  will be close to 0, and thus the weights and bias values will almost remain same in this epoch. Since  $\sigma(O) \in (0, 1)$ , in either case ( $O_j \approx 1$  or  $O_j \approx 0$ ), the unit is “saturated” in that its output is approaching one of its limited values. Put it in another way, when the output unit is saturated, the gradient  $\mathbf{g}_t$  in Eq. (10.10) is close to zero, and thus (stochastic) gradient descent procedure makes very small progress in updating the model parameters  $\theta$  or is even stuck (i.e.,  $\theta_{t+1} \approx \theta_t$ ). See the plateau in Fig. 10.9 for an illustration.

This issue could be further exacerbated due to the backpropagation of the error from the output layer to the hidden layers. From Line 14 of Fig. 10.4, we can see that the error  $\delta_j$  of a hidden unit  $j$  might quickly approach zero due to the recursive multiplication of a set of small numbers. That is, in order to calculate the error for a hidden unit  $j$ , we first aggregate errors of units in the next higher layer that unit  $j$  connects to (i.e.,  $\sum_l \delta_l w_{jl}$ ), and then decay the aggregation result by the derivative of the sigmoid function of the unit  $j$  (i.e.,  $O_j(1 - O_j)$ ), which is always between 0 and 1. By recursively propagating the error terms backward, the error of a hidden unit, especially those from the lower layers, are likely to approach zero, even if the output unit itself is not saturated. In other words, their gradients *vanish* during the backpropagation process. Consequently, the corresponding model parameters  $\theta$  (the weights and the bias values) will remain almost the same.

An effective way to address the **gradient vanishing** problem is to replace the sigmoid function with alternative, more responsive activation functions that are less likely to be saturated. **Rectified linear unit** (ReLU) is one such example. A ReLU activation function is defined as  $O = f(I) = I$  if  $I > 0$  and  $O = f(I) = 0$  otherwise. In other words, if the net input is negative, a ReLU unit simply outputs 0 (i.e., the unit is inactive); otherwise, it just passes through the same positive net input as its output. Compared with the sigmoid function, the output of the ReLU function will be saturated only on one side when the net input is negative. The derivative of a ReLU function with respect to its net input can be calculated as  $\frac{\partial O}{\partial I} = 1$  if  $I > 0$  and  $\frac{\partial O}{\partial I} = 0$  otherwise.<sup>6</sup>

<sup>6</sup> If the net input  $I = 0$ , the derivative does not exist for the ReLU function. We can simply set it as 0. Mathematically, we can set any number between 0 and 1, known as the subgradient of the unit output w.r.t. the net input when  $I = 0$ .

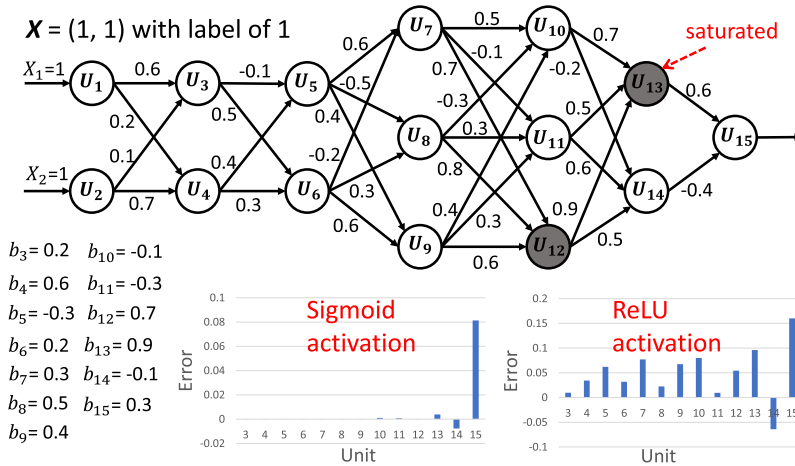


FIGURE 10.10

An illustration of gradient vanishing problem caused by sigmoid activation and how ReLU can avoid it. Weights and biases are initialized as in the figure. For two different hidden unit activation functions (i.e., ReLU and sigmoid), errors  $\delta(i)$  for each unit are summarized in the bar charts. The outputs of units 12 and 13 (i.e.,  $O(12)$ ,  $O(13)$ ), approximate 1 and therefore are saturated. We observe that with sigmoid as the activation, the error decreases dramatically as it backpropagates, whereas the ReLU activation bears comparable errors for different units at different layers.

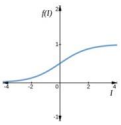
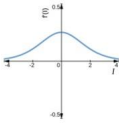
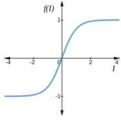
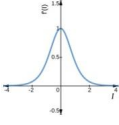
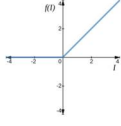
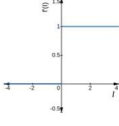
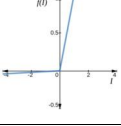
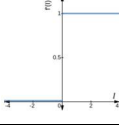
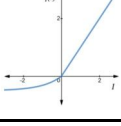
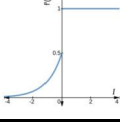
So, what would happen if we use ReLU, instead of the sigmoid function, as the activation function? In a given epoch, if a unit  $j$  is inactive (i.e., its net input is negative and the output is zero), its error term  $\delta_j = 0$  and its parameters (weights and the bias value) will remain the same in this epoch. On the other hand, if the unit  $j$  is active (i.e., its output  $O_j = I_j > 0$ ), since the derivative  $\frac{\partial O_j}{\partial I_j} = 1$ , we simply aggregate all the error terms from the units in the next higher layer that unit  $j$  connects to, without decaying it by a small number (e.g.,  $O_j(1 - O_j)$  with the sigmoid function). In this way, we effectively avoid the gradient vanishing problem, even if we propagate the error terms through many layers in a deep neural network. See Fig. 10.10 for an illustration.

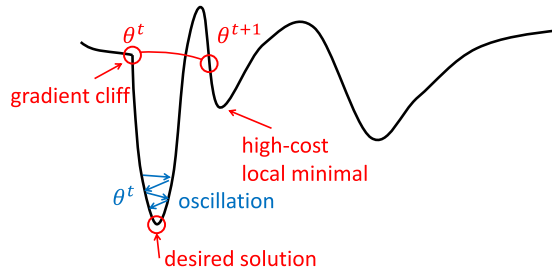
Other than ReLU, several alternative activation functions exist, which are more responsive than the sigmoid function and thus are less likely to encounter gradient vanishing problem. Table 10.6 provides a summary.

### 10.2.2 Adaptive learning rate

For the learning rate  $\eta$  in backpropagation algorithm in Fig. 10.4 (or equivalently  $\eta$  in Eq. (10.10)), if it is too small, the model parameters  $\theta$  might change very slowly, and thus it might take many epochs for the algorithm to terminate. On the other hand, if the learning rate  $\eta$  is too big, the algorithm might “jump” over the desired search region during certain epochs. For example, if the current model parameter  $\theta_j$  is at the edge of the “gradient cliff” in Fig. 10.11, with a large learning rate  $\eta$ , the updated model

**Table 10.6 Summary of common activation functions.**

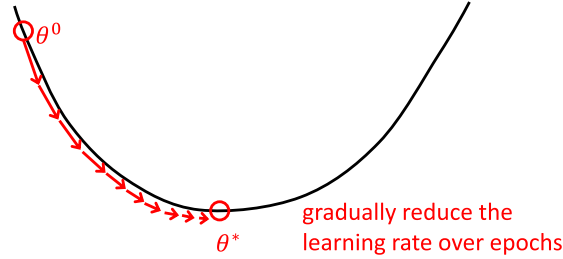
| Name       | Definition ( $f(I)$ )                                             | Plot                                                                              | Derivative of $f(I)$                                         | Plot                                                                               |
|------------|-------------------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------|------------------------------------------------------------------------------------|
| Sigmoid    | $\frac{1}{1+e^{-I}}$                                              |  | $f(I)(1 - f(I))$                                             |  |
| Tanh       | $\frac{e^I - e^{-I}}{e^I + e^{-I}}$                               |  | $1 - f(I)^2$                                                 |  |
| ReLU       | $\begin{cases} 0, I \leq 0 \\ I, I > 0 \end{cases}$               |  | $\begin{cases} 0, I \leq 0 \\ 1, I > 0 \end{cases}$          |  |
| Leaky ReLU | $\begin{cases} 0.01 \times I, I < 0 \\ I, I \geq 0 \end{cases}$   |  | $\begin{cases} 0.01, I < 0 \\ 1, I \geq 0 \end{cases}$       |  |
| ELU        | $\begin{cases} \alpha(e^I - 1), I \leq 0 \\ I, I > 0 \end{cases}$ |  | $\begin{cases} \alpha e^I, I \leq 0 \\ 1, I > 0 \end{cases}$ |  |



**FIGURE 10.11**

An illustration of a large learning rate leading to jump over or oscillate around the desired solution.

parameter  $\theta_{t+1}$  might leap over the desired search region (where a low-cost local minimal or even a global minimal locates) and instead makes the algorithm eventually terminate at a high-cost local minimal. In other cases, a large learning rate might cause the backpropagation algorithm to oscillate around the local minimal and thus take a long time to terminate. Therefore instead of fixing the learning rate



**FIGURE 10.12**

An illustration of adaptive learning rate. The learning rate decreases as the epoch number increases.

during the entire backpropagation algorithm, a more reasonable choice is to use an *adaptive* learning rate  $\eta_t$  whose value changes with respect to the epoch number  $t$ .

A generic strategy is to shrink the learning rate as the algorithm progresses. That is, the larger the epoch/iteration number, the smaller the learning rate. The intuition of such a strategy is as follows. At the beginning of the algorithm, it is likely that the model parameters are far away from the desired solution (i.e., a low-cost local minimal). Therefore we use a larger learning rate so that the algorithm can make bigger progress to update the model parameters. On the other hand, as the algorithm progresses, it is likely that the current model parameters are in the vicinity of the final desired solution. Therefore we use a smaller learning rate to avoid oscillating or even jumping over the desired solution. See Fig. 10.12 for an illustration.

A simple strategy is to set the learning rate  $\eta_t$  to be in the reverse proportion of the epoch number  $t$ . That is, we choose the learning rate as  $\eta_t = \frac{1}{t}\eta_0$ , where  $\eta_0$  is the manually set initial learning rate (e.g.,  $\eta_0 = 0.9$  as in Example 10.1). In practice, it is common to set the learning rate to be a small constant  $\eta_\infty$  (e.g.,  $\eta_\infty = 10^{-9}$ ) after certain number of epochs  $T$  (e.g.,  $T = 10,000$ ), to prevent the learning rate from approaching zero. In this case, the adaptive learning rate  $\eta_t$  can be set as linear interpolation between the initial learning rate  $\eta_0$  and the minimum learning rate  $\eta_\infty$ , that is,  $\eta_t = (1 - \frac{t}{T})\eta_0 + \frac{t}{T}\eta_\infty$  if  $t \leq T$  and  $\eta_t = \eta_\infty$  otherwise. We can see that as the algorithm progresses (i.e.,  $t$  increases), the learning rate will put more weight ( $\frac{t}{T}$ ) on the minimum learning rate  $\eta_\infty$  and thus become smaller, until it reaches certain epoch number  $T$ .

In many cases, the magnitude of the gradient  $\mathbf{g}_t$  in Eq. (10.10) provides an important indicator on the overall progress of the backpropagation algorithm. For a given model parameter  $\theta_i$  (e.g., the weight of the connection between two units), we use the square root of the sum of squared historical gradient values to measure its magnitude:  $r_i = \sqrt{\sum_{k=1}^{t-1} \mathbf{g}_{i,k}^2}$ . The intuition of the magnitude measure  $r_i$  is as follows. The larger the  $r_i$ , the more likely the algorithm has made greater progress about the corresponding model parameter  $\theta_i$  in the previous epochs (i.e., from epoch 1 to epoch  $(t-1)$ ). Therefore we should use a smaller learning rate. **AdaGrad** (which stands for Adaptive Gradient Algorithm) follows this intuition and sets the adaptive learning rate  $\eta_t$  in the reverse proportion of the gradient magnitude as  $\eta_t = \frac{1}{\rho + r_i}\eta_0$ , where  $\rho$  is some small constant (e.g.,  $\rho = 10^{-8}$ ) to prevent the numerical instability in case  $r_i = 0$ . Note that the gradient magnitude  $r_i$  changes with respect to the epoch number, and at the beginning of the algorithm, we can simply set it as zero. See Table 10.7 for an example.

**Table 10.7** Comparison of adaptive learning rate strategies, linear vs. AdaGrad ( $\eta_0 = 0.9$ ,  $\eta_\infty = 10^{-9}$ ,  $T = 1000$ ,  $\rho = 10^{-8}$ ). We observe that, for AdaGrad, larger historical gradient values lead to larger changes in the learning rate, whereas the learning rate for the linear strategy does not depend on the historical gradient.

| Epoch number (t) | Gradient value | Linear learning rate | AdaGrad learning rate |
|------------------|----------------|----------------------|-----------------------|
| 1                | 0.923          | 0.8991               | 0.9751                |
| 2                | 0.831          | 0.8982               | 0.7247                |
| 3                | 0.756          | 0.8973               | 0.6190                |
| 4                | 0.324          | 0.8964               | 0.6042                |
| 5                | 0.517          | 0.8955               | 0.5708                |
| 6                | 0.453          | 0.8946               | 0.5486                |
| 7                | 0.967          | 0.8937               | 0.4726                |
| 8                | 1.153          | 0.8928               | 0.4043                |
| 9                | 1.072          | 0.8919               | 0.3642                |
| 10               | 0.879          | 0.8910               | 0.3432                |

In AdaGrad, the gradient values from historical epochs ( $\mathbf{g}_{i,k}^2$ ,  $k = 1, \dots, (t - 1)$ ) are treated equally when computing the gradient magnitude  $r_i$ . An alternative choice is to put more weight on more recent gradient values. That is, if the gradient values in the more recent epochs are larger, we want to shrink the learning rate  $\eta_t$  more. **RMSProp** (which stands for Root Mean Square Propagation) follows this intuition and uses an exponentially decaying weighted sum of squared historical gradient values to measure the gradient magnitude  $r_i$ . Compared with AdaGrad, RMSProp was found to converge faster for training deep neural networks.

### 10.2.3 Dropout

A major advantage of deep neural networks lies in the ability to learn a hierarchy of features. This, however, could be a double-bladed sword. This is because the high complexity of deep neural network models also gives rise to *overfitting*. This means that the algorithm could have learned some fictitious features, which leads to a small training error but bears a high test error.

A simple yet very effective strategy to prevent the deep neural networks from learning such fictitious features is **dropout**. It works as follows. In a given epoch of the backpropagation algorithm, we first randomly *dropout* or de-activate some nonoutput units by deleting all incoming and outgoing links of the corresponding units. Then, we perform standard forward and backward propagation operations (e.g., those in Fig. 10.4) on the *dropout network*, the network without the de-activated units, to update the model parameters, including the weights and bias values. A de-activated unit has no impact on updating the model parameters in the current epoch. We can think of a de-activated unit as being “frozen.” In other words, the model parameters of a de-activated unit, including its weights and bias scalar, are kept unchanged, until the next epoch that it is not dropped out (i.e., activated). It then updates the “frozen” model parameters in that epoch.

**Example 10.3.** Suppose we want to train a two-layer feed-forward neural network in Fig. 10.13(a). In a given epoch of backpropagation algorithm, we randomly drop a hidden unit  $U_3$  by removing all of

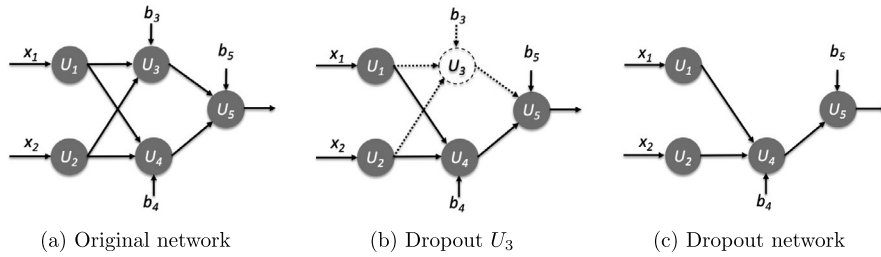


FIGURE 10.13

An illustration of creating a dropout network.

its incoming and outgoing links (dashed lines in Fig. 10.13(b)). By removing the de-activated unit  $U_3$  from the original network, we have a dropout network in Fig. 10.13(c). We update the model parameters on the dropout network by performing forward and backward propagation. In the next epoch, we will create another dropout network to further update the model parameters, and so on. □

Given a deep neural network with  $n$  nonoutput units, there could be an exponential number of dropout networks. For the two-layer feed-forward neural networks, there are nine dropout networks (shown in Fig. 10.14) that the backpropagation algorithm can use to update the model parameters. Note

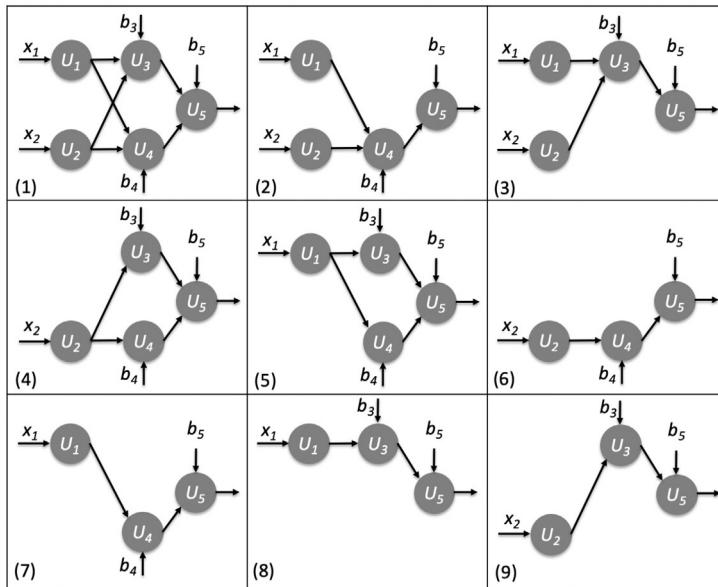


FIGURE 10.14

Dropout networks for the feed-forward neural network in Fig. 10.13.



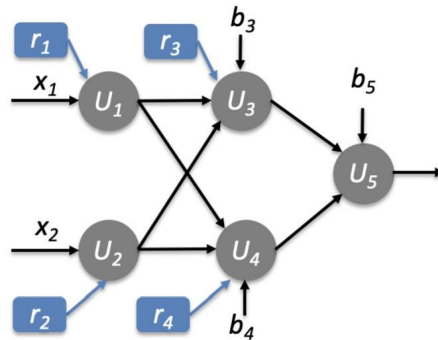


FIGURE 10.15

Dropout networks implementation with binary mask gates  $r_j$  ( $j = 1, 2, 3, 4$ ) for the feed-forward neural network in Fig. 10.13.

that we never drop an output unit. Mathematically, there could be as many as  $2^n$  dropout networks. However, if there is no path from any input unit to an output unit, the backpropagation algorithm will not be able to use it for updating the model parameters, and therefore we just ignore it. For the example in 10.13(a), if we dropout both  $U_3$  and  $U_4$ , the two input units will be disconnected with the output unit  $U_5$ . We simply ignore this dropout network (which consists of  $U_1, U_2$  and  $U_5$ ).

Even if we ignore all disconnected dropout networks, there are still a large number of possible dropout networks that the backpropagation algorithm could use to update the model parameters. In practice, however, we do not need to actually create such dropout networks, as it would be very time-consuming. Instead, we can introduce a *binary mask gate* for each nonoutput unit (the rectangle nodes in Fig. 10.15). At each epoch, each mask gate  $r_j$  ( $j = 1, 2, 3, 4$ ) outputs a Bernoulli random variable. That is,  $r_j$  outputs 1 with a probability of  $\rho$  and it outputs 0 with a probability  $1 - \rho$ , where  $0 < \rho < 1$ ,  $\rho$  is the keep rate, and  $(1 - \rho)$  is the dropout rate. If we set  $\rho = 0.5$ , on average, half of the nonoutput units are dropped at each epoch. The output of each mask gate is multiplied with the net input of each unit. Therefore if the output of a mask gate is 0, it will make the net input of the corresponding unit to be zero and thus make it de-activated (i.e., be dropped from the original network). After the backpropagation algorithm terminates, it is common to scale the final model parameters  $\theta^*$  by  $\rho$  in the test stage  $\theta^* \leftarrow \rho \cdot \theta^*$ , in order to ensure that the model output in the test stage will roughly match the expected output of the final dropout network. For example, if the dropout rate  $\rho = 0.5$ , on average, there are twice as many units in the original network as in each dropout network. By shrinking the model parameters by 0.5 in the test stage, its output is likely to be comparable to the expected output of the final dropout network. An alternative method, called *inverted dropout*, exists that scales the output of a nondropout unit by  $\frac{1}{\rho}$  at each epoch during the training. The intuition is that the lower the keep rate  $\rho$ , the fewer units will be kept active, and the more scaling that is needed.

*Why does dropout work well in practice?* From the optimization perspective, dropout can be viewed as a regularization technique. By randomly dropping some input or hidden units, we force the model to be robust to the random noise, so that the learned features are more likely to generalize well to future test tuples and thus lead to a small test error. We can also make an analogy between dropout and the ensemble methods that were introduced in Section 6.7. For example, in the bagging method, we create

a large number of base models in the following way. Each base model is trained on a bootstrap sample, that is, independently sampled with replacement from the input training tuples. By aggregating (e.g., averaging) the prediction of multiple base models, we can often improve the generalization performance of the ensembled model. Conceptually, we can view each dropout network as a base model, which is obtained by randomly sampling the input and hidden units of the original neural network that we target to learn. By training (e.g., running an epoch of backpropagation algorithm on) each dropout network and aggregating the results over different dropout networks (i.e., base models), we are likely to obtain a better model. That is, using the aggregated weights and bias values as the model parameters of the original neural network, it is likely to generalize well. It is worth pointing out there is a subtle and important difference between the two. In bagging, each base model is trained *independently* on a separate bootstrap sample; whereas in dropout, the model parameters of the current dropout network are updated based on that of the previous dropout network(s).

### 10.2.4 Pretraining

Due to the nonconvexity of the objective function that a deep learning model aims to minimize (Eq. (10.9)), the initial model parameter  $\theta_0$  could have a significant impact on the quality of its solution. As we can see from Fig. 10.9, if the initial model parameters are located in the *suitable region* (e.g., area between the cliff and saddle point in Fig. 10.9), using backpropagation algorithm, we will eventually end up with either a low-cost local minimal or even a global minimal. However, if the initial model parameters are not in such a suitable region, we could either end up with a high-cost local minimum, or the algorithm could be stuck at a plateau or a saddle point. **Pretraining** refers to the process of initializing the model parameters (the weights and bias values of a feed-forward neural network) in a suitable region.

An effective pretraining approach is called *greedy supervised pretraining*, which aims to preset the model parameters layer-by-layer in a greedy way. Its general strategy is as follows. Instead of training a complex deep learning model with many layers directly, we train some simpler models first (e.g., the neural network with much fewer layers). We can then use the learned parameters of the simpler models as a hint to help train the original complex model.

**Example 10.4.** Let us use a four-layer feed-forward neural network in Fig. 10.16(a) as an example to explain how greedy supervised pretraining works. Note that in Fig. 10.16, the bias values are not shown for clarity. Instead of directly finding the model parameters (weights and bias values) of all units from all noninput layers, we aim to pretrain the network in an iterative way, with the goal of pretraining the model parameters for one hidden layer at each iteration. In the first iteration, we focus on a simple model, with the two input units ( $U_1$  and  $U_2$ ) and one output unit ( $U_{10}$ ), but only the hidden units from the first hidden layer ( $U_3$  and  $U_4$ ). For this simple model (Fig. 10.16(b)), we run the backpropagation algorithm to find its parameters, including the weights and bias values for the hidden units ( $U_3$  and  $U_4$ ) and those for the output unit ( $U_{10}$ ). For the learned model parameters for the output unit  $U_{10}$ , we ignore them (denoted as the dashed lines in Fig. 10.16(b)). Then, in the second iteration, we increase the model complexity by adding another hidden layer, including the three hidden units ( $U_5$ ,  $U_6$ , and  $U_7$ ) from the second hidden layer of the original network in Fig. 10.16(a). In this way, we have a three-layer feed-forward neural network (Fig. 10.16(c)). Here, the key point is that for the two hidden units ( $U_1$  and  $U_2$ ) at the first hidden layer, we *fix* their model parameters as those were learned in the first iteration (shown as the red (gray in print version), bold lines in Fig. 10.16(c)). We again run the backpropagation

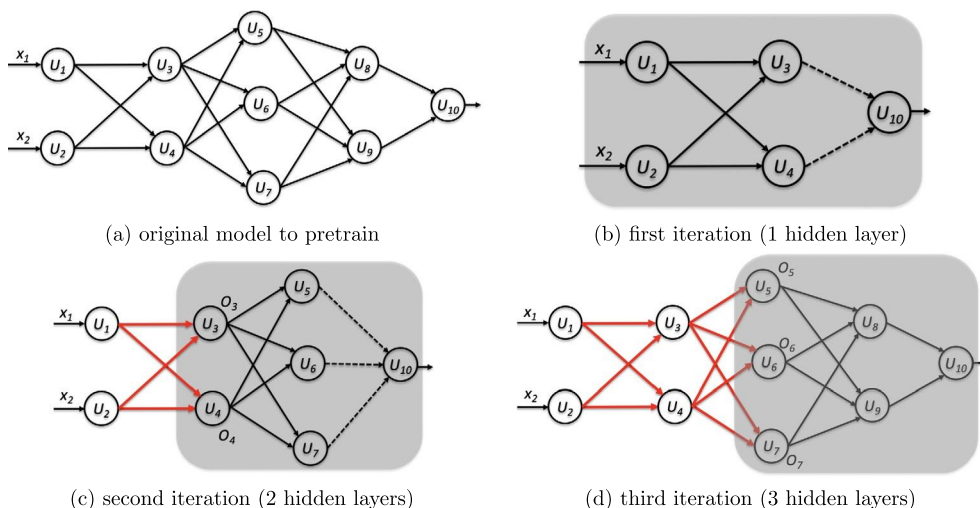


FIGURE 10.16

An illustration of greedy supervised pretraining. The bias values are not shown in the figure. The learned weights to the output unit (except the last iteration) are ignored, shown as dashed lines. The learned weights for the hidden layer from the previous iterations are fixed (shown as red (gray in print version), bold lines) to pretrain the weights and bias values of the newly added hidden layer at each iteration. At each iteration, we simply run backpropagation on a two-layer feed-forward network (shown as a shadow box in (b)–(d)) to pretrain the parameters of the newly added hidden layer, with the output of the last hidden layer from the previous iteration as the input units.

algorithm to train this model, to find the model parameters for the three hidden units ( $U_5$ ,  $U_6$ , and  $U_7$ ) in the second hidden layer and those for the output unit  $U_{10}$ . Like before, we ignore the weights and bias values for the output unit  $U_{10}$  (shown as the dashed line in Fig. 10.16(c)). Then, in the third (final) iteration, we add another hidden layer, including units  $U_8$  and  $U_9$  from the third hidden layer of the original model. Again, we fix the model parameters for the hidden units we have pretrained so far ( $U_3$ ,  $U_4$ ,  $U_5$ ,  $U_6$ , and  $U_7$ ) from the previous iterations (shown as red (gray in print version) bold lines in Fig. 10.16(d)), and we perform backpropagation algorithm to learn the weights for the two hidden units of the newly added layer ( $U_8$  and  $U_9$ ) and the output unit  $U_{10}$ .  $\square$

To summarize, greedy supervised training starts with a simple model with only one hidden layer. At each iteration, it incrementally adds one additional hidden layer, while keeping the model parameters of the hidden units learned from the previous iterations unchanged. It runs the backpropagation algorithm to find the model parameters of the newly added hidden units. It keeps doing this until all the hidden layers from the original model have been added. The model parameters from the last iteration are used as the initial model parameters of the original deep learning model. We call it a pretrained model. The method is *greedy*, since at each iteration, we always fix the model parameters of hidden units from the previous iterations unchanged. Since we always fix the model parameters of the hidden units from the previous iteration(s) unchanged, pretraining at each iteration is equivalent to training a two-layer neural

network with the outputs of the last hidden layer from the previous iteration as the input units. See the shaded boxes of Fig. 10.16(b)–(d) for an illustration.

The pretraining is often followed by a *fine-tuning* process. That is, using the model parameters found by pretraining as the initial solution, we perform the backpropagation algorithm to further find better model parameters. In other words, we fine-tune the initial parameters found by pretraining.

The pretraining method described above is *supervised*, since at each iteration, the backpropagation algorithm needs the actual target values of the training tuples to learn (or preset) the model parameters of the newly added hidden layer. We can also use pretraining in the unsupervised learning setting. For example, we can use a special type of neural network, called *autoencoder* (which will be introduced in Section 10.2.6), to pretrain a deep neural network. We can also use a hybrid strategy that combines both supervised and unsupervised pretraining.

Another scenario for using pretraining is *transfer learning*. As introduced in Chapter 7, in a typical transfer learning setting, there are a *source mining task* (e.g., classification of the tweets sentiment for movies) with a large amount of labeled training tuples; and a *target mining task* (e.g., classification of the tweets sentiment for electronics) with a very limited number of labeled training tuples. We first train a deep neural network on the source task (referred to as the source network). Then, we create another deep neural network for the target task (referred to as the target network). This target network is almost the same as the source network. That is, they share the same number of hidden layers and the same number of hidden units at each layer. Most importantly, the model parameters for the hidden units of the target network are the same as those for the source network. In other words, we use the source network to pretrain the hidden units of the target network. Finally, we run the backpropagation algorithm, with the limited amount of the labeled training tuples to train (i.e., fine-tune) the parameters for the output units of the target network. In this way, pretraining helps improve the generalization performance of the target mining task by “borrowing” parameters from the source task.

### 10.2.5 Cross-entropy

For the backpropagation algorithm (Fig. 10.4) introduced in Section 10.1.2, we have used the mean-squared loss to measure the disagreement between the actual ( $T$ ) and predicted ( $O$ ) target values. For the classification task, it is more common to use **cross-entropy** as the loss function. Let us illustrate and compare cross-entropy and mean-squared loss for the binary classification task.

For a binary classification task, each training tuple  $X$  has an actual target value, that is,  $T = 1$  if  $X$  belongs to the positive class and  $T = 0$  if  $X$  belongs to the negative class. In this case, we only need one output unit in the neural network. Using the sigmoid activation function, the output unit outputs a real value between 0 and 1, indicating the posterior probability that the given training tuple belongs to the positive class (i.e.,  $O = P(T = 1|X)$ ). Cross-entropy measures the disagreement between the actual and predicted target values as follows:

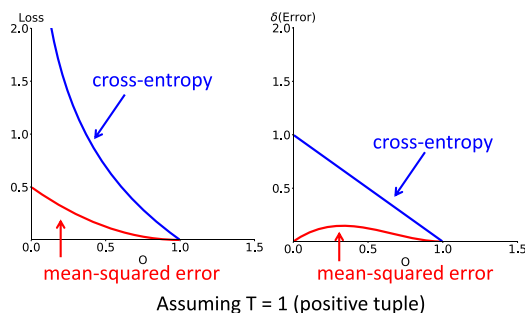
$$\text{Loss}(T, O) = -T \log O - (1 - T) \log (1 - O), \quad (10.11)$$

where we set  $0 \log 0 = 0$  per convention. The cross-entropy loss is small when the predicted output  $O$  agrees with the actual class label  $T$ . For example, if  $T = 1$ ,  $\text{Loss}(T, O) = -\log O$  monotonically decreases as the predicted output  $O$  increases. In contrast, if  $T = 0$ ,  $\text{Loss}(T, O) = -\log (1 - O)$  monotonically increases as the predicted output  $O$  increases. Note that the cross-entropy loss is the same as the negative log likelihood loss for logistic regression classifier introduced in Chapter 6.

|                | Mean-squared error     | Cross-entropy                     |
|----------------|------------------------|-----------------------------------|
| Loss           | $\frac{1}{2}(T - O)^2$ | $-T \log O - (1 - T) \log(1 - O)$ |
| Error $\delta$ | $O(1 - O)(O - T)$      | $O - T$                           |

**FIGURE 10.17**

A comparison between cross-entropy and mean-squared error for binary classification. For the binary classification task, we only need one output unit, and  $O$  is the output of the output unit. The last row is the error  $\delta$  of the output unit during the backpropagation process with the sigmoid activation function.

**FIGURE 10.18**

An illustration of cross-entropy and mean-squared error. Assume the training tuple is a positive tuple (i.e.,  $T = 1$ ).

Fig. 10.17 compares mean-squared loss and cross-entropy loss for a neural network with a single output unit with the sigmoid activation function for the binary classification task. As we have analyzed in Section 10.1.2, with the mean-squared error, when the output unit is saturated (e.g.,  $O \approx 0$  or  $O \approx 1$ ), the error of the output unit  $\delta \approx 0$ , even if the predicted output does not match the actual output (i.e., large  $|T - O|$ ). Such a gradient vanishing problem can be naturally avoided by the cross-entropy loss. As we can see Fig. 10.17, the error of the output unit  $\delta = O - T$  with the cross-entropy loss. Therefore as long as the predicted output deviates from the actual output (i.e., large  $|T - O|$ ), the error will not vanish even if the output unit itself is saturated ( $O \approx 0$  or  $O \approx 1$ ). Fig. 10.18 provides a further illustration on how cross-entropy loss avoids the gradient vanishing problem when the output unit is saturated.

For a multiclass classification task, suppose there are  $C$  class labels in total. The actual target value  $T$  is a binary vector of length  $C$ :  $T = (T_1, T_2, \dots, T_C)$ , where  $T_j = 1$  ( $j = 1, \dots, C$ ) if the training tuple belongs to the  $j$ th class and  $T_j = 0$  otherwise. There are  $C$  output units, and therefore the predicted target value  $O$  is also a vector of length  $C$ :  $O = (O_1, O_2, \dots, O_C)$ , where  $O_j = P(T = j | \mathbf{X})$  ( $j = 1, \dots, C$ ). The cross-entropy in this case is defined as follows.

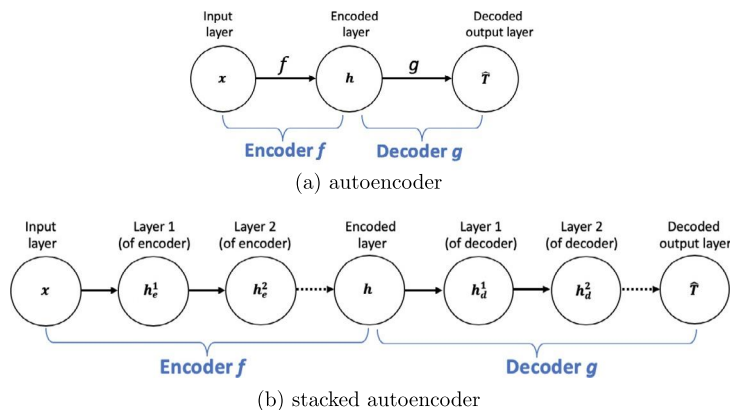
$$\text{Loss}(T, O) = - \sum_{j=1}^C T_j \log O_j \quad (10.12)$$

### 10.2.6 Autoencoder: unsupervised deep learning

**Introducing autoencoder.** For the backpropagation algorithm in Fig. 10.4, it requires the actual target value  $T$ , which could be the real-valued output for regression problem or the discrete-valued output for classification problem. In other words, the training process is *supervised*. But, what if there is no such supervision? Can we still use the backpropagation algorithm to train a feed-forward neural network? The answer is yes. Simply put, we set the output layer to share the same number of the input units, and just use the input data  $x$  as the target output  $T = x$ . Then, we use the backpropagation algorithm to minimize the loss between the predicted output  $\hat{T}$  and the target output  $T = x$ . In other words, we aim to use the neural network to *reconstruct* the input data  $x$ . This type of neural network is called **autoencoder**.

The simplest autoencoder has one hidden layer (see Fig. 10.19(a)), which consists of two parts, including the encoder  $f$  and the decoder  $g$ . The encoder  $f$  corresponds to the hidden layer that maps (i.e., encodes) the input  $x$  to its latent representation (or the code of the input)  $h$ . The decoder  $g$  corresponds to the output layer that maps (i.e., decodes) the latent presentation  $h$  to the predicted output  $\hat{T}$ . The output layer has the same number of units as the input layer. The goal is to use the predicted output  $\hat{T}$  to reconstruct the input  $x$ . This can be done by running the backpropagation algorithm to minimize the disagreement between the predicted output  $\hat{T}$  and the original input  $x$ :  $\text{Loss}(x, \hat{T})$ , where the loss function could be a mean-squared error or a cross-entropy loss. For both the encoder  $f$  and the decoder  $g$ , we could have multiple layers (Fig. 10.19(b)), and this is called *stacked autoencoder*.

A major difficulty with autoencoder is that it might simply *copy* the input to the output. This could happen, for example, if we use the same number of hidden units as the input layer in Fig. 10.19(a) and linear activation function for the hidden layer, which would make the autoencoder useless. To address this issue, we could either constrain the architecture of the autoencoder to require that the hidden layer has less units than the input layer or impose regularization terms on the model parameters (e.g., to



**FIGURE 10.19**

Autoencoder (a) and stacked autoencoder (b). In both cases,  $x$ ,  $h$ , and  $\hat{T}$  are vectors, and the output layer has the same number of units as the input layer. By minimizing  $\text{Loss}(x, \hat{T})$ , we aim to reconstruct the input  $x$  by the output  $\hat{T}$ , where  $\text{Loss}()$  is the loss function (e.g., mean-squared error, cross-entropy, etc.)

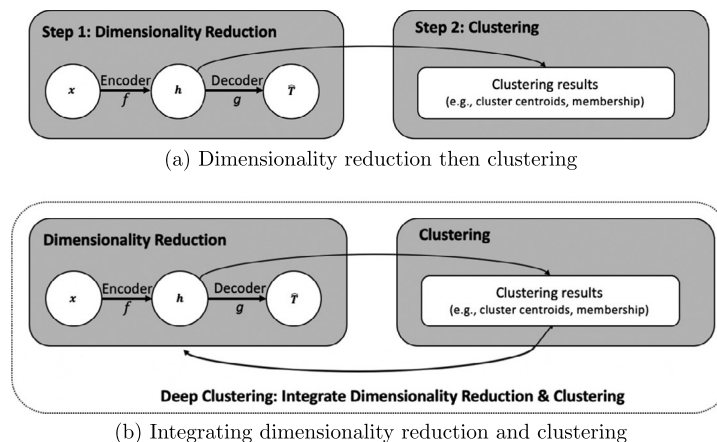


FIGURE 10.20

An illustration of two strategies of applying autoencoder as a dimensionality reduction technique for clustering high dimensional data. For clarity, we assume there is only one encoder layer and one decoder layer.

require the latent representation  $\mathbf{h}$  to be sparse), or adopt *denoising training*, where we aim to minimize the loss between the output of the autoencoder and a *perturbed version* of the input.

Autoencoder can be naturally used for unsupervised learning. Next, we give three examples of applying autoencoder for unsupervised learning, including dimensionality reduction, deep clustering and unsupervised pretraining.

**Autoencoder for dimensionality reduction.** The vector  $\mathbf{h}$  at the encoder layer can be naturally viewed as a latent representation of the input data  $\mathbf{x}$ . If the number of hidden units of the encoder layer is less than the input units, the autoencoder (Fig. 10.19(a)) or stacked autoencoder (Fig. 10.19(b)) effectively performs *dimensionality reduction*. This is conceptually similar to principle component analysis (PCA, introduced in Chapter 2). Indeed, if the decoder uses a linear activation function and mean-squared error as the loss function, the autoencoder in (Fig. 10.19(a)) is equivalent to PCA. By using a nonlinear activation function in the decoder or stacking multiple autoencoders together (Fig. 10.19(b)), (stacked) autoencoder is capable of learning a more powerful low-dimensional representation of the input data than PCA.

**Autoencoder for deep clustering.** In order to cluster high-dimensional data, a natural solution is to find clusters in a low-dimensional space where the clustering structure is more evident than the original space. Generally speaking, there are two strategies, to both of which autoencoder can be applied. See Fig. 10.20 for an illustration.

In the first strategy, we conduct dimensionality reduction, and *then* perform clustering in the low dimensional space. For example, we can first apply autoencoder to the input data (e.g.,  $\mathbf{x}$  in Fig. 10.19(a)) and the output of its encoder (e.g.,  $\mathbf{h}$  in Fig. 10.19(a)) provides the representation of the input data in a low-dimensional space. Then, we perform clustering (e.g., by K-means) on the low-dimensional space  $\mathbf{h}$ . The advantage of this strategy lies in that the dimensionality reduction is performed as a preprocessing step of the clustering task. Once the low-dimensional space  $\mathbf{h}$  is found, we can readily plug in any

off-the-shelf clustering algorithm on it. However, since the dimensionality reduction step is conducted *before* the clustering step, the clustering structure of the input data set might become obscured in the low-dimensional space. For example, the two well-separated clusters in the original high-dimensional space might be overlapped with each other in the low-dimensional space, which would in turn impair the quality of the clustering results.

The second strategy aims to integrate the dimensionality reduction and the clustering steps together. In other words, we want to *embed* the dimensionality reduction step into the clustering step, so that the clustering structure of the input data is well preserved in the low-dimensional space. A variety of different dimensionality reduction approaches (e.g., PCA, autoencoder, matrix factorization) and various clustering methods (e.g., K-means, NMF, probabilistic clustering methods) can be adopted in this strategy. When we use a deep learning model (e.g., autoencoder) for the dimensionality reduction step, this strategy is referred to as *deep learning*. Compared with the first strategy, deep clustering often leads to a better clustering quality, at the cost of increased computational cost, owing to the coupling of the dimensionality reduction and clustering steps.

*So, how can we integrate dimensionality reduction and clustering together?* Let us take autoencoder (for the dimensionality reduction step) and K-means (for the clustering step) as an example to explain its key ideas from the optimization perspective.

In autoencoder, we aim to find the best encoder  $f$  and decoder  $g$  to re-construct the input data. In other words, we wish to minimize the following reconstruction loss

$$\min \sum_{i=1}^N \|\hat{T}_i - \mathbf{x}_i\|_2^2 \quad \text{where } \hat{T}_i = g(f(\mathbf{x}_i)). \quad (10.13)$$

Recall that in k-means (introduced in Chapter 8), we aim to find the best cluster centers  $c_j$  ( $j = 1, \dots, C$ ) and cluster membership for each input data tuple  $\mathbf{x}_i$  ( $i = 1, \dots, N$ ) by minimizing another type of loss, namely the sum of squared errors (SSE):

$$\min \sum_{i=1}^N \sum_{\mathbf{x}_i \in c_j} \|\mathbf{x}_i - c_j\|_2^2. \quad (10.14)$$

Now, in order to integrate the dimensionality reduction (Eq. (10.14)) and clustering (Eq. (10.13)) together, we instead aim to simultaneously find the best the encoder  $f$ , the decoder  $g$ , the cluster centers  $c_j$  ( $j = 1, \dots, C$ ), and the cluster membership by minimizing a linear weighted sum of the above two types of loss (i.e., the reconstruction loss in Eq. (10.13) and the SSE in Eq. (10.14)):

$$\min \underbrace{\sum_{i=1}^N \|\hat{T}_i - \mathbf{x}_i\|_2^2}_{\text{reconstruction loss}} + \alpha \underbrace{\sum_{i=1}^N \sum_{f(\mathbf{x}_i) \in c_j} \|f(\mathbf{x}_i) - c_j\|_2^2}_{\text{SSE}}, \quad (10.15)$$

where  $\hat{T}_i = g(f(\mathbf{x}_i))$ , and  $\alpha > 0$  is a regularization parameter to balance the relative weight of the two types of loss. Notice that the SSE in Eq. (10.15), we use  $f(\mathbf{x}_i)$ , that is, the output of the encoder  $f$ , instead of the original input data  $\mathbf{x}_i$ . This is the key that differentiates itself from the first strategy



(i.e., dimensionality reduction then clustering). In this way, we integrate the two steps (dimensionality reduction and clustering) together.

In order to solve the optimization problem in Eq. (10.15), we can use an iterative algorithm. In each iteration, we do the following two steps in an alternating way. First, we fix the encoder  $f$  and the decoder  $g$ , and run k-means algorithms on  $f(\mathbf{x}_i)$  (i.e., the output of the encoder  $f$ ) to find the current cluster centers  $c_j$  ( $j = 1, \dots, C$ ) and the cluster membership for each input data tuple. Then, we fix the cluster centers  $c_j$  ( $j = 1, \dots, C$ ) and the cluster membership to train an autoencoder to update the encoder  $f$  and the decoder  $g$ . This step can be viewed as training a *regularized autoencoder*, where the cluster centers and membership jointly provide regularization for both the encoder and the decoder. By iteratively updating the autoencoder that respects the current cluster centers and membership, the produced low-dimensional space (i.e.,  $f(\mathbf{x}_i)$ ) is tailored for the clustering task where the clustering structure is likely to be well preserved.

Deep clustering provides a powerful and flexible framework to integrate the dimensionality reduction and clustering together. For example, in addition to autoencoder, we can use alternative deep learning architectures in Eq. (10.15) depending on the specific input data type (e.g. CNNs for images, RNNs for sequence, GNNs for graph data). In addition to the SSE loss in Eq. (10.15), we can also use, for instance, nonnegative matrix factorization (NMF) or Kullback–Leibler (KL)-divergence as the loss for the clustering step. NMF was introduced in Chapter 9. KL-divergence loss is another popular choice for clustering. In a nutshell, it minimizes the KL divergence between a cluster centroid-based distribution and an auxiliary target distribution, which was found to be quite effective in deep clustering.

**Autoencoder for unsupervised pretraining.** We can also use autoencoder to pretrain a feed-forward neural network, which is referred to as *unsupervised pretraining*. Similar to greedy supervised pretraining introduced in Section 10.2.4, we can use autoencoder to preset the model parameters layer-by-layer in a greedy layer way. In order to pretrain the model parameters (the weight matrix and bias vector) of a given layer  $k$ , we keep the input layer and all the hidden layers up to and including layer  $k$ , and we treat them as the encoder  $f$ . In the meaning, we remove all the remaining hidden layers after layer  $k$  (layer  $(k + 1)$ , ..., layer  $L$ ) and the output layer, and we replace them by a new output layer with the same number of units as the input layer, which is treated as the decoder  $g$ . By running the backpropagation algorithm on this autoencoder, we obtain the preset (or pretrained) model parameters for layer  $k$ .

### 10.2.7 Other techniques

Responsive activation functions (e.g., ReLU, Section 10.2.1), adaptive learning rate (Section 10.2.2), dropout (Section 10.2.3), pretraining (Section 10.2.4), cross-entropy loss function (Section 10.2.5), and autoencoder (Section 10.2.6) have significantly improved the training of deep learning models. In this section, we briefly introduce some additional techniques to further improve the training of deep learning models.

#### Gradient exploding problem

As we can see from Fig. 10.9, when we are close to the “gradient cliff” where the magnitude of the gradient  $\mathbf{g}_t$  is high, the updated model parameters by stochastic gradient descent might jump over the desirable region. One common cause of a gradient cliff is due to the backpropagation algorithm. To see this, suppose we use the ReLU activation function for the hidden units, as introduced in Section 10.2.1, to avoid gradient vanishing problem. Then, during the backpropagation process, the error  $\delta_j$  for a given

hidden unit  $j$  is the aggregated errors of hidden units from the next higher layer (i.e.,  $\delta_j = \sum_l \delta_l w_{jl}$ ), if this unit  $j$  is active (i.e.,  $I_j > 0$ ). Therefore if the current weights  $w_{jl}$  are large, the backpropagation could amplify the error. That is, the magnitude of error  $\delta_j$  could be significantly larger than those from the next higher layer. When we propagate the errors through the deep neural networks, the errors of the hidden units at the lower layers could be even higher, leading to a gradient cliff. This phenomenon is known as *gradient exploding*.

The adaptive learning rate introduced in Section 10.2.2, such as AdaGrad or RMSProp, can help alleviate the gradient exploding problem. This is because, when the magnitude of the gradient at the current iteration is high, AdaGrad or RMSProp will automatically shrink the adaptive learning rate more, and thus the model parameters will be updated less (i.e.,  $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_t$ ). Another simple yet effective way to handle gradient exploding is through *gradient clipping*. That is, we set a threshold  $c$ , as the ceiling for the maximum magnitude of the gradient. At a given epoch, if the magnitude of the gradient  $\mathbf{g}_t$  exceeds this threshold, we normalize the gradient to reduce its magnitude. To be specific, if the Euclidean norm  $\|\mathbf{g}_t\| > c$ , we set  $\tilde{\mathbf{g}}_t = \mathbf{g}_t \cdot \frac{c}{\|\mathbf{g}_t\|}$ . Then, we use the normalized gradient  $\tilde{\mathbf{g}}_t$  to update the model parameters:  $\theta_{t+1} = \theta_t - \eta_t \tilde{\mathbf{g}}_t$ .

### Early stopping

As mentioned in Section 10.1.3, backpropagation algorithm aims to minimize the (approximated) training error in Eq. (10.9). Therefore it is possible that after a certain number of epochs, the algorithm continues to decrease the objective function value  $E(\theta)$ , but the trained model might perform poorly on the future test tuples. An effective way to avoid such overfitting is *early stopping*. To be specific, we maintain a separate *validation set*. At each epoch of the backpropagation algorithm, we measure the validation error on this validation set using the learned model parameters at the current epoch. If the validation error does not decrease in a certain number of consecutive epochs (e.g., 10 or 100), we terminate the algorithm, even though the training error might continue to decrease.

### Batch normalization

From the optimization perspective, the gradient  $\mathbf{g}_t$  from Eq. (10.10) provides a guidance in terms of how one should update the model parameters in order to reduce the training error in Eq. (10.9). There is an implicit assumption for (stochastic) gradient descent in Eq. (10.10). That is, the model parameters from the other units are *fixed*. However, backpropagation algorithm (Fig. 10.4) updates *all* model parameters in one epoch.

A simple yet effective strategy to address this discrepancy is **batch normalization**. It works as follows by making one simple change to the standard backpropagation algorithm, that is, *z-score normalization* of the outputs of input or hidden units. To be specific, at a given epoch of backpropagation, we draw a mini-batch of  $m$  tuples. For each input or hidden unit  $j$ , we use the current model parameters to calculate the output of each tuple from the given unit  $j$ :  $O_j^1, \dots, O_j^m$ , where the superscript is used to index different training tuples. We calculate the sample mean  $\mu_j$  and sample standard deviation  $\sigma_j$  for  $O_j^1, \dots, O_j^m$ . Then, we perform z-score normalization on the output of unit  $j$  as  $\tilde{O}_j^l = \frac{O_j^l - \mu_j}{\sigma_j}$  ( $l = 1, \dots, m$ ). We will use the normalized output  $\tilde{O}_j^l$  ( $l = 1, \dots, m$ ) in the remaining computation of the backpropagation algorithm. Z-score normalization was introduced in Chapter 2.

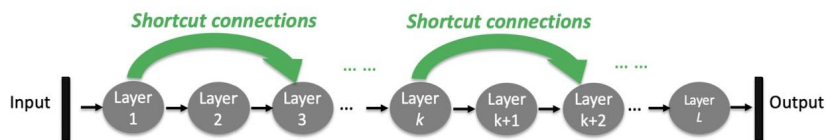


FIGURE 10.21

An illustration of shortcut connections. The black is the standard feed-forward neural network with  $L$  layers. The bold, green (dark gray in print version) arrows are the shortcut connections, which link the units in a lower layer to the units in a nonadjacent upper layer, skipping one or more layers in between. In this example, the shortcut connections add links from layer  $k$  to layer  $k + 2$ , skipping one layer in between (layer  $k + 1$ ). The added connections provide extra “shortcuts” to forward propagate the input to the output, as well as backward propagate the errors and gradients.

### Moment-based approaches

In Eq. (10.10), we have used the *current* gradient  $\mathbf{g}_t$  to update the model parameters. That is, at each epoch, we move or update model parameters in the negative direction of the current gradient:  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}$ , where  $\Delta\boldsymbol{\theta} = -\eta\mathbf{g}_t$  and  $\eta$  is the learning rate. Alternatively, we can use **moment**  $\mathbf{r}_t$  to update the model parameters:  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{r}_t$ . Unlike the gradient descent method that only uses the current gradient  $\mathbf{g}_t$  to update the model parameters, the moment  $\mathbf{r}_t$  considers *all* historical gradients.

Formally, the moment is calculated as follows. At the beginning of the algorithm, we initialize the moment as a zero vector (i.e.,  $\mathbf{r}_0 = \mathbf{0}$ ). At each epoch, we update the moment  $\mathbf{r}_t$  as  $\mathbf{r}_t = \alpha\mathbf{r}_{t-1} - \eta\mathbf{g}_t$ , where  $0 < \alpha < 1$  is the decaying factor,  $\eta$  is the learning rate and  $\mathbf{g}_t$  is the current gradient. We can see that the moment is the *exponentially decaying average* of all historical gradients, with higher weights for more recent gradients. Compared with the standard stochastic gradient descent, moment-based approaches tend to reduce the variance of the gradient estimated from the mini-batch. We can develop similar strategies as AdaGrad or RMSProp to adaptively choose the learning rate for moment-based approaches. A commonly used moment-based approach with adaptive learning rate in deep learning is called **Adam** (which stands for Adaptive Moment Estimation).

### Shortcut connections

In a standard feed-forward neural network, we forward propagate the input to the output and then backward propagate the error and gradients. When the network becomes deeper, with an increased number of hidden layers, it becomes more difficult to forward and backward propagate such information, which in turn leads to the gradient vanishing or exploding problems. In addition to the techniques we have already introduced (e.g., responsive activation function, pretraining, cross-entropy loss function, etc.), another simple yet effective method is to use *shortcut connections* (also referred to as skip connections). Fig. 10.21 provides an illustration of shortcut connection, where we have added extra connections from lower layers to nonadjacent higher layers, skipping one or more layers in between. If a lower layer (layer  $k$  in this example) has the same number of units as the higher layer (layer  $k + 2$  in this example) that shortcut connections link to, we can simply copy the outputs of the lower layer as the additional net input of the higher layer. The corresponding network is referred to as *resNets*, which stands for residual networks. If layer  $k$  and layer  $k + 2$  have different units, we can introduce additional parameters as the weights that connecting different units in layer  $k$  to different units in layer  $k + 2$ . The correspond-

ing network is referred to as *HighwayNets*, in that the added shortcut connections can be viewed as the highway (relative to the normal connection, i.e., the black arrows in the figure) to fast forward and backward propagate the information (e.g., input, output, errors, gradients). The shortcut connections have been found to be able to greatly mitigate the gradient vanishing problem.

## 10.3 Convolutional neural networks

One of the most successful deep learning models is **convolutional neural networks** (CNNs), which have achieved tremendous success in numerous applications. To name a few, CNNs often play a central role in the state-of-the-art methods for various image object recognition tasks; CNNs have been successfully applied for face recognition in the challenging scenario where the faces could be partially occluded with various rotations (e.g., upside down). In healthcare, CNNs have been applied to detect biomarkers, assess patient's health risks, and re-purpose drugs (i.e., drug discovery). CNNs have been applied to time series data to forecast the future measurements and detect anomalies. CNNs were used in AlphaGo, a computer program developed by DeepMind Technologies, which defeated the world champion in a highly complex game called Go in 2016.

CNNs are designed for grid-like data, such as images. Mathematically, the input grid data for CNNs are represented as a *multidimensional array* (i.e., a *tensor*). For example, the time-series data (1-D grid) can be represented as a 1-D tensor (i.e., a single-dimensional array) whose elements provide the measures at the corresponding time stamps; the gray image can be represented as a 2-D tensor (i.e., a matrix) whose elements measure the gray scales of the corresponding pixels; and the color-image can be represented as a 3-D tensor with the third mode representing different color channels (e.g., red, green, blue).

We start with 1-D convolution operation as a feature extraction process (Section 10.3.1), and its generalization to multidimensional convolution (Section 10.3.2). Based on that, we introduce the convolution layer, which will be in turn used to construct a convolutional neural network (Section 10.3.3).

### 10.3.1 Introducing convolution operation

Suppose we are given a 1-D time-series data (e.g., room temperature over time). In one scenario, we might be interested in the overall trend of temperature without considering the noise or small fluctuation of temperature. To this end, we could *smooth* the input time series by replacing the original temperature by the average temperature in the local vicinity of the corresponding time stamp. In another scenario, we might want to find out whether there is a sudden temperature change at certain time stamp(s). To this end, we could compute the difference of the temperature before and after the corresponding time stamps. *Is there an automatic way to extract such signals or features (e.g., average, difference) from the input data?* A powerful technique that can be used to answer this question is called **convolution**, which is the key operation in CNNs.

Given an input  $I$  represented as a tensor, and a **kernel**  $K$  (which is also referred to as a *filter*) represented as another, but often smaller, tensor; the convolution operation generates a **feature map**  $S$  that is also a tensor. We use  $*$  to represent the convolution operation:  $S = I * K$ . Roughly speaking, each element in the feature map  $S$ , as its name suggests, is the feature of the corresponding element of

the input. It is generated by a linear sum of the elements in the vicinity of the corresponding element of the input  $\mathbf{I}$ , where the linear weights are provided by the kernel  $\mathbf{K}$ .

If the input  $\mathbf{I}$  is 1-D, we represent it as a single dimensional array of length  $N$ . Let the index of the array  $\mathbf{I}$  start with 1. That is, the first and the last elements of  $\mathbf{I}$  are  $\mathbf{I}[1]$  and  $\mathbf{I}[N]$ , respectively. The kernel  $\mathbf{K}$  is also a single-dimensional array, whose length  $P$  is often much smaller than the input data  $\mathbf{I}$  (i.e.,  $P \ll N$ ). We often choose  $P$  to be an odd number and let the index of the middle element of the kernel to be 0. That is, the first and the last elements of the kernel  $\mathbf{K}$  are  $\mathbf{K}[-\frac{P-1}{2}]$  and  $\mathbf{K}[\frac{P-1}{2}]$ , respectively. With these notations, 1-D convolution operation is formally defined as follows<sup>7</sup>:

$$S[i] = \sum_{p=-\frac{P-1}{2}}^{\frac{P-1}{2}} \mathbf{I}[i+p]\mathbf{K}[p], \quad (i = 1, \dots, N), \quad (10.16)$$

where the index of the feature map  $\mathbf{S}$  starts with 1. In order to compute an element of the feature map  $\mathbf{S}[i]$ , we first find the corresponding element of the input  $\mathbf{I}[i]$ . Then, we draw a vicinity (or window or neighborhood) of size  $P$  centered around element  $\mathbf{I}[i]$ , with  $\frac{P-1}{2}$  elements before and after  $\mathbf{I}[i]$  respectively. Finally, we weight each element in such a vicinity by the corresponding element in the kernel  $\mathbf{K}$  and take the weight sum as the feature map element  $\mathbf{S}[i]$ .

Put another way, we can think of convolution operation as the following process. The kernel  $\mathbf{K}$  traverses the input  $\mathbf{I}$ , and the feature map  $\mathbf{S}$  is the *response* of the input  $\mathbf{I}$  with respect to the kernel. That is, each time the kernel  $\mathbf{K}$  visits an element of the input  $\mathbf{I}[i]$ , it identifies a neighborhood:  $\{\mathbf{I}[i - \frac{P-1}{2}], \dots, \mathbf{I}[i + \frac{P-1}{2}]\}$ . The identified neighborhood has the same size as the kernel, and it is referred to as the *receptive field* of  $\mathbf{I}[i]$  in computer vision. The weighted sum of all the elements in the receptive field, weighted by the elements in the kernel, becomes the corresponding element in the feature map, which can be viewed as the *response* of the input element  $\mathbf{I}[i]$  with respect to the kernel. This process is equivalent to the dot product between the receptive field and kernel.

Note that Eq. (10.16) has a boundary issue. That is, when we compute the first few or the last few elements of the feature map, there are not enough elements in the input  $\mathbf{I}$  according to Eq. (10.16). For example, in order to compute  $\mathbf{S}[1]$ , Eq. (10.16) requires  $\mathbf{I}[-\frac{P-1}{2} + 1], \dots, \mathbf{I}[0]$ , none of which exists in the input  $\mathbf{I}$ . In order to address this boundary issue, we can resort to **zero padding**. For example, we can append  $\frac{P-1}{2}$  zeros before  $\mathbf{I}[1]$  and  $\frac{P-1}{2}$  zeros after  $\mathbf{I}[N]$ , respectively. Without zero padding, the generated feature map  $\mathbf{S}$  will have a smaller size than the input  $\mathbf{I}$ , without the first and the last few elements. Another parameter that affects the size of the feature map is *stride*, which controls how the kernel center visits the elements of the input to perform convolution. If the stride is 1, the kernel center visits each element of the input; if the stride is 2, it will visit every other element of the input (and thus the feature map is roughly half the size of the input), and so on. For the discussion in the rest of this section, we assume that the stride is always 1 and the input is appropriately zero-padded so that the feature map shares the same size as the input.

**Example 10.5.** Let us look at an example in Fig. 10.22. In this example, the input  $\mathbf{I}$  has eight elements:  $\mathbf{I}[1] = 10$ ,  $\mathbf{I}[2] = 50$ ,  $\mathbf{I}[3] = 30$ ,  $\mathbf{I}[4] = 40$ ,  $\mathbf{I}[5] = 50$ ,  $\mathbf{I}[6] = 80$ ,  $\mathbf{I}[7] = 90$ ,  $\mathbf{I}[8] = 85$ , and  $N = 8$ .

<sup>7</sup> Several alternative convolution definitions exist with different indexing mechanisms for the input, the kernel or feature map. For example, the 1-D convolution can also be defined as  $S[i] = \sum_p \mathbf{I}[i-p]\mathbf{K}[p]$ , which is equivalent to Eq. (10.16) by flipping the kernel at the center  $\mathbf{K}[0]$ . In literature, 1-D in Eq. (10.16) is also referred to as *cross-correlation*.

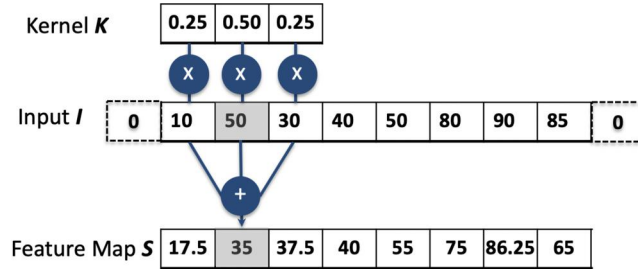


FIGURE 10.22

An example of 1-D convolution. The input  $I$  has eight elements. The kernel  $K$  has three elements. In order to make sure that the feature map  $S$  has the same length as the input  $I$ , we pad one zero before  $I[1]$  and one zero after  $I[8]$  (two dashed boxes). The blue (mid gray in print version) lines and circles demonstrate how to compute one element ( $S[2]$ , the shaded cell in  $S$ ) from the vicinity of the corresponding input element ( $I[2]$ , the shaded cell in  $I$ ), with the weights provided by the kernel  $K$ . The three elements, including  $I[1]$ ,  $I[2]$  and  $I[3]$ , form the receptive field of  $I[2]$ .

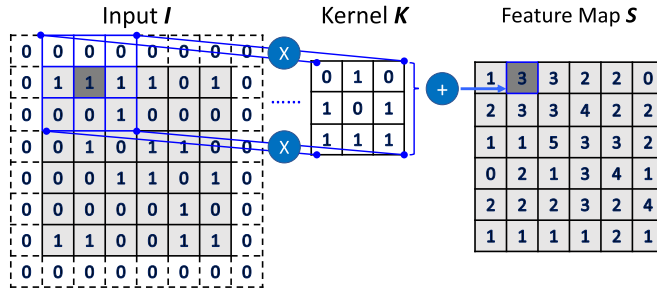
The kernel  $K$  has three elements (i.e.,  $P = 3$ ). Note that the middle element of the kernel is indexed as 0 (i.e.,  $K[-1] = 0.25$ ,  $K[0] = 0.50$ , and  $K[1] = 0.25$ ). In order to compute an element in the feature map  $S[2]$ , we first find the corresponding element in the input  $I[2]$ . Then, we draw a window of size 3 (i.e., the same size as the kernel), centered around  $I[2]$ . The three elements in this window are  $I[1] = 10$ ,  $I[2] = 50$ , and  $I[3] = 30$ . We weight each of these three elements by the corresponding elements in the kernel and take the resulting sum as  $S[2]$ . That is,  $S[2] = K[-1] \times I[1] + K[0] \times I[2] + K[1] \times I[3] = 0.25 \times 10 + 0.50 \times 50 + 0.25 \times 30 = 35$ . In order to compute  $S[1]$  and  $S[8]$ , we pad a zero before  $I[1]$  and after  $I[8]$ , respectively (i.e.,  $I[0] = I[9] = 0$ ). With the padded zeros, we can compute  $S[1]$  by Eq. (10.16):  $S[1] = 0.25 \times 0 + 0.50 \times 10 + 0.25 \times 50 = 17.5$ . Similarly, we have that  $S[8] = 0.25 \times 90 + 0.50 \times 85 + 0.25 \times 0 = 65$ .  $\square$

We can view the convolution operation as a *feature extraction* process. To see this, suppose the kernel  $K = (1/3, 1/3, 1/3)$ . Then, in the above example, the convolution operation acts as a *smoothing* operator. That is, the elements of the feature map tell us the *average* input measures in the local vicinity of the corresponding elements of the input. With the same input  $I$ , if we choose a different kernel  $K = (-1, 0, 1)$ , the convolution operation acts as a *change* (or difference) detector. That is, the corresponding feature map tells whether there is a sudden increase or decrease in the local vicinity of the input  $I$ .

### 10.3.2 Multidimensional convolution

If the input  $I$  is a 2-D matrix of size  $N \times M$ , the kernel (or filter)  $K$  is also a 2-D matrix of size  $P \times Q$ , where  $P \ll N$  and  $Q \ll M$ .  $P$  and  $Q$  are often set as odd numbers, and typical choices are  $P = Q = 3$  or  $P = Q = 5$ . The 2-D convolution will generate a 2-D feature map  $S$  as follows:

$$S[i][j] = \sum_{q=-\frac{Q-1}{2}}^{\frac{Q-1}{2}} \sum_{p=-\frac{P-1}{2}}^{\frac{P-1}{2}} I[i+p][j+q]K[p][q], \quad (i = 1, \dots, N; j = 1, \dots, M), \quad (10.17)$$



**FIGURE 10.23**

An example of 2-D convolution. The input  $I$  is a  $6 \times 6$  binary matrix. The kernel  $K$  is a  $3 \times 3$  binary matrix. We pad zeros around the input  $I$ . The  $3 \times 3$  matrix with blue (mid gray in print version) borderline forms the receptive field of  $I[1][2]$ .

where the indices of the input and feature map start with 1. That is, the first elements of the input and the feature map are  $I[1][1]$  and  $S[1][1]$ , respectively, and the last elements of the input and the feature map are  $I[N][M]$  and  $S[N][M]$ , respectively. Like in the 1-D convolution, the center element of the kernel is indexed as  $K[0][0]$ . In other words, some elements in the kernel have negative indices. We can also zero-pad the input  $I$  to ensure the feature map has the size as the input.

**Example 10.6.** Let us look at an example in Fig. 10.23, to see how 2-D convolution works. In order to compute the element in the feature map  $S[1][2]$ , we first find the corresponding element in the input  $I[1][2]$  and draw the receptive field of  $I[1][2]$  (i.e., a  $3 \times 3$  matrix centered around  $I[1][2]$  with blue (mid gray in print version) border line). We index the center element of the kernel as  $K[0][0]$ . Similar to 1-D convolution, we then weight each element in the receptive field of  $I[1][2]$  by the corresponding elements in the kernel  $K$  and take the resulting sum as  $S[1][2] = I[0][1] \times K[-1][-1] + I[0][2] \times K[-1][0] + I[0][3] \times K[-1][1] + I[1][1] \times K[0][-1] + I[1][2] \times K[0][0] + I[1][3] \times K[0][1] + I[2][1] \times K[1][-1] + I[2][2] \times K[1][0] + I[2][3] \times K[1][1] = 0 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 1 + 0 \times 1 + 1 \times 1 = 3$ . Since we pad zeros around the input  $I$ , we can compute other elements in the feature map  $S$  according to Eq. (10.17). The resulting feature map  $S$  is shown on the right side of Fig. 10.23.  $\square$

We can generalize the above definitions for 1-D (Eq. (10.16)) and 2-D convolutions (Eq. (10.17)) when the input is a multidimensional tensor. For example, if the input is a 3-D tensor, mathematically, we can define a 3-D tensor with a much smaller size than the input, and then by generalizing the 2-D convolution in Eq. (10.17), we will output a 3-D feature map. The process is similar to 1-D or 2-D convolutions. That is, we let the kernel traverse or slide over each element of the input. For each element of the input, we take the dot product between the elements of its receptive field and those of the kernel, as the response of this element with respect to the kernel and it becomes the corresponding element of the feature map. In principle, we could let the kernel traverse or slide along *each* of the three dimensions, with the help of zero-padding. Thus, the feature map  $S$  could also be a 3-D tensor.

In many real applications (such as computer vision) with 3-D tensor input, we often use a variant of the 3-D convolution introduced above. For example, given an input image, we can represent it as

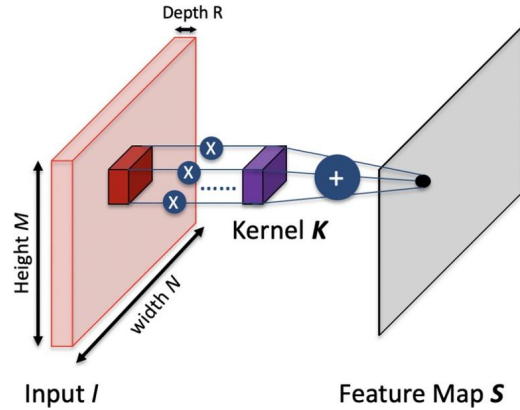


FIGURE 10.24

An example of 3-D convolution. The kernel  $\mathbf{K}$  is local in width and height, but full in the depth (i.e., it has the same depth as the input  $\mathbf{I}$ ). The feature map  $\mathbf{S}$  is a 2-D matrix.

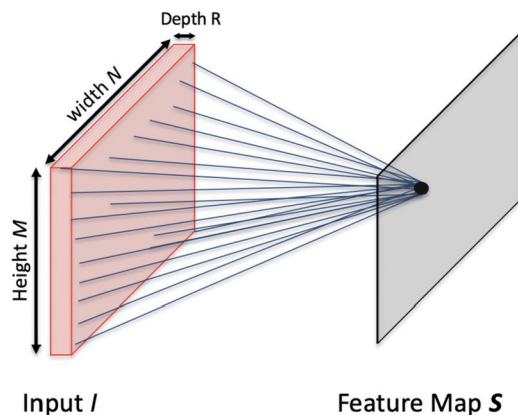
a 3-D tensor  $\mathbf{I}$  of size  $N \times M \times R$ , where  $N$  and  $M$  represent the width and the height of the image, respectively, and  $R$  is the depth (e.g.,  $R = 3$  representing the number of color channels, including red, green, and blue colors of the image). In practice, the kernel  $\mathbf{K}$  is *local* in spatial (width and height) but *full* in depth. That is, the kernel  $\mathbf{K}$  has a size of  $P \times Q \times R$ , where  $P \ll N$  and  $Q \ll M$  but the depth of the kernel is the same as the input  $\mathbf{I}$ . The 3-D convolution operation is defined as follows:

$$S[i][j] = \sum_{q=-\frac{Q-1}{2}}^{\frac{Q-1}{2}} \sum_{p=-\frac{P-1}{2}}^{\frac{P-1}{2}} \sum_{r=1}^R I[i+p][j+q][r] \mathbf{K}[p][q][r], \quad (i = 1, \dots, N; j = 1, \dots, M), \quad (10.18)$$

where the indices of the input, the kernel, and the feature map are defined in the similar way as in 1-D and 2-D convolutions. The only difference is that the index of the depth dimension of the kernel starts with 1, which is shared with that of the input. In other words, when we slide the kernel over the input, we only allow it to slide along the dimensions of the width and the height. As such, the resulting feature map  $\mathbf{S}$  is a 2-D matrix (instead of a 3-D tensor), which shares the same width and height as the input with the help of zero-padding and setting stride as 1. See Fig. 10.24 for an illustration.

*Why do we need 3-D convolution?* For the example of Fig. 10.24, in order to generate an element in the feature map, we only need to take the dot product between the kernel and the corresponding receptive field. If we treat the input as fixed and the kernel as the model parameters (which can be learned from the training tuples), there are  $P \cdot Q \cdot R$  parameters. More importantly, we use the *same* kernel to traverse the input to generate the entire feature map  $\mathbf{S}$ . In other words, if we view the kernel as a neuron (i.e., a unit of the neural network), we only need one single kernel with  $P \cdot Q \cdot R$  parameters to generate the feature map  $\mathbf{S}$ . In contrast, if we resort to the fully connected feed-forward neural network, for each element in the feature map, it needs to be connected to *all* elements of the input  $\mathbf{I}$ , and thus it requires  $N \times M \times R$  parameters. What is more, for different elements of the feature map, they have different sets of connections to the input. Since there is no parameter sharing among different sets of



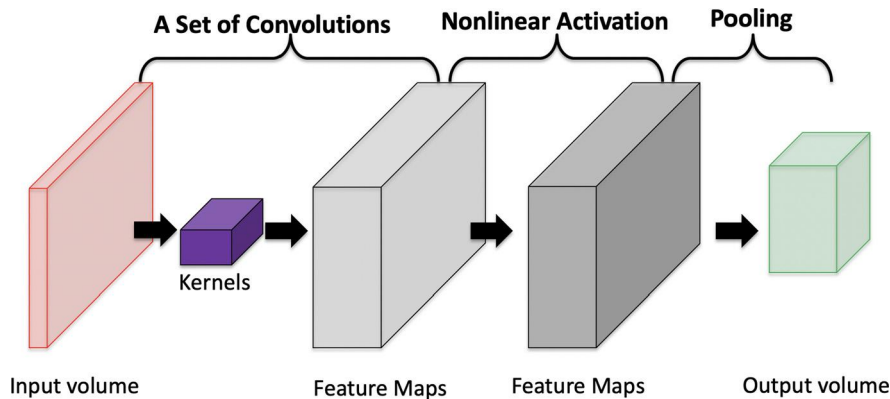
**FIGURE 10.25**

An example of using a fully connected layer to generate the feature map. Each element of the feature map (the black dot on the right) is connected to every element of the input (on the left). There is no parameter sharing among different elements of the feature map. It requires significantly more parameters, compared with 3-D convolution (Fig. 10.24).

connections, each set would require  $N \times M \times R$  parameters. Suppose the feature map has the same size as the input (e.g., by zero-padding and setting stride as 1), a feed-forward neural network layer would need  $N \times M \times R \times N \times M$  parameters, which are significantly more than the 3-D convolution. See Fig. 10.25 for an illustration.

**Example 10.7.** Given image of size  $320 \times 280 \times 3$ , we aim to generate a feature map of size  $320 \times 280$ . If we use a kernel of size  $5 \times 5 \times 3$ , we only need  $5 \times 5 \times 3 = 75$  parameters. In contrast, if we use a fully connected layer, we would need  $320 \times 280 \times 3 \times 320 \times 280 = 24,084,480,000$  (more than 24 billion!) parameters. Note that here, we have ignored the additional parameters for bias values for simplicity.  $\square$

There are two additional advantages of convolution. First, compared with a feed-forward neural network, which is fully connected, the kernel is *locally* or *sparsely* connected. As mentioned before, we only need to consider a small neighborhood around a given element of the input (i.e., its receptive field) to calculate the corresponding element of the feature map. For applications like computer vision, the locally connected property is highly desirable. For example, in order to detect whether certain features (e.g., edge, object parts) exist at a certain location of an image, we only need to look at the vicinity of that location. Second, in 3-D convolution, we apply the same kernel at different locations of an image. This leads to an important property for some applications (such as computer vision), namely *translational equivalence*. To see this, suppose the kernel is designed to detect a certain type of edge in the input image. If we shift the location of every pixel of the input image by the same amount, intuitively, we should be able to use the same kernel to detect the shifted edge, only that the corresponding edge is shifted in the feature map. Mathematically, if function  $f(x)$  represents the convolution and function  $g(x)$  represents the translation (e.g., shifting the pixel location), then  $f(g(x)) = g(f(x))$ . Convolution



**FIGURE 10.26**

A convolution layer. It takes the input volume and generates an output volume, both in the form of a 3-D tensor. A convolutional layer consists of a set of convolutions, followed by a nonlinear activation (such as ReLU) and a pooling operation. (In some literature, the convolution layer is referred to the set of the convolutions only; and the nonlinear activation with ReLU and the pooling operations are referred to as two separate layers, namely ReLU layer and pool layer, respectively.) Each kernel is applied separately to the input volume, generating a feature map. The number of feature maps is the same as the number of kernels. The pooling operation reduces the size of feature maps in both width and height, but keeps the same size of the depth.

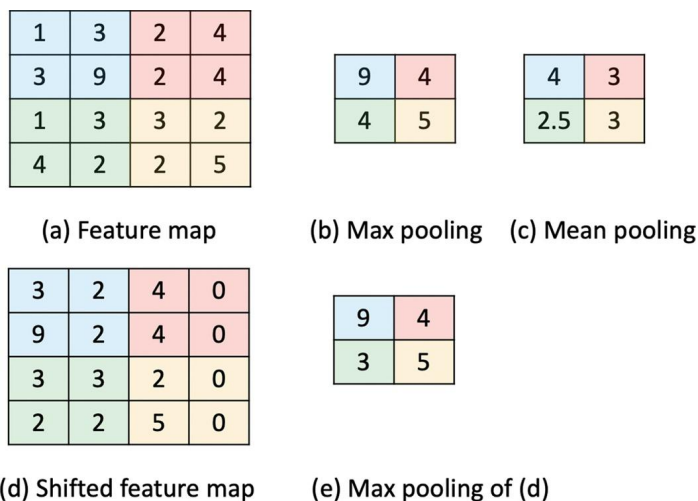
is translational equivalent, which means that if we first translate the image (function  $g(x)$ ) and then apply the convolution (function  $f(x)$ ), the result is equivalent to if we first apply the convolution (function  $f(x)$ ) and then translate it (function  $g(x)$ ).

### 10.3.3 Convolutional layer

In a **convolutional layer**, there are often multiple kernels (i.e., a set of kernels), which are organized into a tensor (see the purple (mid gray in print version) box in the left part of Fig. 10.26). Each kernel is applied separately to the input volume (or input for short) to perform a convolution operation between them. Since each kernel is full in depth, meaning that it shares the same depth as the input (e.g., the number of color channels), the convolution operation between the input and a given kernel will produce a 2-D feature map. By applying all the kernels to the input volume, we will produce multiple feature maps. The number of generated feature maps is the same as the number of kernels. We stack (organize) all the feature maps into a 3-D tensor. The depth of the feature map tensor represents the number of generated feature maps, and therefore it is the same as the number of kernels.

For the feature map tensor generated by the convolution operations, it is further fed into two additional operations, including a **nonlinear activation** and a **pooling** operation. The typical choice for nonlinear activation is ReLU, which basically sets all the negative entries in the feature maps as zeros while keeping all other entries intact.

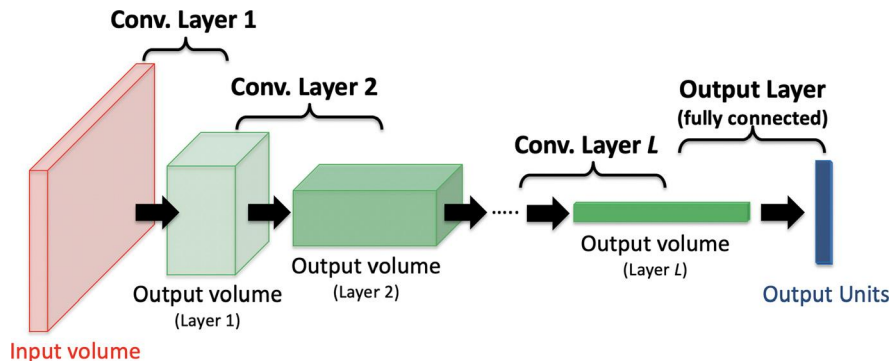
The output feature map tensor from the nonlinear activation is further fed into a pooling operation. The pooling is essentially a down-sampling operation, and it works as follows. For each feature map, we

**FIGURE 10.27**

An example of pooling operation. (a) The original feature map of size  $4 \times 4$ , where each color represents one region for pooling. (b) The max pooling result of the feature map in (a). (c) The average pooling result of the feature map in (a). (d) The small translation of the feature map in (a), by shifting each entry to the left by one position, with zero-padding on the right most column. (e) The max pooling of the translated feature map in (d), which approximately remains the same as the pooling result in (b).

partition it into several nonoverlapping regions (along the width and height dimensions). All the entries in each region are then aggregated into a single entry. The typical aggregations including *maximum* (max pooling) and *average* (averaging pooling). By the pooling operation, we effectively reduce the size of the feature map tensor and thus reduce the overall model complexity to prevent overfitting. The output of the pooling operation is referred to as the output volume (or output for short), which has the same depth as the feature map tensor, but smaller width and depth (see the right part of Fig. 10.26). In addition to reducing the size of output volume, the pooling (especially max pooling) also helps make the output volume to be robust with respect to the perturbation of the feature maps. For example, if values of the entries in the feature maps have small changes, due to either the noise or the small amount of location shift, it is likely that the max pooling output will remain the same. For the latter case, where the output of the max pooling remains (approximately) invariant with respect to the small translation of the input (e.g., a small amount of location shift), it is also referred to as *approximate translation invariant*.

**Example 10.8.** Let us look at an example of pooling operation in Fig. 10.27. The feature map in Fig. 10.27(a) is of size  $4 \times 4$ . We divide it into four regions of size  $2 \times 2$ , and each region is represented by one color. In max pooling (Fig. 10.27(b)), all the entries of the same region are aggregated into the maximum value. For example, the four entries from the top-left corner are aggregated into  $\max(1, 3, 3, 9) = 9$ . Therefore the blue element in (b) is 9. In average pooling (Fig. 10.27(c)), all the entries of the same region are aggregated into the mean value. For example, the four entries from the blue region are aggregated into  $\text{ave}(1, 3, 3, 9) = 4$ . Therefore the blue element in (c) is 4. Other entries in (b) and (c) are computed in a similar way. In Fig. 10.27(d), we translate the original feature map

**FIGURE 10.28**

An example of CNNs, consisting of an input volume,  $L$  convolutional layers, and one fully connected output layer.

in (a) by a small amount (i.e., by shifting all the entries in (a) to the left by one position). By such shifting, the right most column will be empty and, we pad it with zeros. The max pooling result of the shifted feature map (Fig. 10.27(e)) is almost the same as Fig. 10.27(b). In other words, max pooling is approximately translation-invariant with respect to a small amount of translation.  $\square$

**Convolutional Neural Networks (CNNs)** are neural networks that have at least one convolutional layer. Fig. 10.28 presents a typical architecture of CNNs. It takes a 3-D tensor as the input volume, followed by  $L$  convolutional layers, and an output layer. As explained earlier, each convolutional layer typically consists of a set of convolutional operations, followed by ReLU nonlinear activation and the pooling operation.<sup>8</sup> All the kernels of different convolutional layers and the weights and bias values of the output layer form the model parameters of CNNs, which will be learned during the training stage.

Each convolutional layer consists of multiple kernels, each of which shares the same depth as that of the input volume and is separately applied to the input volume. The pooling operation reduces the size of the output volume in terms of width and height. Due to the repeated pooling across different convolutional layers, the width and the height of the output volumes (the green (dark gray in print version) boxes in Fig. 10.28) will keep shrinking, whereas the depth of the output volume is always the same as the number of kernels of the corresponding convolutional layer. For the binary classification, we only need one output unit in the output layer; for the multiclass classification task, we need multiple (as many as the number of classes) output units. Similar to the feed-forward neural network, the output layer is often a fully connected layer, meaning that each output unit is connected to *all* elements of the output volume of the last convolutional layer. We often use the sigmoid activation function at the output layer and use cross-entropy as the loss function.

The number of layers (i.e., the depth) of CNNs often plays a critical role in model performance. This is because, as the model goes deeper, CNNs tend to learn a more complex and powerful representation of the input data (e.g., images), due to the fact that the later (i.e., higher) layer is able to search and

<sup>8</sup> Depending on the specific design of CNNs, the ReLU activation or the pooling operation might be omitted in some convolutional layers.

learn features from a larger receptive field. This, in turns leads to a higher classification accuracy. For example, **LeNet** (one of the early popular CNN architecture, invented in the 1990s) has eight layers. It is now common for a convolutional neural network to have more than 100 layers (e.g., **ResNet**, **DenseNet**). The increasing depth of CNNs has helped significantly improve the image recognition accuracy.

---

## 10.4 Recurrent neural networks

**Recurrent Neural Networks** (RNNs) are powerful models for mining *sequential* data, such as text, audio, time series. For text, RNNs and related techniques have been playing a central role in applications like machine translation (e.g., Google Translate), question answering, sentence classification (e.g., sentiment classification), token classification (e.g., information extraction), and so on. For audio, RNNs have been successfully applied to many important applications like speech recognition and speech synthesis. For time series, RNNs provide an alternative method of CNNs for prediction and anomaly detection. When combined with CNNs, RNNs can be applied to an interesting application called visual question answering (VQA), which provides an answer for a question regarding a given image, and both the question and the answer are in the form of natural language. Other applications of RNNs include protein structure prediction based on the sequence of amino acids in computational biology; temporal recommender systems in e-commerce to model the temporal evolution of users' interest; and robot-assisted minimally invasive surgery (MIS) where the system learns how to tie suture knots based on a special type of RNNs called long short-term memory.

In RNNs, the input is represented as a sequence of values  $(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^T)$ , where  $\mathbf{x}^t$  ( $t = 1, \dots, T$ ) is the value at a particular position or time stamp  $t$ , and  $T$  is the length of the sequence. For natural language processing, the sequence could be a sentence, where  $\mathbf{x}^t$  is the  $t$ th word of the input sentence and  $T$  is the total length (number of words) of the sentence; in time series, the sequence could be a sequence of temperature measurements at different time stamps, where  $\mathbf{x}^t$  is the temperature at the  $t$ th time stamp and  $T$  is the number of the total time stamps. In both applications (and other sequential data analysis), the key is to model the *sequential dependence*. For example, the temperature measures at different time stamps could be strongly correlated; different words in the same sentence are likely to be correlated with each other.

We start with the basic RNN models and their applications in Section 10.4.1. A major challenge in RNNs lies in how to effectively model *long-term dependence*. A powerful family of techniques to address this challenge is called *gated RNNs*, which will be introduced in Section 10.4.2. Additional techniques to address long-term dependence, such as attention mechanism, will be introduced in Section 10.4.3.

### 10.4.1 Basic RNN models and applications

Sequential data naturally arises in many applications. Take the natural language process as an example. Given an input sentence, we might want to predict an overall label for the entire sentence (i.e., *sentence-level classification*, e.g., the sentiment); we might want to predict the same sentence but shifted by one position (i.e., *next word prediction*); we might want to predict a label for each of the input words (i.e., *token-level classification*, e.g., in information extraction, we want to decide whether each word is a location or a person or others); we could predict another sequence of the same language as the

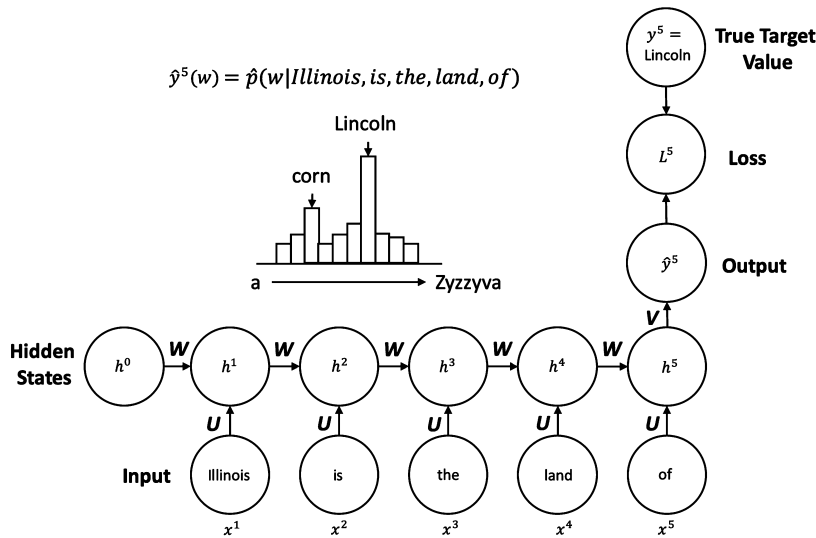


FIGURE 10.29

An example of RNNs, consisting of five input units, five hidden units, and one output unit.

output (i.e., *question-answering*, where the input sentence is the question, and the output sentence is the answer.); or we could predict another sequence in a different language (i.e., *machine translation*). Accurately modeling the sequential dependence is the key to these applications. To see this, suppose we have two sentences, including “Tom beats Jack in the tournament” vs. “Jack beats Tom in the tournament.” These two sentences have exactly the same length, with the same words. However, since the orders of the (same) words are different, the two sentences have completely different meanings. Note that the sequential dependence is not restricted to the nearby positions or time stamps.<sup>9</sup> For example, the temperature at the same location might exhibit daily periodic patterns (i.e., the temperature measures around 8 am on different days might be close to each other); in order to predict the overall sentiment of the entire sentence, we might need to consider two words that are far away from each other in the sentence collectively. We will talk more about this in the next section. RNNs are powerful models to capture such sequential dependence.

“So, what is recurrent neural networks?” Let us take the next-word prediction as an example to look at the basic architecture of RNNs.

**Example 10.9.** Given a (partial) sentence “Illinois is the land of,” we want to predict the next possible word (e.g., “Lincoln” or “Corn”). We can use an RNN model shown in Fig. 10.29.

Formally, there are three main types of units in an RNN model, including **input unit**, **hidden unit**, and **output unit**. Each input unit represents the input data at a given position  $x^l$ . In Example 10.9, the input is a sentence of five words. Therefore there are five input units in Fig. 10.29, each corresponding

<sup>9</sup> In this section, we use the terms “position” and “time stamp” interchangeably.

to a word. For example,  $\mathbf{x}^1 = \text{“Illinois,”}$   $\mathbf{x}^2 = \text{“is,”}$  and so on. For each hidden unit in an RNN model, it is connected to the corresponding input unit *and* the hidden unit at the previous time stamp. For the example in Fig. 10.29, the hidden unit  $\mathbf{h}^2$  is connected to the input unit  $\mathbf{x}^2$  at time stamp 2 and the hidden unit  $\mathbf{h}^1$  at time stamp 1. There are six hidden units in total, five of which ( $\mathbf{h}^1, \dots, \mathbf{h}^5$ ) correspond to the five input units. Notice that the “previous hidden state” for  $\mathbf{h}^1$  should be  $\mathbf{h}^0$ . Since the sequence starts with time stamp 1, we can simply set  $\mathbf{h}^0$  as a zero vector by default. In Fig. 10.29, there is only one output unit  $\hat{\mathbf{y}}^5$  that is connected to the last hidden unit  $\mathbf{h}^5$ . We use the output unit  $\hat{\mathbf{y}}^5$  to predict the next word.

Each hidden unit  $\mathbf{h}^t$  is represented as a vector, which is also called a hidden state. Each input  $\mathbf{x}^t$  and each output  $\hat{\mathbf{y}}^t$  are also represented as a vector respectively. In our example, let the  $N$  be the size of the vocabulary, which contains all the possible words, from “a” to “Zyzyva.” Since each input unit  $\mathbf{x}^t$  represents a single word, we represent  $\mathbf{x}^t$  as an  $N$ -dimensional one-hot vector. Each entry of the vector  $\mathbf{x}^t$  represents a word in the vocabulary; the entry in  $\mathbf{x}^t$  that corresponds to the input word is set as 1, and all other entries in  $\mathbf{x}^t$  are set as zeros. Likewise, the output  $\hat{\mathbf{y}}^t$  ( $t = 5$  in our example) is an  $N$ -dimensional vector, the entry of which represents the *conditional probability* of the corresponding word, given the input up to time stamp  $t$  (i.e.,  $\mathbf{x}^1, \dots, \mathbf{x}^t$ ). In our example,  $\hat{\mathbf{y}}^5(w)$  represents the conditional probability of a word  $w$  given the input partial sentence “Illinois is the land of.” From Fig. 10.29, since the  $\hat{\mathbf{y}}^5(\text{“Lincoln”})$  has the highest conditional probability value among all possible words, the word “Lincoln” is chosen as the predicted next word for “Illinois is the land of.”  $\square$

To summarize, in an RNN model, the input unit is connected to a hidden unit; the hidden unit is connected to the hidden unit at the next time stamp; and optionally, the hidden unit could be connected to an output unit. For the example in Fig. 10.29, only the last hidden unit  $\mathbf{h}^5$  is connected to an output unit  $\hat{\mathbf{y}}^5$ . In order to precisely describe the relationship between different types of units, we need three types of weight matrices, including **input-to-hidden** weight matrix  $\mathbf{U}$ , **hidden-to-hidden** weight matrix  $\mathbf{W}$ , and **hidden-to-output** weight matrix  $\mathbf{V}$ . The RNN model in Fig. 10.29 is formally defined as

$$\mathbf{h}^t = f(\mathbf{U}\mathbf{x}^t + \mathbf{W}\mathbf{h}^{t-1} + \mathbf{a}) \quad (10.19)$$

$$\hat{\mathbf{y}}^T = g(\mathbf{V}\mathbf{h}^T + \mathbf{b}), \quad (10.20)$$

where  $T$  is the length of the input sequence ( $T = 5$  for the example in Fig. 10.29), and  $\mathbf{a}$  and  $\mathbf{b}$  are two bias vectors.  $f()$  is the activation function for the hidden state, and a typical choice for  $f()$  is the tanh function. (The definition of tanh activation function can be found in Fig. 10.6.)  $g()$  is the activation function for the output unit, and a typical choice for  $g()$  is the softmax function that converts a vector of values to a probability distribution. In order to train an RNN model (i.e., to find the optimal weight matrices  $\mathbf{U}$ ,  $\mathbf{W}$ , and  $\mathbf{V}$  and the bias vectors  $\mathbf{a}$  and  $\mathbf{b}$ ), there are some additional nodes in the RNN model in Fig. 10.29, including the true target value ( $\mathbf{y}^T = \text{“Lincoln”}$  in this example) and the loss function  $L$ , which measures the difference between the predicted output  $\hat{\mathbf{y}}^T$  and the true target value  $\mathbf{y}^T = \text{“Lincoln.”}$  We can use either the mean-squared loss or the cross-entropy loss.

Let us take a closer look at Eq. (10.19) and Eq. (10.20). From Eq. (10.19), we can see that each hidden state  $\mathbf{h}^t$  is produced in the following way. First, it takes a linear transformation of the corresponding input  $\mathbf{x}^t$ , that is,  $\mathbf{U}\mathbf{x}^t$ . Meanwhile, it takes a linear transformation of the previous hidden state  $\mathbf{h}^{t-1}$ , that is,  $\mathbf{W}\mathbf{h}^{t-1}$ . Then, it adds the results of the two linear transformations and offsets it by the bias vector  $\mathbf{a}$ , that is,  $(\mathbf{U}\mathbf{x}^t + \mathbf{W}\mathbf{h}^{t-1} + \mathbf{a})$ . Finally, it feeds the sum  $\mathbf{U}\mathbf{x}^t + \mathbf{W}\mathbf{h}^{t-1} + \mathbf{a}$  into the nonlinear activation function  $f()$  to output the hidden state  $\mathbf{h}^t$ . From Eq. (10.20), we can see that the output  $\hat{\mathbf{y}}^T$

is produced in a similar process. That is, it takes a linear transformation of the corresponding hidden state ( $\mathbf{V}\mathbf{h}^t$ ), offsets it by another bias vector  $\mathbf{b}$ , and finally feeds the sum ( $\mathbf{V}\mathbf{h}^t + \mathbf{b}$ ) into the activation function  $g()$  to produce the output  $\hat{\mathbf{y}}^t$ .

If we compare Eqs. (10.19) and (10.20) with the matrix form of feed-forward neural networks ( $\mathbf{h}_i = f(\mathbf{W}_i\mathbf{h}_{i-1} + \mathbf{b}_i)$ ), it seems that they bear some similarity with each other. “So, what is the relationship between RNNs and feed-forward neural networks?” Indeed, if we remove all but the first input unit  $\mathbf{x}^1$  and the hidden state  $\mathbf{h}^0$  from Fig. 10.29, we can view the remaining network as a feed-forward neural network. It takes  $\mathbf{x}^1$  as the input, feed-forwards through five hidden layers (each corresponds to a hidden state  $\mathbf{h}^t$ ) and finally outputs  $\hat{\mathbf{y}}^5$ . However, in this case, the output  $\hat{\mathbf{y}}^5$  (the predicted word) would only be dependent on the first input (the first word “Illinois”). By connecting each hidden state  $\mathbf{h}^t$  to its corresponding input  $\mathbf{x}^t$ , the RNN model enables the predicted output  $\hat{\mathbf{y}}^5$  to be dependent on *all* the input units  $\mathbf{x}^1, \dots, \mathbf{x}^5$ . Another important characteristic of an RNN model that differs itself from feed-forward neural networks is as follows. In RNNs, we use the same hidden-to-hidden weight matrix  $\mathbf{W}$ , the same input-to-hidden weight matrix  $\mathbf{U}$ , and the same bias vector  $\mathbf{a}$  in Eq. (10.19). In other words, Eq. (10.19) is **recurrent** in the sense that the same equation with the same parameters is applied at different time stamps. In contrast, in a feed-forward neural network, the weight matrices and bias vectors in different layers are often different. Eq. (10.19) can also be viewed as a *parameter sharing* technique, in that different input units and different hidden layers share the same weight matrix and bias vector, respectively. In the similar spirit that the parameter sharing in CNNs (i.e., the same kernel is applied at different locations of the input image) significantly reduces the number of the model parameters, there are only three weight matrices ( $\mathbf{U}$ ,  $\mathbf{W}$ , and  $\mathbf{V}$ ) and two bias vectors ( $\mathbf{a}$  and  $\mathbf{b}$ ) in the RNN model, which are significantly less than the number of model parameters of a feed-forward neural network with the same number of layers.

“Well, you might wonder, speaking of CNNs, why do not we use CNNs for sequential data?” Indeed, since sequential data can be viewed as 1-D grid data, we could use a 1-D convolutional neural network to model the input sequence, such as a sentence in Fig. 10.29. However, there are subtle and important differences between 1-D CNNs and RNNs. Recall that the connection in a CNN is always *localized*, in that each entry in the output unit is only determined by the kernel and the receptive field, which is a small neighborhood of the corresponding entry of the input. Therefore for the example in Fig. 10.29, the predicted word  $\hat{\mathbf{y}}^5$  might only depend on the last few words of the input sentence (e.g., “the,” “land,” “of”). In contrast, thanks to its recurrent nature (Eq. (10.19)), an RNN model captures the *long-term dependence*. For the example in Fig. 10.29, the predicted word depends on each of the five input words before it.

For the example in Fig. 10.29, the input sentence “Illinois is the land of” misses the last word. In the RNN model, there is only one output unit  $\hat{\mathbf{y}}^5$  connecting to the last hidden state  $\mathbf{h}^5$ , and we use output unit  $\hat{\mathbf{y}}^5$  to predict the missing last word. In a general next-word prediction setting, we wish to predict *every* next word, given the partial sentence the RNN model has seen so far. To this end, we introduce an output unit  $\hat{\mathbf{y}}^t$ , connecting to each hidden state  $\mathbf{h}^t$ . The output unit is used to predict the next word, given the partial sentence the model has seen so far, including  $\mathbf{x}^1, \dots, \mathbf{x}^t$ . Let us look at an example in Fig. 10.30.

**Example 10.10.** In Fig. 10.30, there are eight input units, eight hidden states, and eight output units, respectively. The input  $\mathbf{x}^t$  ( $t = 1, \dots, 8$ ) represents a sentence “The president Abraham Lincoln was born in Kentucky.” Each hidden state  $\mathbf{h}^t$  is connected to an output unit  $\hat{\mathbf{y}}^t$ , which is used to predict the next word given the partial sentence the model has seen so far ( $\mathbf{x}^1, \dots, \mathbf{x}^t$ ). For instance,  $\hat{\mathbf{y}}^1 = \text{“president”}$  is



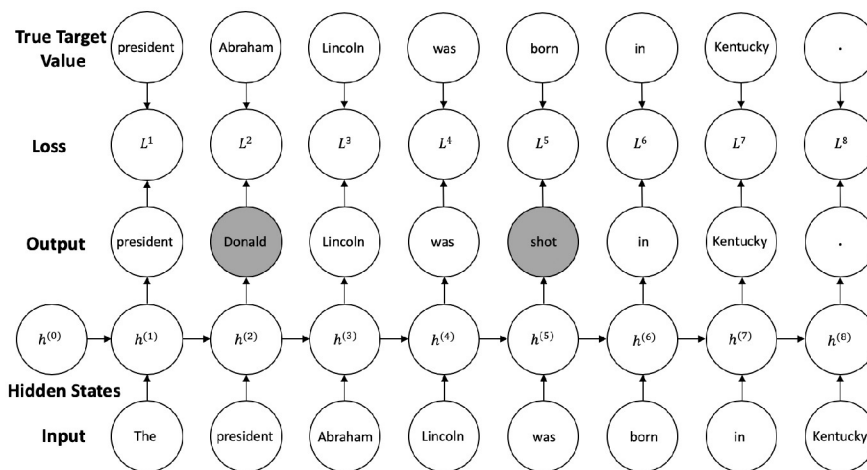


FIGURE 10.30

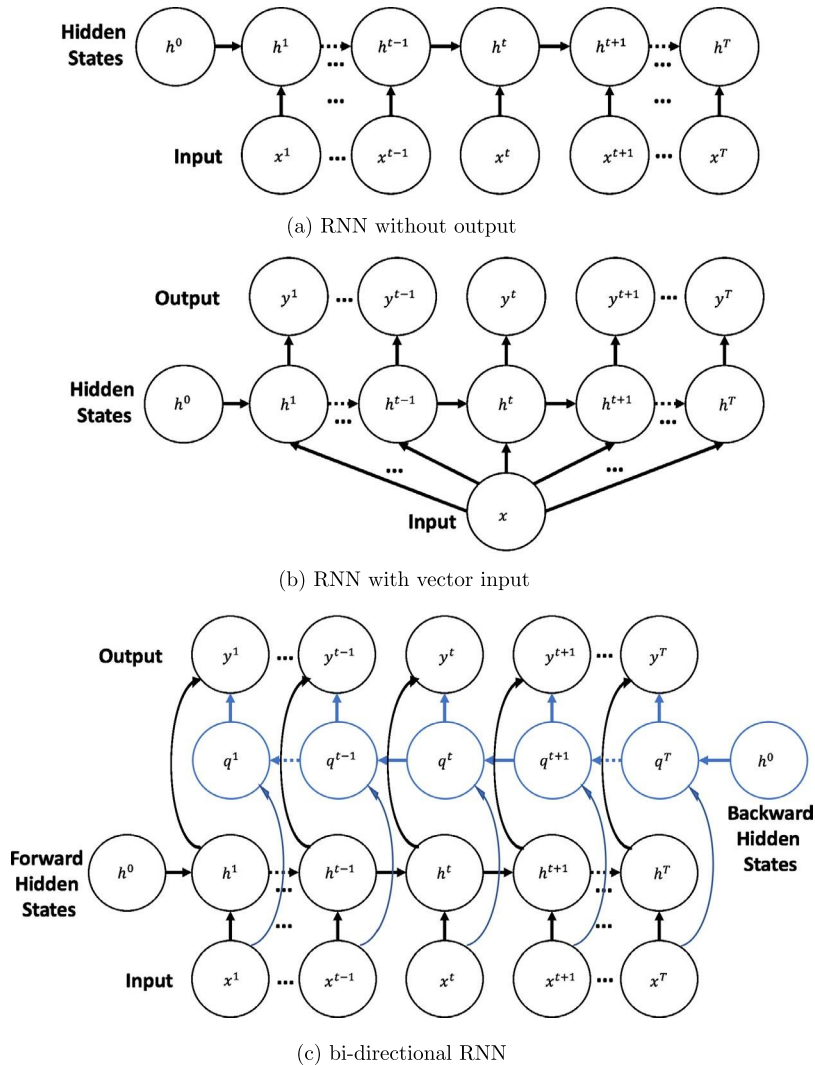
An example of RNNs for next word prediction. Each output unit  $\hat{y}^t$  predicts the next word given the partial sentence the model has seen so far ( $x^1, \dots, x^t$ ). There are eight time stamps  $t = 1, 2, \dots, 8$  in total. The two shaded nodes (“Donald” and “shot”) are wrong predictions. The model parameters, including the input-to-hidden weight matrix, the hidden-to-hidden weight matrix, the hidden-to-output weight matrix, and bias vectors are not shown for clarity.

the predicted next word for the partial sentence “The” (i.e., only the first word  $x^1$ );  $\hat{y}^4 = \text{“was”}$  is the predicted next word for the partial sentence “The president Abraham Lincoln” (i.e., the first four words  $(x^1, x^2, x^3, x^4)$ ). Some predictions, including  $\hat{y}^2 = \text{“Donald”}$  and  $\hat{y}^5 = \text{“shot”}$ , are wrong predictions. Each output unit is compared with the true target value to measure the disagreement (or loss) between them. We train the RNN model (i.e., to find out the best model parameters) by minimizing the total loss  $\sum_{t=1}^8 L^t$ .  $\square$

In addition to next-word prediction, we can use RNN models in Fig. 10.29 and Fig. 10.30 for other applications. For example, the RNN model in Fig. 10.29 can be used for *sentence-level* prediction, which predicts a label for the entire sentence (e.g., the overall sentiment of the input sentence); the RNN model in Fig. 10.30 can be used for *token-level* prediction, which predicts a label for each word (i.e., token) of the input sentence (e.g., the entity type, such as “person” vs. “location” of the corresponding token).

In addition to two RNN models in Fig. 10.29 and Fig. 10.30, there are many different types of RNNs. Fig. 10.31 presents some examples. The RNN model in Fig. 10.31(a) has no output unit at all. This type of RNN is often used to learn a hidden representation of the input sequence. That is, the last hidden state  $h^T$  provides a vector representation of the entire input sequence, which can be in turn used as the input feature vector of data mining tasks, such as classification, clustering, and outlier detection.<sup>10</sup> For the RNN model in Fig. 10.31(b), the input units share the same vector  $x$ . If the input vector  $x$  is the

<sup>10</sup> If the input sequence is a sentence,  $h^T$  is called *sentence embedding*. Likewise, each  $h^t$  ( $t = 1, 2, \dots$ ) provides a vector representation of the corresponding word with the preceding context considered, called *contextualized word embedding*.

**FIGURE 10.31**

Different types of RNNs. (a) An RNN model without any output unit. (b) An RNN model with a vector input. (c) A bi-directional RNN. For clarity, the model parameters of RNNs are not shown.

(hidden) representation of an image (say, learned from a CNN model), the RNN model can be used for image captioning, where the output units produce a set of keywords (one keyword from each output unit) to describe the input image.

For the RNNs model we have introduced so far (Fig. 10.29, Fig. 10.30, and Fig. 10.31(a–b)), the information is propagated from the past to the future (i.e., from  $h^t$  to  $h^{t+1}$ ). For applications like

next-word prediction, this assumption makes sense. However, in some applications, only considering the information in the past to make the prediction about the current state has its limitations. Take information extraction as an example, in order to predict whether the current word is “location,” or “person,” or “others,” it is desirable to consider the words both before and after the current word. In this setting, a reasonable choice is to use *bi-directional RNNs* (Fig. 10.31(c)). In a bi-directional RNN model (Fig. 10.31(c)), we introduce an additional hidden state  $p^t$  (the blue (mid gray in print version) circles in Fig. 10.31(c)), one for each time stamp. Like the hidden state  $h^t$ , each  $p^t$  is connected from the corresponding input unit  $x^t$  and is connected to the corresponding output unit  $\hat{y}^t$  (blue (mid gray in print version) lines in Fig. 10.31(c)). However, the information in the newly introduced hidden units  $q^t$  is propagated from the future to the past (i.e., from  $q^t$  to  $q^{t-1}$ ). By connecting the output unit  $\hat{y}^t$  to both hidden units ( $h^t$  and  $q^t$ ), it is dependent on the input units from both the past and the future (i.e., the entire input sequence).

By combining these RNNs models together, we can apply them to even more complicated applications. One such example is *machine translation*, where we want to automatically translate a sentence in the source language (e.g., “I speak English.” written in English) to another sentence in the destination language (e.g., “Yo hablo inglés.” written in Spanish). One way to do machine translation is connecting two RNN models together as follows. The first RNN model is the one in Fig. 10.31(a), which takes the source sentence as the input, and its last hidden state  $h_S^T$  produces a vector presentation for the entire source sentence. The second RNN is the one in Fig. 10.31(b), which takes the last hidden state of the first RNN model  $h_S^T$  as the input and produces another sentence in the target language. An alternative choice for the second RNN model is to use the RNN model in Fig. 10.30 for next word prediction, and the last hidden state of the first RNN model is used as the initial hidden state of the second RNN  $h_D^0$ . The advantage of this approach is that we are able to leverage what has already been translated so far (which is used as the input of the second RNN model) to help improve the translation accuracy for the remaining sentence. See Fig. 10.32 for an example. We can see that by connecting two RNNs together, we are able to transfer one sequence (e.g., the source sentence) to another sequence (e.g., the target sentence). This is called *sequence-to-sequence* learning or *seq2seq* for short.

## 10.4.2 Gated RNNs

A major challenge of RNNs is **long-term dependence**. Let us go back to the next-word prediction problem in Example 10.9. In this example, it is highly likely that the next word  $w$  is either “Lincoln” or “corn” since they are both the symbols of “Illinois,” and the predicted word  $w$  is quite close to the relevant keyword “Illinois” (i.e., there are only four other words between them). For such cases, an RNN model such as the one in Fig. 10.29 works well. Now, let us look at a harder example.

**Example 10.11.** Given a sentence “The Urbana Sweetcorn Festival is held in every August... There will be live music, food, beers... Enjoy the fresh corn.” without the last word “corn,” we would like to predict it based on the remaining sentence. If we look at the short-term preceding words (e.g., “Enjoy,” “the,” “fresh”), the predicted word  $w$  is likely to be a type of food or entertainment. However, if we want to narrow down the specific food or entertainment, we have to look further back into the input sentence to locate the keyword “Sweetcorn.” In this example, the gap between the predicted word  $w$  and its relevant keyword “Sweetcorn” is large. In other words, the dependence between them is long.  $\square$

Conceptually, the RNN models we have seen so far should be still able to capture such long-term dependence by propagating the information of any preceding word to where we want to make a predic-

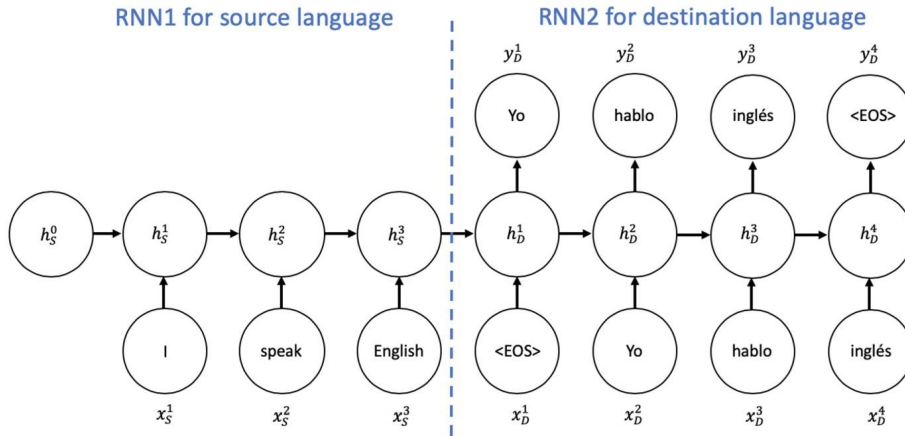
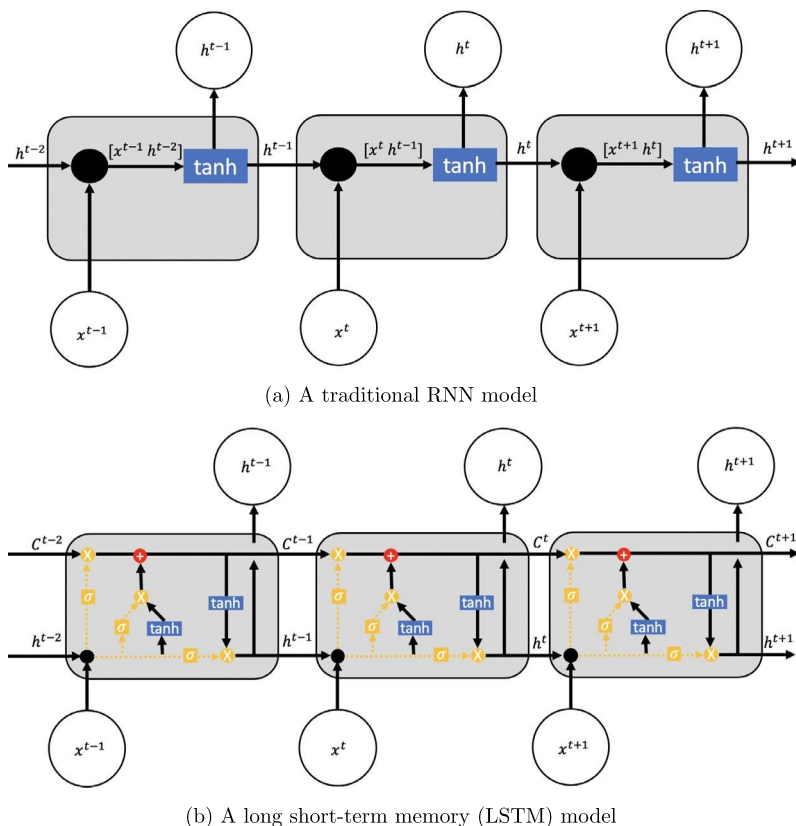


FIGURE 10.32

An example of using two RNN models for machine translation. Note that the first input unit of the second RNN model is a special token <EOS>, which stands for the end of the sentence. This will cause the second RNN to start to produce the first output (“Yo”). The process will continue until the second RNN model outputs a special token <EOS>. Adapted from Charu C. Aggarwal “Neural Networks and Deep Learning.”

tion. However, this means that we will have to use a *deep RNN*, with many hidden states (one for each preceding word). Similar to deep feed-forward neural networks, it is very challenging to train a deep RNN, due to gradient vanishing or gradient exploding problems. Even worse, the gradient vanishing problem or the gradient exploding problem with deep RNNs could be further exacerbated due to its recurrent nature (i.e., the same activation function Eq. (10.19) is repeatedly applied to different hidden states).

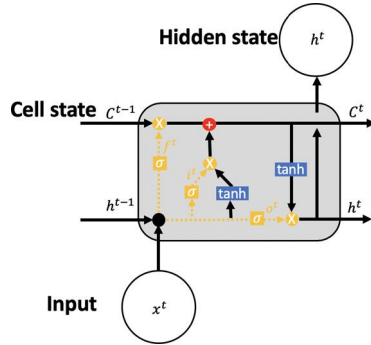
One of the most successful techniques to address the long-term dependence challenge in RNNs is called **long short-term memory** (LSTM) model. Fig. 10.33 provides a pictorial comparison between a traditional RNN model and an LSTM model. Fig. 10.33(a) is the same as the RNN model without any output unit described in Fig. 10.31(a). To see this, we introduce the *vector concatenation* operation that connects two vectors and combines them into a single vector. In Fig. 10.33, the vector concatenation is denoted as a black dot, and it connects the input  $x^t$  and the previous hidden state  $h^{t-1}$ , and combines them into a single vector  $[x^t, h^{t-1}]$  whose length is equal to the sum of the length of  $x^t$  and that of  $h^{t-1}$ . A hidden unit in Fig. 10.33(a) feeds the concatenated vector  $[x^t, h^{t-1}]$  to the nonlinear activation function (e.g., tanh) to generate the new hidden state  $h^t$ , which will be fed into the next hidden unit. Note that this process is equivalent to Eq. (10.19). To see this, let us use tanh as the activation function  $f()$  in Eq. (10.19). We introduce a  $2 \times 2$  block matrix  $\tilde{W}$ , whose first diagonal block is the input-to-hidden weight matrix  $U$ , the second diagonal block is the hidden-to-hidden weight matrix  $W$  in Eq. (10.19), and the two off-diagonal blocks are zeros. Then, Eq. (10.19) becomes  $h^t = \tanh(\tilde{W}[x^t, h^{t-1}] + a)$ , which is the same as the hidden unit described in Fig. 10.33(a). By repeatedly applying the same activation function (i.e., to replace  $h^{t-1}$  by the previous input  $x^{t-1}$  and its preceding hidden state  $h^{t-2}$ , and so on), we can see that the influence of a distant input  $x^{t-\tau}$  on the current hidden state  $h^t$  will decrease rapidly as the gap  $\tau$  increases.

**FIGURE 10.33**

A comparison between a traditional RNN model (a) and an LSTM model (b). Each gray box in (a) is a hidden unit of the RNN model. Each gray box in (b) is an LSTM cell of the LSTM model. The black dot (at the left side of the hidden unit or LSTM cell) denotes vector concatenation operation. For clarity, the output units and the bias vectors are not shown. Adapted from <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

An LSTM model (Fig. 10.31(b)) addresses this issue by replacing the hidden unit in the traditional RNN model by an **LSTM cell**, which is indicated as a gray box in Fig. 10.31(b). Like the traditional RNN model, an LSTM cell concatenates the input  $x^t$  and the previous hidden state  $h^{t-1}$  and eventually produces a new hidden state  $h^t$  that will be fed into the next LSTM cell. However, unlike the traditional RNN model, an LSTM cell has an additional input called **cell state**  $C^t$ . The cell state is the key of the LSTM model. It is specifically designed to accumulate (or to remember) the information from the past for a long time, and thus it is able to address the long-term dependence challenge.

“How does the LSTM cell do the magic to address the long-term dependence challenge?” Let us take a close look at Fig. 10.34. At a given time stamp  $t$ , an LSTM cell takes the previous cell state  $C^{t-1}$  as the input (the top left corner), updates it, and feeds the updated cell state  $C^t$  to the next time stamp


**FIGURE 10.34**

An illustration of an LSTM cell. The black dot represents vector concatenation operation. The two tanh layers (the two blue (mid gray in print version) boxes) are used for generating temporary cell state (the middle bottom) and the temporary hidden state (the right middle), respectively. The three gates (the yellow (light gray in print version) parts) are used to control how much information is passed through the corresponding variables, including the forget gate  $f^t$  for the old cell state, the input gate  $i^t$  for the temporary cell state, and the output gate  $o^t$  for the temporary hidden state, respectively. Adapted from <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

(the top right corner). The cell state is designed to fulfill two seemingly competing objectives. First, it needs to accumulate the information from the past for a long time to address the long-term dependence challenge. Meanwhile, we want to update the cell state based on the current information, and in this way, it is able to “forget” the irrelevant information from the past. The LSTM model achieves these design objectives through three **gates**, namely the forget gate  $f^t$ , the input gate  $i^t$  and the output gate  $o^t$  (the yellow (light gray in print version) parts in the figure).

Each gate takes the concatenated vector  $[x^t, h^{t-1}]$  as the input of a sigmoid activation function  $\sigma(\cdot)$ . Since the sigmoid activation function produces an output between 0 and 1, by multiplying the output of the sigmoid activation function with the corresponding variable (the yellow (light gray in print version) circles marked with  $\times$ ), we can control how much information we want to pass through or remove with respect to the corresponding variable. To be specific, each of the three gates is defined as follows:

$$f^t = \sigma(W_f[x^t, h^{t-1}] + a_f), \quad (10.21)$$

$$i^t = \sigma(W_i[x^t, h^{t-1}] + a_i), \quad (10.22)$$

$$o^t = \sigma(W_o[x^t, h^{t-1}] + a_o), \quad (10.23)$$

where subscripts for the weight matrices and bias vectors are used to differentiate among different gates:  $f$  for the forget gate,  $i$  for the input gate, and  $o$  for the output gate, respectively.

The output of the forget gate  $f^t$  is multiplied with each element of the previous cell state  $C^{t-1}$ . In other words, it is used to control how much information of the previous cell state  $C^{t-1}$  will be removed (or “forgotten”): the smaller the  $f^t$ , the more information will be forgotten. In order to update the cell state, we first feed the concatenated vector  $[x^t, h^{t-1}]$  to a tanh activation function (the blue (mid gray in print version) box in the middle bottom part) to generate a temporary cell state:  $\tilde{C}^t =$

$\tanh(W_c[\mathbf{x}^t, \mathbf{h}^{t-1}] + \mathbf{a}_c)$ , where  $W_c$  and  $\mathbf{a}_c$  are the weight matrix and bias vector for the temporary cell state, respectively. The temporary cell state  $\tilde{\mathbf{C}}^t$  is then multiplied with the input gate  $\mathbf{i}^t$ , which is further used to update the cell state:

$$\mathbf{C}^t = \underbrace{\mathbf{f}^t}_{\text{forget gate}} \cdot \underbrace{\mathbf{C}^{t-1}}_{\text{old cell state}} + \underbrace{\mathbf{i}^t}_{\text{input gate}} \cdot \underbrace{\tilde{\mathbf{C}}^t}_{\text{temporary cell state}}, \quad (10.24)$$

where the forget gate  $\mathbf{f}^t$  controls how much old information from the previous cell state is removed, and the input gate  $\mathbf{i}^t$  controls how much new information from the temporary cell state is added. By adjusting the forget gate and input gate, the LSTM cell strikes a balance between accumulating the past information (from  $\mathbf{C}^{t-1}$ ) and updating with the new information (from  $\tilde{\mathbf{C}}^t$ ).

Finally, the updated cell state  $\mathbf{C}^t$  is used to generate a temporary hidden state  $\tilde{\mathbf{h}}^t = \tanh(\tilde{W}\mathbf{C}^t + \mathbf{a})$ , which is further multiplied with the output gate  $\mathbf{o}^t$  to generate the new hidden state  $\mathbf{h}^t$

$$\mathbf{h}^t = \underbrace{\mathbf{o}^t}_{\text{output gate}} \cdot \underbrace{\tilde{\mathbf{h}}^t}_{\text{temporary hidden state}}. \quad (10.25)$$

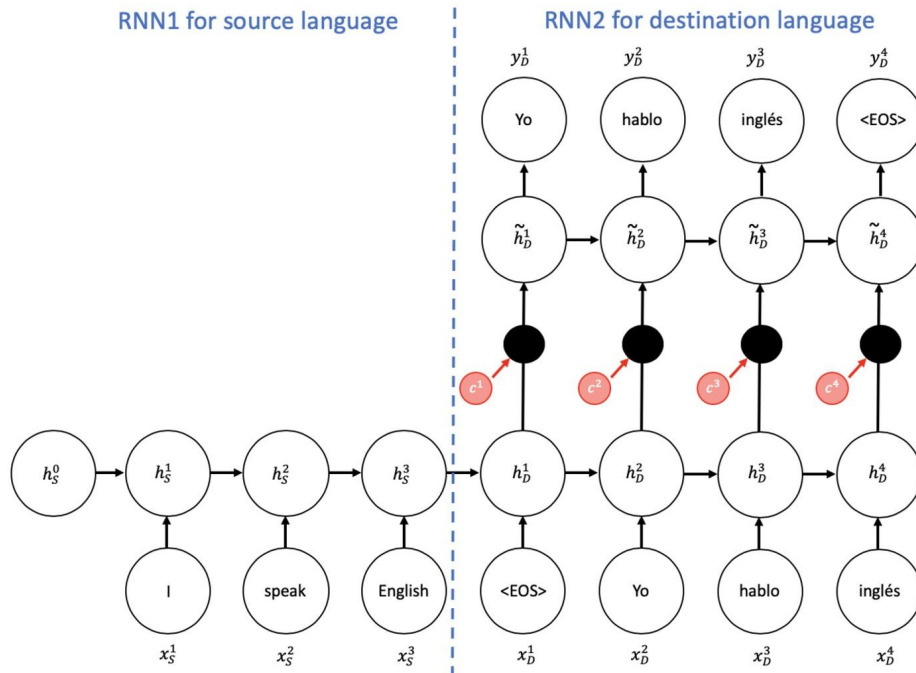
We can see that the key of the LSTM model is to use various gates to control how much information is passed through. In particular, the forget gate  $\mathbf{f}^t$  controls how much information of the old cell state  $\mathbf{C}^{t-1}$  is kept (or forgotten); the input gate  $\mathbf{i}^t$  controls how much new information from the temporary cell state is used to update the cell state vector; and the output gate  $\mathbf{o}^t$  controls how much information of the new cell state is used to generate the new hidden state. RNNs model equipped with gate functions is called **gated RNNs**. Many variants of the LSTM model exist. For example, some variant only uses two gates, by setting the input gate  $\mathbf{i}^t = 1 - \mathbf{f}^t$ ; some variant (called *gated recurrent units*, or GRUs for short) further merges the cell state and hidden state.

LSTM and its variants have been successfully applied in a variety of real-world applications, such as machine translation, speech recognition, image captioning, and question answering.

### 10.4.3 Other techniques for addressing long-term dependence

In addition to gated RNNs, other techniques for addressing long-term dependence exist. For example, we can apply the *shortcut connections* method (Section 10.2.7) to add additional links between non-adjacent hidden states; alternatively, we can replace some links between adjacent hidden states by longer connections; we can also add self-connections to the hidden units with the weights close to 1 in order to “remember” the historical information (such units are called leaky units).

Another alternative strategy to address long-term dependence is through **attention** mechanism. In the machine translation example (Fig. 10.32), the sentence in the source language is summarized as the hidden state of the last time stamp of the first RNN model, which is used as the initial hidden state of the second RNN model. For short sentences, this treatment works fine. However, when the sentence to be translated becomes longer or even becomes paragraphs, the influence of such a summary vector on translation becomes weaker and weaker as the time stamp of the second RNN increases. In other words, the gap between the summary vector  $\mathbf{h}_S^T$ , and the current word to translate  $\mathbf{x}_D^t$  becomes larger.



**FIGURE 10.35**

An example of using attention mechanism in RNN models for machine translation. The black dot represents the vector concatenation operation. The pink (light gray in print version) dots ( $c^1, c^2, c^3, c^4$ ) are context vectors. The context vector  $c^j$  is concatenated with the hidden state  $h_D^j$  to produce a new hidden state  $\tilde{h}_D^j$ , which is in turn used to produce the output  $y_j$ .

The attention mechanism addresses this issue by augmenting each hidden state of the second RNN model with an additional input, called *context* vector. For the example in Fig. 10.32, we augment each of the four hidden states of the second RNN model with a context vector, and the resulting RNN model is shown in Fig. 10.35. Intuitively, for a given time stamp  $j$  of the second RNN model, the context vector  $c^j$  *attends* to one or more hidden states of the first RNN model that are most relevant to the current hidden state  $h_D^j$ . In this way, we will be able to pass the most relevant information from the first RNN model (the sentence in the source language) to the current time stamp  $j$ , regardless of how far away they are apart in the RNN model. The context vector  $c^j$  is concatenated with the hidden state  $h_D^j$  as the new hidden state  $\tilde{h}_D^j$ , which is used to produce the corresponding output  $y_D^j$  (the translated word at the  $j$ th location).

“How can we obtain the context vectors?” If the words in the source language and the destination language are perfectly aligned (i.e., the  $j$ th word in the destination language always corresponds to the  $j$ th word in the source language), we can simply set  $c^j = h_s^j$ . However, due to the grammar difference between different languages, the perfect alignment almost never exists. In this case, we introduce an



*alignment vector*  $\mathbf{a}$ . The alignment vector  $\mathbf{a}$  has the same length of the sequence in the source language and its elements measure the relevance between the corresponding hidden state of the first RNN model and the current hidden state of the second RNN model:  $\mathbf{a}(i) = \text{Rel}(\mathbf{h}_S^i, \mathbf{h}_D^j)$ , where  $\text{Rel}()$  is a relevance score function. We further require that  $\sum_i \mathbf{a}(i) = 1$  and  $\mathbf{a}(i) \geq 0$ . In other words, we can interpret  $\mathbf{a}(i)$  as the probability that the  $i$ th hidden state of the first RNN model is aligned with the  $j$ th hidden state of the second RNN model. With the alignment vector  $\mathbf{a}$ , the context vector is formally defined as

$$c^j = \sum_i \mathbf{a}(i) \mathbf{h}_S^i. \quad (10.26)$$

In other words, the context vector  $c^j$  is a weighted average of the hidden states of the first RNN model, where the weights are based on the relevance of the corresponding hidden state of the first RNN model with respect to the hidden state  $\mathbf{h}_D^j$  of the second RNN model. If all elements of the alignment vector  $\mathbf{a}$  are nonzeros, each hidden state will attend the hidden state  $\mathbf{h}_D^j$ , which is referred to as *global attention*; if only some elements of the alignment vector  $\mathbf{a}$  are nonzeros, it is referred to as *local attention*; and if only one element of the alignment vector  $\mathbf{a}$  is nonzero, it is referred to as *hard attention*.

A typical way to compute the alignment vector  $\mathbf{a}$  is to take the softmax of a score function between different hidden states

$$\mathbf{a}(i) = \frac{\exp(\text{score}(\mathbf{h}_S^i, \mathbf{h}_D^j))}{\sum_{i'} \exp(\text{score}(\mathbf{h}_S^{i'}, \mathbf{h}_D^j))}, \quad (10.27)$$

where  $\text{score}(\mathbf{h}_S^i, \mathbf{h}_D^j)$  can be viewed as the unnormalized relevance score between two hidden states. We can use the dot product as the score function:  $\text{score}(\mathbf{h}_S^i, \mathbf{h}_D^j) = \mathbf{h}_S^i \cdot \mathbf{h}_D^j$ . Alternatively, we can take a bilinear form as the score function:  $\text{score}(\mathbf{h}_S^i, \mathbf{h}_D^j) = (\mathbf{h}_S^i)^T \mathbf{W}_a \mathbf{h}_D^j$ , where the matrix  $\mathbf{W}_a$  is the parameter and  $T$  is the vector transpose operation. We can even use a single layer full-connected neural network to calculate the score function, with the two hidden states ( $\mathbf{h}_S^i$  and  $\mathbf{h}_D^j$ ) as the input. The scores (called attention weights) provide a natural way for interpretation as they tell the relative importance of different hidden states  $\mathbf{h}_S^i$  ( $i = 1, 2, \dots$ ) with respect to the current hidden state  $\mathbf{h}_D^j$ .

The attention mechanism provides an effective way to address the long-term dependence. A powerful deep learning architecture for text mining, called *Transformer*, is built entirely based on a specific type of attention mechanism called *self-attention* without any recurrent neural networks. At the time *Transformer* was invented, it outperformed the best RNN-based methods on several sequence-to-sequence tasks, such as machine translation. The attention mechanism in *BERT*, which stands for Bidirectional Encoder Representations from Transformers, is also credited for its strong performance in many natural language processing tasks.

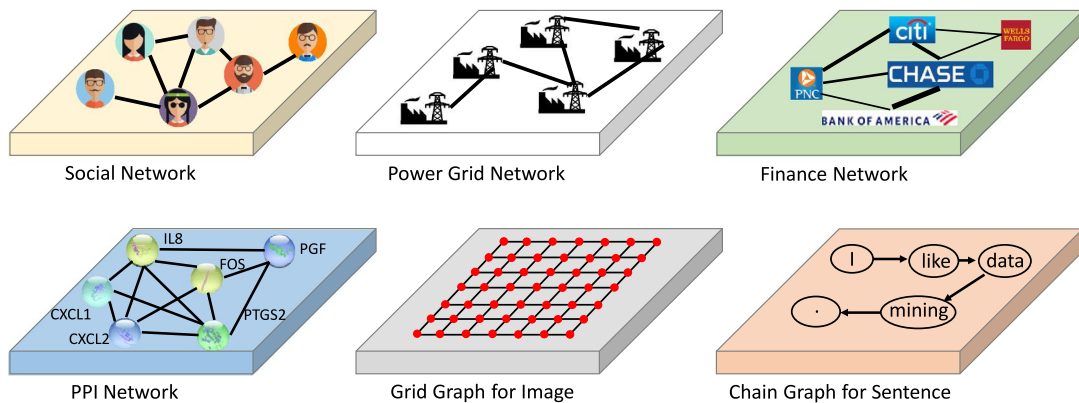
The cell state in an LSTM model can be viewed as an internal or implicit memory to address the long-term dependence challenge. If we manage the memory separately as the explicit or external memory, and let the data mining model (such as an RNN model) access the memory through *read* and *write* operations, it becomes a powerful technique called *memory mechanism*. Examples include *neural Turing machines* and *memory networks*.

## 10.5 Graph neural networks

Graph data,<sup>11</sup> which is essentially a collection of *nodes* (or vertices) linked with each other by *edges*, is a ubiquitous data type arising in many applications. For example, in a social network, nodes are users and edges represent the friendship between two users; in a power-grid, nodes are power plants and edges are the power lines connecting two power plants; in a financial transaction network, nodes are financial accounts and edges are the transactions between two accounts; in bioinformatics, nodes of a protein-protein interaction (PPI) network are proteins and edges indicate the interactions between different proteins. Fig. 10.36 shows some graphs from various applications.

The high-level idea of **graph neural networks** (GNNs) is to turn graph data into multidimensional data, where each node of the graph is represented as a multidimensional vector. For example, if we can represent each user in a social network as a multidimensional vector, we can train a classification model (e.g., decision trees, logistic regression) to classify if the given user is likely to leave the social network site (churn prediction); alternatively, we can run a clustering algorithm on the vector representation of users to find a group of similar users (community detection). Likewise, if we can represent each financial account in a financial transaction network as a vector, we can apply outlier detection techniques (which will be introduced in Chapter 11) to find fraudulent accounts (fraud detection); or we can feed the vector representation of power plants of a power grid into a regression model to forecast the likelihood that a power plant might fail in the near future (failure prediction); we can calculate the similarity between a user and an item based on their vector representation derived from a user-item rating graph to predict which item(s) the user might like (recommender systems).

“So, how can we turn the graph data into multidimensional vectors?” In this section, we start with some basic concepts of graph neural networks (GNNs)—deep learning models designed to answer this



**FIGURE 10.36**

Graphs from various applications.

<sup>11</sup> In literature, graph data is also referred to as network data. In this section, we use the term *graphs* and *networks* interchangeably.

question (Section 10.5.1). Then, we will introduce a particular type of GNNs called graph convolutional networks (GCNs) (Section 10.5.2) and other types of GNNs (Section 10.5.3).

### 10.5.1 Basic concepts

Mathematically, a graph with  $n$  nodes can be represented by its *adjacency matrix*  $A$  of size  $n \times n$ . The rows and columns of the adjacency matrix  $A$  represent the nodes. Given two nodes  $i$  and  $j$ , if there is a connection between them, we set the corresponding entries of the adjacency matrix as 1s (i.e.,  $A(i, j) = A(j, i) = 1$ ); otherwise, we set  $A(i, j) = A(j, i) = 0$ .<sup>12</sup> In addition, nodes or edges of a graph might have attributes, which can be represented as a node (or edge) attribute matrix  $X$ , whose rows are nodes (or edges), columns are attributes, and entries of the matrix are the attribute values. Note that the grid (e.g., images) and sequence (e.g., text) introduced in Sections 10.3 and 10.4 can be represented as special types of graphs as well (see Fig. 10.36). For example, we can represent an image as a *grid graph*, where nodes are pixels, nearby pixels are connected with each other by edges, and RGB colors are treated as the attributes of the pixels; and we can represent a sentence as a *chain graph*, where nodes are words and adjacent words are linked with each other by edges.

**Example 10.12.** Consider an undirected and unweighted social network in Fig. 10.37(a) with six users, and each user has three attributes, including gender, age, and occupation. We represent it by a  $6 \times 6$  adjacency matrix  $A$  (Fig. 10.37(b)) and a  $6 \times 3$  node attribute matrix  $X$  (Fig. 10.37(c)). Each row and column of the adjacency matrix  $A$  represent a user (e.g., the first row and column represent User 1; the second row and column represent User 2, etc.). Each entry of the adjacency matrix  $A$  indicates if the two corresponding users are connected. For example, since User 1 and User 3 are connected, we have that  $A(1, 3) = A(3, 1) = 1$ ; since User 2 and User 5 are not connected, we have that  $A(2, 5) = A(5, 2) = 0$ . Each row of the attribute matrix  $X$  represents a user, and each column of the attribute matrix  $X$  represents an attribute. For example, the first column of  $X$  represents whether the user is male (1) or female (0), the second column of  $X$  represents whether the user is young (1) or senior (0), and the third column of  $X$  represents whether the user is a student (1) or a professor (0). The entry of the attribute matrix  $X$  is the attribute value of the corresponding user on the corresponding attribute. For example, since the first user is a young male student, we have that  $X(1, 1) = 1$  (male),  $X(1, 2) = 1$  (young), and  $X(1, 3) = 1$  (student).  $\square$

The goal of GNNs is to use neural networks to map nodes (or subgraphs or graphs themselves) to low-dimensional vectors. For the example in Fig. 10.37, we might use GNNs to map each user into a 2-D vector (Fig. 10.37(d)) (We will introduce how to compute the specific value of these 2-D vectors in a minute.) Such low-dimensional vectors are also referred to as *embedding or representation* of the nodes. For this reason, the term graph neural networks is sometimes used interchangeably with *graph embedding* or *network representation learning*.<sup>13</sup> We choose the term of graph neural networks

<sup>12</sup> For brevity, we assume the graph is *undirected* (which means that an edge from node  $i$  to node  $j$  always implies there is another edge from node  $j$  to node  $i$ ) and *unweighted* (which means that the adjacency matrix  $A$  is a binary matrix). We can generalize the techniques described in this section to a directed and weighted graph as well.

<sup>13</sup> Do not confuse the term “network” in network representation learning with the term “network” in GNNs. For the former, the term “network” refers to the input graph data.

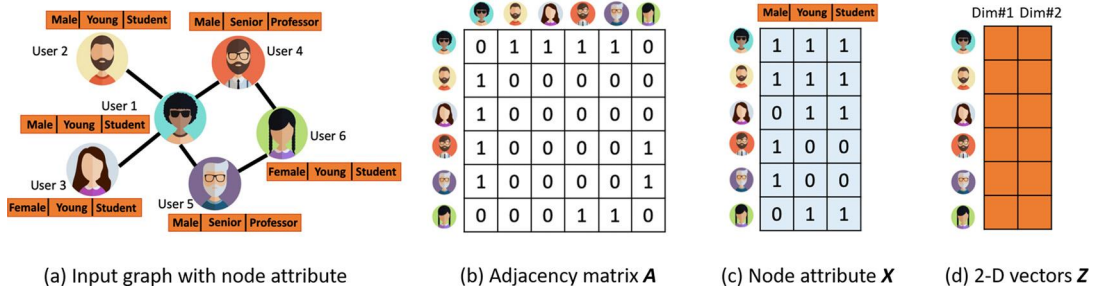


FIGURE 10.37

A toy social network (a). The topology of the social network is represented by its adjacency matrix  $A$  (b) and the node attributes are represented by the attribute matrix  $X$  (c). (d) The 2-D embedding vectors of nodes are denoted by  $Z$ .

to emphasize the use of neural networks techniques to learn the node embedding. The learned low-dimensional vectors (i.e., the node embedding) can be used as the input of a variety of downstream applications, such as classification, clustering, outlier detection, and link prediction.

### 10.5.2 Graph convolutional networks

An effective type of GNNs is called GCNs. Given an input graph  $A$  with node attributes  $X$ , a GCN model uses a set of weight matrices  $W^l$  ( $l = 1, \dots, L$ ), one weight matrix for each layer, to produce the node embedding matrix  $Z^l$  whose rows are the embedding of the corresponding nodes. The algorithm is summarized in Fig. 10.38. The steps are described next.

**Preprocessing and initialization:** We first add a self-edge to each node  $i$ . For a given node  $i$ , this will enable the GCN model to “remember” its own embedding while aggregating its neighboring nodes’ embedding that will be described next. In terms of the adjacency matrix, adding a self edge to each node is equivalent to updating the adjacency matrix by an identity matrix  $I$  (Step 1), where  $I(i, i) = 1$  and  $I(i, j) = 0 \forall i \neq j$  ( $j = 1, \dots, n$ ). Then, in Step 2, we calculate the degree matrix  $D$  of the updated adjacency matrix  $A$ , where  $D(i, i) = \sum_{j=1}^n A(i, j)$  and  $D(i, j) = 0 \forall i \neq j$  ( $i, j = 1, \dots, n$ ). Using the degree matrix  $D$ , in Step 3, we calculate a normalized matrix  $\hat{A}$  as  $\hat{A} = D^{-1/2} A D^{-1/2}$ , which is also referred to as the normalized graph Laplacian of matrix  $A$ . Each element in the matrix  $\hat{A}$  is obtained by normalizing the corresponding element in  $A$  by the square root of the degrees of the source and the target nodes of the given element:  $\hat{A}(i, j) = \frac{A(i, j)}{\sqrt{D(i, i)} \sqrt{D(j, j)}}$ . In Step 4, the initial embedding is simply set as the input node attribute matrix  $Z^0 = X$ .

**Producing node embedding layer-by-layer:** GCNs produce the node embedding at a given layer  $Z^l$  based on the embedding from the previous layer  $Z^{l-1}$  through the following three steps. First (Step 6), for each dimension  $p$  of the previous node embedding of each node  $i$ , we take a weighted aggregation of the embedding of its neighboring nodes:  $\tilde{Z}^{l-1}(i, p) = \sum_{j=1}^n \hat{A}(i, j) Z^{l-1}(j, p)$ . This step is also referred to as *propagation*, and in the matrix form, we have that  $\tilde{Z}^{l-1} = \hat{A} Z^{l-1}$ . Second

**Algorithm: Graph convolutional networks** for producing node embedding at different layers.

**Input:**

- $A$ , adjacency matrix of the input graph of size  $n \times n$ ;
- $X$ , the node attribute matrix of size  $n \times d$ ;
- $W^l$  ( $l = 1, \dots, L$ ), the weight matrix at each layer;
- $L$ , the number of layers;
- $f$ , nonlinear activation function (e.g., sigmoid or ReLU function).

**Output:** The node embedding matrices  $Z^l$  ( $l = 1, \dots, L$ ).

**Method:**

```

//Preprocessing and Initialization
(1) Add a self-edge for each node:  $A \leftarrow A + I$  where  $I$  is an identity matrix;
(2) Calculate the degree matrix  $D$  of  $A$ ;
(3) Normalize  $\hat{A} = D^{-1/2}AD^{-1/2}$ ;
(4) Initialize  $Z^0 = X$ ;
(5) for ( $l = 1, \dots, L$ ) { // for each layer of GCNs
    //Propagation
(6)  $\tilde{Z}^{l-1} = \hat{A}Z^{l-1}$ ; // aggregate the neighboring embedding
    //Linear Transformation
(7)  $\tilde{Z}^l = \tilde{Z}^{l-1}W^l$ ; // linear transformation of aggregated embedding
    //Nonlinear activation
(8)  $Z^l = f(\tilde{Z}^l)$ ; // nonlinear activation of linearly transformed embedding
(9) }

```

**FIGURE 10.38**

Graph convolutional networks. We assume the weights are given and the bias vectors are omitted for brevity.

(Step 7), we take a linear transformation of the aggregated embedding through the weight matrix:  $\tilde{Z}^l = \tilde{Z}^{l-1}W^l$ . That is, we take a linear weighted sum of different dimensions of the aggregated embedding:  $\tilde{Z}^l(i, p) = \sum_{j=1}^n \tilde{Z}^{l-1}(i, j)W^l(j, p)$ . Third (Step 8), we pass the linearly transformed embedding matrix  $\tilde{Z}^l$  through a nonlinear activation function, such as sigmoid function or ReLU function.

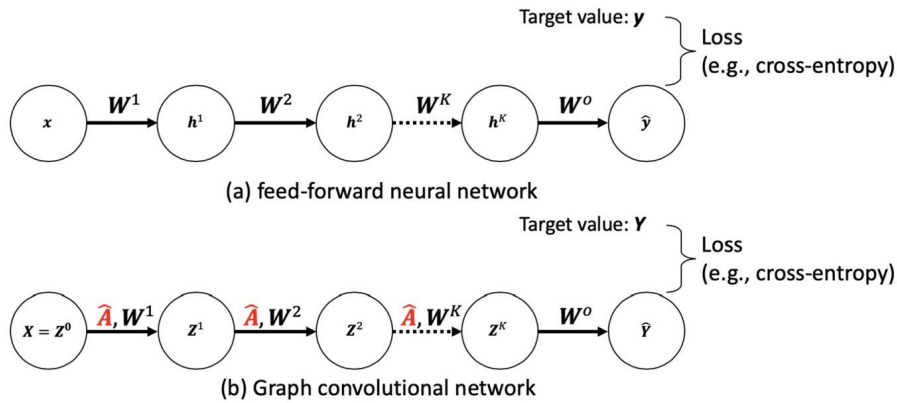
Notice that the last two steps are similar to a fully connected layer of a feed-forward neural network, with the aggregated embedding matrix  $\tilde{Z}^{l-1}$  as the input, the weight matrix  $W^l$  as the model parameters, and the node embedding matrix  $Z^l$  in Step 8 as the output of this layer. That is,

$$Z^l = f(\tilde{Z}^{l-1}W^l). \quad (10.28)$$

Eq. (10.28) is called batch implementation, in that we have stacked the embedding of all  $n$  nodes (i.e., samples) into an  $n \times d^l$  matrix  $Z^l$ , where  $d^l$  is the embedding dimension of layer- $l$ , and each row of  $Z^l$  is the embedding of one node (i.e., sample). It is not hard to re-write Eq. (10.28) with respect to each individual node or sample

$$Z^l(i, :) = f(\tilde{Z}^{l-1}(i, :)W^l), \quad (10.29)$$

where  $Z^l(i, :)$  is the embedding of node  $i$  of the  $l$ th layer ( $i = 1, \dots, n$ ). A subtle point with respect to notation is that GCNs typically use a *row* vector (e.g.,  $Z^l(i, :)$ ) to represent the embedding of a node;



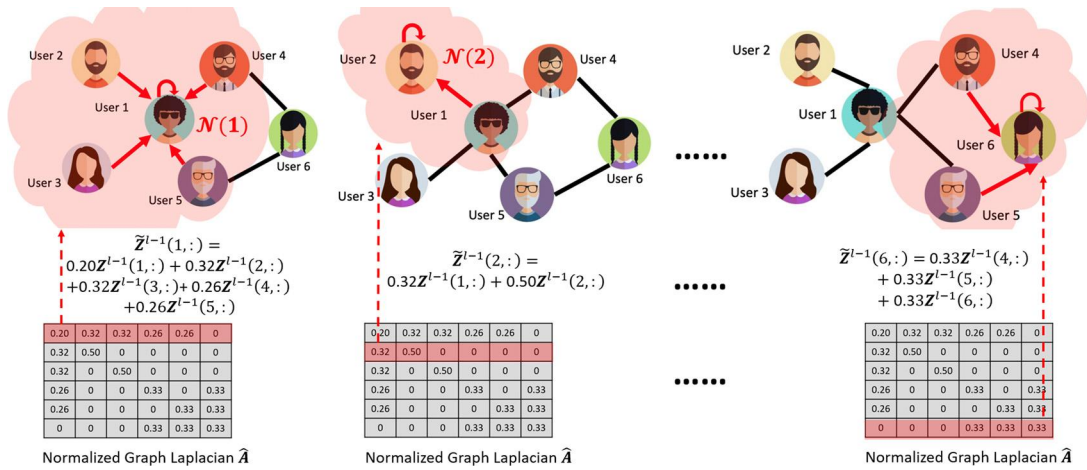
**FIGURE 10.39**

A conceptual comparison between GCNs (b) and feed-forward neural networks (a). In the matrix form, the main difference lies in the extra propagation (i.e., aggregation) step through the normalized graph Laplacian matrix  $\hat{A}$  in GCNs.

whereas in other deep learning models (e.g., feed-forward neural networks, CNNs, RNNs) and other GNNs models (e.g., GraphSAGE that will be introduced next), the embedding of a data sample (e.g., a node) is represented by a *column* vector by convention. These two forms can be converted with each other. For example, Eq. (10.29) is equivalent to  $(Z^l(i, :))^T = f((W^l)^T (\tilde{Z}^{l-1}(i, :))^T)$ , where the column vector  $(Z^l(i, :))^T$  is the transpose of the row vector  $Z^l(i, :)$ .

Conceptually, the main difference between GCNs and feed-forward neural networks lies in the propagation (Step 6). That is, we first propagate (or aggregate) the node embedding matrix  $Z^{l-1}$  from the previous layer and then use the aggregated embedding matrix  $\tilde{Z}^{l-1} = \hat{A}Z^{l-1}$ , instead of  $Z^{l-1}$ , as the input of this layer (layer- $l$ ). See Fig. 10.39 for a comparison.

“But, why do we want to introduce this extra (Propagation or Aggregation) step? Which part of the GCNs algorithm in Fig. 10.38 corresponds to convolution?” Recall that for 2-D convolution, the filter is a fixed, small-sized (e.g.,  $3 \times 3$ ) matrix. For a given element of the input matrix (e.g., a pixel), we place the center of the filter on top of the pixel and thus identify a local neighborhood of the pixel, called the receptive field. Then, we take a weighted sum of all the elements in the receptive field, weighted by the corresponding elements in the filter, as the element in the feature map. However, we cannot directly apply this strategy to perform convolution operations on graphs because the neighborhood of different nodes often have different sizes, and there is no natural ordering of nodes in the neighborhood. The propagation in Step 6 provides a clever way to address these issues as follows. First, the nonzero entries in the  $i$ th row of matrix  $\hat{A}$  define the neighborhood of node  $i$ :  $\mathcal{N}(i) = \{j | j \neq i, \hat{A}(i, j) \neq 0\}$  (analogous to the receptive field in 2-D convolution). Then, we take a weighted sum of the embedding of node  $i$ ’s neighbors, weighted by the corresponding elements in  $\hat{A}$  (analogous to the filter in the 2-D convolution). Finally, the aggregated embedding from the neighborhood is combined with node  $i$ ’s current embedding by a linear weighted sum as the updated embedding of node  $i$  (analogous to the



**FIGURE 10.40**

An illustration of the convolution operation in GCNs. The nonzero elements in each row of the normalized graph Laplacian matrix  $\hat{A}(i, :)$  identify the neighborhood  $\mathcal{N}(i)$  of node  $i$ , plus node  $i$  itself (i.e., the pink (light gray in print version) shaded area). In order to update node  $i$ 's embedding  $\tilde{Z}^{l-1}(i, :)$ , we first aggregate the embedding of its neighboring nodes' embedding. Equivalently, we can think of this process as propagating neighboring nodes' embedding to the current node  $i$ . Then, we take a linear weighted sum of the aggregated embedding from the neighborhood and node  $i$ 's current embedding.

corresponding element in the feature map in 2-D convolution):  $\tilde{Z}^{l-1}(i, :) = \sum_{j=1}^n \hat{A}(i, j)Z^{l-1}(j, :) = \sum_{j \in \mathcal{N}(i)} \hat{A}(i, j)Z^{l-1}(j, :) + \hat{A}(i, i)Z^{l-1}(i, :)$ . See Fig. 10.40 for an illustration.

**Example 10.13.** Let us walk through the GCN algorithm using the example in Fig. 10.12, where we consider the first GCN layer (i.e.,  $l = 1$ ). The computational results are summarized in Fig. 10.41. Given the input  $Z^0 = X$  and the normalized graph Laplacian matrix  $\hat{A}$  (computed in Steps 1–3), we first compute the weighted aggregation of the embedding of neighboring nodes by Step 6 and output  $\tilde{Z}^{(l-1)}$ . Then we compute the linear transformation of  $\tilde{Z}^{(l-1)}$  (i.e., Step 7). Here we assume the weight matrix  $W^1$  at the first layer is fixed. The output of this step  $\tilde{Z}^l$  is then passed through a nonlinear activation (we use the sigmoid function in this example), and we obtain the output node embedding matrix  $Z^1$ . □

The GCN algorithm in Fig. 10.38 assumes that the weight matrices  $W^l$ , ( $\forall l = 1, \dots, L$ ) are given. In case these parameters are unknown, but we do know the class labels for a subset of nodes, one way to learn such model parameters is as follows. We augment the last layer of Fig. 10.38 with a fully connected layer without the propagation step (e.g., with sigmoid activation function). Then, we can use the backpropagation algorithm (Fig. 10.4) to learn the weight matrices  $W^l$  (e.g., with the cross-entropy as the loss function).

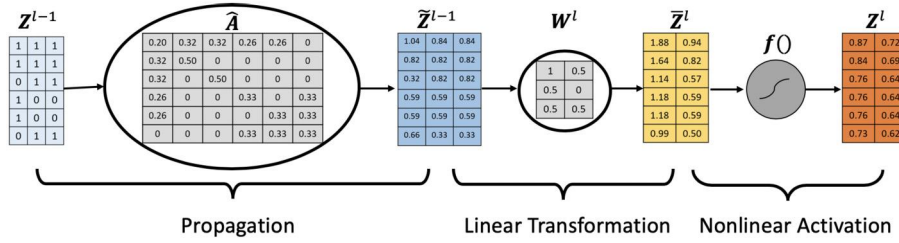


FIGURE 10.41

An example of one-layer GCN computation steps ( $l = 1$ ).

### 10.5.3 Other types of GNNs

For the GCNs model in Fig. 10.38, we aggregate the embedding of the neighboring nodes by a weighted sum, where the weights are determined by the corresponding entries in the normalized graph Laplacian matrix  $\hat{\mathbf{A}}$ . Many alternative choices for aggregation exist, which lead to other types of GNN models.

Unlike GCNs where the node embedding is represented by a row vector, these other types of GNNs often use column vectors to represent node embedding. To avoid notation confusion with GCNs, we use  $\mathbf{h}_i^l$ , a *column vector*, to represent the embedding of node  $i$  in the  $l$ th layer in this section. Likewise, the attributes of a node are presented by a column vector  $\mathbf{x}_i$  ( $i = 1, \dots, n$ ).

**GraphSAGE**, which stands for Graph SAmple and aggreGatE, can be viewed as a generalization of graph convolutional networks. The algorithm of GraphSAGE is summarized in Fig. 10.42. We can see that the main difference between GraphSAGE and GCNs lies in Step 4, where we use an aggregation function ( $\text{agg}$ ) to aggregate the embedding of neighboring nodes. In addition to taking a linear weighted sum of neighboring nodes’ embedding as in GCNs, the function  $\text{agg}$  in GraphSAGE also supports other types of aggregation. For example, we can use *pooling aggregator* as follows:

$$\tilde{\mathbf{h}}_i^{l-1} = \max(g(\mathbf{W}_{\text{pool}}\mathbf{h}_j^{l-1} + \mathbf{b}), \forall j \in \mathcal{N}(i)), \tag{10.30}$$

where  $g$  is the activation function for the pooling aggregator, and  $\mathbf{W}_{\text{pool}}$  and  $\mathbf{b}$  are the corresponding weight matrix and bias vector, respectively. A pooling aggregator first feeds the embedding of neighboring nodes through a fully connected layer and then takes the element-wise maximum of the output of the fully connected layer as the aggregation result. In addition to the pooling aggregator, we can also use an LSTM model applied to a random permutation of the neighboring nodes to aggregate the embedding of the neighboring nodes. Another subtle difference between GraphSAGE and GCNs lies in Step 5, where GraphSAGE uses a separate weight matrix  $\mathbf{U}^l$  applied to node  $i$ ’s current embedding  $\mathbf{h}_i^{l-1}$ .<sup>14</sup>

Many other types of GNNs have been developed. For example, in a *propagation-based spatial graph convolutional network*, we can simply add the embedding of the neighboring nodes as the aggregated embedding  $\tilde{\mathbf{z}}^{l-1}$ . In *Graph Attention Network* (GAT), it uses the *attention* mechanism (similar

<sup>14</sup> This is equivalent to first concatenating  $\tilde{\mathbf{h}}_i^{l-1}$  and  $\mathbf{h}_i^{l-1}$  and then applying a linear transformation on the concatenated vector.



**Algorithm: GraphSAGE** for producing node embedding at different layers.

**Input:**

- $A$ , adjacency matrix of the input graph of size  $n \times n$ ;
- $\mathbf{x}_i (i = 1, \dots, n)$ , the node attribute vector of length  $d$ ;
- $\mathbf{W}^l, \mathbf{U}^l (L = 1, \dots, L)$ , the weight matrices at each layer;
- $L$ , the number of layers;
- $f$ , nonlinear activation function (e.g., sigmoid or ReLU function);
- `agg`, aggregation method.

**Output:** The node embedding vectors at different layers  $\mathbf{h}_i^l (l = 1, \dots, L, i = 1, \dots, n)$ .

**Method:**

```

(1) Initialize  $\mathbf{h}_i^0 = \mathbf{x}_i (i = 1, \dots, n)$ ;
(2) for  $(l = 1, \dots, L)$  { // for each layer of GraphSAGE
(3)     for  $(i = 1, \dots, n)$  { // for each node
//Neighborhood aggregation via an aggregation method Agg
(4)          $\tilde{\mathbf{h}}_i^{l-1} = \text{Agg}(\mathbf{h}_j^{l-1}, \forall j \in \mathcal{N}(i)); //\mathcal{N}(i)$ : neighborhood of node  $i$ 
//Linear Transformation
(5)          $\tilde{\mathbf{h}}_i^l = \mathbf{W}^l \tilde{\mathbf{h}}_i^{l-1} + \mathbf{U}^l \mathbf{h}_i^{l-1}$ ;
//Nonlinear activation
(6)          $\mathbf{h}_i^l = f(\tilde{\mathbf{h}}_i^l)$ ;
(7)     }
(8) }
```

**FIGURE 10.42**

GraphSAGE, where the weights are given and the bias vectors are omitted for brevity.

to the one introduced in Section 10.4.3) to learn the impact (or the weights) of the embedding of the neighboring nodes during the aggregation process.

GCNs and its variants (e.g., propagation-based spatial GCN and GraphSAGE) are often trained in the supervised fashion, meaning that they require a set of labeled training tuples (i.e., nodes) to learn the model parameters. There exist many methods (e.g., LINE, DeepWalk, node2vec) to learn node embedding in the unsupervised fashion without the access to the labeled nodes. The central idea of these methods is to find node embedding so that the similarity between nodes in the embedding space resembles that in the input graph.

The GNNs models described here are primarily designed to learn *node* embedding that captures the topological information of the input graph (e.g., the adjacency matrix  $A$ ). *Knowledge graph embedding* (e.g., TransE and its variants) aims to find embedding of both nodes and edges to capture the rich semantic information (e.g., node and edge attributes) of the knowledge graph.

The model parameters of GNNs are often shared among different nodes. This renders the *inductive* property to GNNs, in that once the model parameters are trained, we can use them to find embedding of unseen nodes during the training process. After we obtain the embedding of the nodes, we can further aggregate them (e.g., via average) to obtain the embedding of the entire graph.

GNNs are an active research area, and there are many open research questions. We give three examples to illustrate this. First, most of the existing GNNs work with a relatively small number of layers (e.g.,  $L = 2$  or 3). When the number of layers becomes larger, the learned embedding tends

to be “oversmooth”<sup>15</sup> and thus become less discriminative for classification or clustering tasks. It has largely remained as an open problem on how to make GNN “deeper” by allowing more layers without suffering from the oversmoothing issue.<sup>16</sup> Second, most of the existing GNNs are designed to find embedding of the individual nodes or the entire graphs. A much less studied aspect is how to effectively obtain the embedding of a subgraph (i.e., subgraph embedding), which has numerous applications in question-answering, group recommendation, and so on. Third, graphs from some application domains (e.g., finance, social networks) are often collected from multiple sources. How to develop effective GNNs models for such multisourced graphs (e.g., Multi-GNNs) so that nodes from different graphs are embedded in the same embedding space is another active research area.

---

## 10.6 Summary

- A **neural network** is made up of interconnected units. A **unit** is a mathematical function that (1) takes a linear weighted sum of the input and then (2) passes the sum through an activation function. The **activation function** transforms the input of a unit to its output. The activation function is typically a nonlinear function, such as the sign function, the sigmoid function and ReLU function. A **deep neural network** is a neural network with many layers. It is capable of approximating any nonlinear function and learning a hierarchy of features.
- A **multilayer feed-forward neural network** consists of an input layer, one or more hidden layers, and an output layer. In the matrix form, a multilayer feed-forward neural network can be represented as a chain graph.
- **Backpropagation** is a gradient descent based technique to learn the model parameters, including the weights and bias values, of a neural network. It searches for a set of weights and bias values that can model the data to minimize the loss function between the network’s prediction and the actual target output of data tuples. It consists of two major steps, including forward propagating the net input and output of each unit from the input layer to the output layer, and backward propagating the errors from the output layer to the input layer.
- Two major challenges of training a deep neural network including **optimization** and **generalization**. The optimization challenge refers to obtain a low-cost local minimal of the training loss function in a computationally efficient way. The generalization challenge refers to making sure the trained deep neural networks perform well on the future test tuples.
- **Gradient vanishing** refers to the phenomenon that the gradient of the loss function with respect to the input of certain units quickly approaches zero during the backpropagation process. Gradient vanishing could cause the backpropagation algorithm to be stuck at the high-cost region or take a long time to converge. One common cause of gradient vanishing is the **saturation** of the output of the activation function. Using a more responsive activation function, such as the rectified linear unit (ReLU), could alleviate gradient vanishing.

---

<sup>15</sup> This means that the embedding of nodes may become similar even for those nodes that are distinct and far away from each other.

<sup>16</sup> Another type of GNNs, called Gated Graph Neural Networks, updates the node embedding by a gated recurrent unit (GRU), which can handle more than 20 layers. Nonetheless, how to make GNNs go even deeper remains an area of future research.

- Using **adaptive learning rate** in backpropagation can help accelerate the algorithm convergence or prevent oscillation. Commonly used strategies include setting the learning rate to be in reverse proportion to the epoch number or the magnitude of the accumulated historical gradients.
- **Dropout** is an effective strategy to improve the generalization performance of backpropagation algorithm. At each epoch, it randomly drops some nonoutput units and uses the remaining network to update the model parameters.
- **Pretraining** presets the initial model parameters of a deep neural network in a suitable region. It helps accelerate the backpropagation algorithm convergence and improves the generalization performance. A commonly used pretraining strategy is supervised greedy pretraining, which gradually adds hidden layers and pretrains the parameters of the newly added layers while fixing the parameters of the previously added layers.
- For classification tasks, it is more common to use **cross-entropy** as the loss function to train a deep neural network. Cross-entropy can help alleviate the gradient vanishing problem even when the unit is saturated.
- **Convolutional neural networks (CNNs)** are effective deep learning models for grid-like data, such as images. The key operation in CNNs is **convolution**. Given an input and a kernel, the convolution generates a feature map, where the kernel is shared across different entries of the input. A **convolution layer** consists of a set of kernels, often followed up with a **nonlinear activation** and a **pooling** operation. A CNN model is the neural network with at least one convolutional layer.
- **Recurrent neural networks (RNNs)** are effective deep learning models for sequential data, such as text. There are three main types of units in an RNN model, including input unit, hidden unit, and output unit. The same **input-to-hidden weight matrix**, **hidden-to-hidden weight matrix**, and **hidden-to-output weight matrix** are shared across different time stamps. A major challenge of RNNs is **long-term dependence**. The **long short-term memory (LSTM)** is one of the most successful techniques to address the long-term dependence. The key component of LSTM is the **cell state** in each LSTM cell, which is designed to accumulate the information in the past for a long time and meanwhile use the current information to update the cell state. LSTM achieves these design objectives through three **gates**, including the forget gate, the input gate, and the output gate. **Attention** mechanism is another effective way to address long-term dependence.
- The goal of **graph neural networks (GNNs)** is to use neural networks to map nodes of a graph to low-dimensional vectors called **node embeddings**. An effective type of GNNs is **graph convolutional networks (GCNs)**. Given an input graph with node attributes, a GCN model uses a set of weight matrices, one weight matrix for each layer, to produce the node embeddings. The key operations in each layer of GCNs include **propagation**, **linear transformation**, and **nonlinear activation**.

---

## 10.7 Exercises

- 10.1. The following table consists of training data from an employee database. The data have been generalized. For example, “31 . . . 35” for *age* represents the age range of 31 to 35. For a given row entry, *count* represents the number of data tuples having the values for *department*, *status*, *age*, and *salary* given in that row.

| <i>department</i> | <i>status</i> | <i>age</i> | <i>salary</i> | <i>count</i> |
|-------------------|---------------|------------|---------------|--------------|
| sales             | senior        | 31 ... 35  | 46K ... 50K   | 30           |
| sales             | junior        | 26 ... 30  | 26K ... 30K   | 40           |
| sales             | junior        | 31 ... 35  | 31K ... 35K   | 40           |
| systems           | junior        | 21 ... 25  | 46K ... 50K   | 20           |
| systems           | senior        | 31 ... 35  | 66K ... 70K   | 5            |
| systems           | junior        | 26 ... 30  | 46K ... 50K   | 3            |
| systems           | senior        | 41 ... 45  | 66K ... 70K   | 3            |
| marketing         | senior        | 36 ... 40  | 46K ... 50K   | 10           |
| marketing         | junior        | 31 ... 35  | 41K ... 45K   | 4            |
| secretary         | senior        | 46 ... 50  | 36K ... 40K   | 4            |
| secretary         | junior        | 26 ... 30  | 26K ... 30K   | 6            |

Let *status* be the class-label attribute.

- a. Design a multilayer feed-forward neural network for the given data. Label the nodes in the input and output layers.
  - b. Using the multilayer feed-forward neural network obtained in (a), show the weight values after one iteration of the backpropagation algorithm, given the training instance “(sales, senior, 31 ... 35, 46K ... 50K).” Indicate your initial weight values and biases and the learning rate used.
- 10.2.**
- a. **Derivatives of various activation functions.** Show how the derivatives of activation functions, including sigmoid, tanh, and ELU in Table 10.6, are derived in mathematical details.
  - b. **Backpropagation algorithm.** Consider a multilayer feed-forward neural network as shown in Fig. 10.5 with sigmoid activation and mean-squared loss function  $L$ . Prove (1) Eq. (10.3) for computing the error in the output unit ( $\delta_{10}$ ); (2) Eq. (10.4) for computing the errors in hidden units  $\delta_9$  and  $\delta_6$ ; (hint: consider the chain rule); and (3) Eq. (10.5) for updating weights (hint: consider the derivative of the loss with respect to weights).
  - c. **Relation between different activation functions.** Given the sigmoid function  $\sigma(I) = \frac{1}{1+e^{-I}}$  and hyperbolic tangent function  $\tanh(I) = \frac{e^I - e^{-I}}{e^I + e^{-I}}$ , show in mathematics how  $\tanh(I)$  can be transformed from sigmoid through shifting and re-scaling.
- 10.3. Feed-forward neural networks.** In this exercise, we implement a feed-forward neural network for a binary classification task. The goal is to predict whether the e-mail is spam (labeled as 1) or not (0) according to its attributes (e.g., word frequency, length of uninterrupted sequence of capital letters). The detailed description of the data set can be found at <http://archive.ics.uci.edu/ml/datasets/Spambase>. To complete this exercise, you are required to
- a. construct a multilayer feed-forward neural network for the spam prediction task and evaluate the model from the following perspectives, including prediction accuracy, F1 score, and AUC-ROC curve;
  - b. compare the performance of the models with various activation functions and different depth/width.
- 10.4. Autoencoder.** Autoencoder is a classic type of neural networks for unsupervised learning.
- a. Demonstrate that PCA is a special case of the autoencoder by showing that the loss function of PCA is equivalent to the mean-squared error of the autoencoder with linear activation, where the encoder and decoder share the same parameters.
  - b. Implement an autoencoder with nonlinear activation, and use (1) mean-squared-error and (2) binary cross-entropy as the loss function. Note that when using the binary cross-entropy

loss, each data point should be normalized into  $[0, 1]$ . Compare them with PCA on the MNIST data set, which can be found at <http://yann.lecun.com/exdb/mnist/>, in terms of classification accuracy and the visualization of reconstructed images. For the classification task, first use the autoencoder or PCA to obtain the features of the data and then train a linear SVM or logistic regression model using the features.

- 10.5. Dropout.** (a) Explain why dropout can be used to mitigate the overfitting problem. (b) Suppose we have a simple two-layer neural network  $y = W_2(W_1x + b_1) + b_2$  where  $x \in \mathbb{R}^3$ ,  $W_1 \in \mathbb{R}^{2 \times 3}$ ,  $b_1 \in \mathbb{R}^2$ ,  $W_2 \in \mathbb{R}^2$  and  $b_2 \in \mathbb{R}$ . How many possible dropout networks are there that are not disconnected?
- 10.6. Pretraining on autoencoder.**
- Why pretraining helps the learning or converge?
  - How to pretrain an autoencoder?
- 10.7. Loss functions.** Consider the following two loss functions, including (1) mean-squared error  $\text{Loss}(T, O) = \frac{1}{2}(T - O)^2$ , and (2) cross-entropy  $\text{Loss}(T, O) = -T \log O - (1 - T) \log(1 - O)$  for binary classification. Assume the activation function is sigmoid.
- Show the derivation of the error  $\delta$  for the output unit in backpropagation process and compare the two loss functions (e.g., potential problems they might produce).
  - Now, we wish to generalize the cross-entropy loss to the scenario of multiclass classification. The target output is a one-hot vector of length  $C$  (i.e., the number of total classes), and the index of nonzero element (i.e., 1) represents the class label. The output is also a vector of the same length  $O = [O_1, O_2, \dots, O_C]$ . Show the derivation of the categorical cross-entropy loss and the error  $\delta$  of the output unit. (hint: there are two key steps, including (1) normalizing the output values by scaling between 0 and 1, and (2) deriving the cross entropy loss following the definition for the binary case where the loss can be represented as  $\text{Loss}(T, O) = -\sum_{i=1}^2 T_i \log(O_i)$ .)
- 10.8.** What are the key differences between CNNs and feed-forward neural networks? Why are CNNs widely used on grid-like data?
- 10.9. 2-D convolution.** Given the input and two kernels as shown in Fig. 10.43,
- compute the feature maps by applying kernel  $K_1$ ,  $K_2$  (stride is defined as 1, and zeros are padded around  $I$ ), respectively;
  - re-compute the feature maps if the stride is 2;
  - compute the new feature maps after we apply a downsampling process, max pooling, and average pooling (filter size is  $2 \times 2$ ), respectively, to the obtained feature maps in (a).
- 10.10. Training LSTM.** It is usually hard to train LSTM on long sequences, answer the following questions:
- Why it is hard to train LSTM?
  - How to mitigate the problem?
- 10.11. LSTM and GRU** Compare LSTM with GRU, and answer the following questions:
- What do they have in common?
  - What are the differences between them?
  - What are the pros and cons of them?
- 10.12.** Suppose we apply graph convolutional networks (GCNs) on grid-like graphs (e.g., images) without normalizing adjacency matrix (i.e., removing Steps 2–3 in Fig. 10.38). Explain why it is essentially a 2-D convolution with a special type of filters.

| Input $I$ |   |   |   |   |   |   |   | Kernel |   |   |   |
|-----------|---|---|---|---|---|---|---|--------|---|---|---|
| 0         | 1 | 0 | 0 | 1 | 1 | 0 | 0 |        |   |   |   |
| 0         | 1 | 0 | 1 | 1 | 0 | 1 | 1 |        |   |   |   |
| 1         | 0 | 0 | 1 | 0 | 1 | 0 | 1 | $K_1$  | 0 | 1 | 0 |
| 1         | 0 | 1 | 0 | 1 | 0 | 0 | 1 |        | 0 | 0 | 1 |
| 1         | 1 | 0 | 1 | 0 | 0 | 1 | 0 |        | 1 | 1 | 1 |
| 0         | 0 | 1 | 0 | 0 | 1 | 0 | 1 |        |   |   |   |
| 1         | 1 | 0 | 1 | 0 | 0 | 1 | 0 | $K_2$  | 1 | 1 | 0 |
| 1         | 1 | 0 | 1 | 0 | 0 | 1 | 0 |        | 0 | 1 | 1 |
| 1         | 0 | 1 | 1 | 0 | 0 | 0 | 1 |        | 1 | 0 | 1 |

FIGURE 10.43

Input is a  $8 \times 8$  binary matrix and two kernels are of size  $3 \times 3$ .

- 10.13.** In this exercise, we will derive the graph convolutional networks shown in Sec. 10.5.2 from spectral graph signal processing perspective. The classic convolution on graphs can be computed by  $y = U g_\theta(\Lambda) U^T x$  where  $x \in \mathbb{R}^N$  is the input graph signal (i.e., a vector of scalars that denote feature values of all nodes at a certain feature dimension). Matrices  $U$  and  $\Lambda$  denote the eigen-decomposition of the normalized graph Laplacian:  $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$ .  $g_\theta(\Lambda)$  is the function of eigenvalues and is often used as the filter function.
- If we directly use the classic graph convolution formula where  $g_\theta(\Lambda) = \text{diag}(\theta)$  and  $\theta \in \mathbb{R}^N$  to compute the output  $y$ , what are the potential disadvantages?
  - Suppose we use  $K$ -polynomial filter  $g_\theta(\Lambda) = \sum_{k=0}^K \theta_k \Lambda^k$ . What are the benefits, compared to the above filter  $g_\theta(\Lambda) = \text{diag}(\theta)$ ?
  - By applying the  $K$ th order Chebyshev polynomial approximation on the filter  $g_\theta(\Lambda) = \sum_{k=0}^K \theta_k \Lambda^k$ , the filter function can be written as  $g_{\theta'}(\Lambda) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\Lambda})$  with a rescaled  $\tilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I$ , where  $\lambda_{max}$  denotes the largest eigenvalue of  $L$ , and  $\theta' \in \mathbb{R}^K$  denotes the Chebyshev coefficients. The Chebyshev polynomials can be computed recursively by  $T_k(z) = 2zT_{k-1}(z) - T_{k-2}(z)$  with  $T_0(z) = 1$  and  $T_1(z) = z$ . Now derive the approximate spectral graph convolution formula in terms of the Laplacian matrix. What is the computational complexity?
  - By further simplifying to  $K = 1$ ,  $\lambda_{max} = 2$  and  $\theta = \theta'_0 = -\theta'_1$ , derive the graph convolutional layer for the input graph with  $A$  as the adjacency matrix and  $X \in \mathbb{R}^{N \times C}$  as the input node attribute matrix.
- 10.14.** In this exercise, we aim to implement and learn the graph convolutional networks (GCNs) shown in Fig. 10.38. Specifically, we apply a two-layer GCN for semisupervised node classification on Cora data set. The data can be downloaded at <https://linqs.soe.ucsc.edu/data>. Specifically, first construct the undirected graph based on citation relations (i.e., there exists one undirected edge between two papers if one paper cites another), and node attributes represent the content words. Randomly split 500 nodes for validation and 1000 nodes for testing and the rest for training. Set the hidden dimension to be 64. Use ReLU and softmax functions as the nonlinear activation functions of the first and second layers, respectively. For semisupervised

node classification, use cross-entropy loss function and Adam optimizer with a learning rate 0.01 and a dropout rate 0.5. Implement this model, then evaluate and report the node classification accuracy.

---

## 10.8 Bibliographic notes

The history of deep learning dates back a long time ago. To date, it has experienced three waves since the 1940s. The first wave was in 1940s–1960s. During this wave, the perceptron, a single neuron, was invented in 1958 to perform a simple classification task, that is, to find a linear classification decision boundary. The significance lies in that it demonstrates that we can learn a single neuron to perform (relatively simple) classification tasks. The second wave was in 1980s–1990s. During this wave, the backpropagation algorithm was invented in 1986 to train feed-forward neural networks. This demonstrates the neural network’s capability to obtain a more complicated and intelligent learning system (e.g., a nonlinear classifier) by connecting many simple units (e.g., perceptron) in the form of a network. Neural network learning during this wave is also referred to as *connectionist learning* or *connectionism* due to the connections between units. The third wave began around 2006 and lasted to the date of the writing of this book. The major breakthrough during this wave lies in the invention of a multitude of advanced techniques, often collectively referred to as “deep learning,” to effectively train a large, deep neural network. The trained deep neural network has demonstrated great power, and it has achieved the performance that is close to or even surpasses humans in many applications, such as computer vision, natural language processing, social media analysis, and so on. Generally speaking, there are three key reasons for the success of the third (the current) wave, including (1) the availability of large-scale labeled training data, (2) the dramatic improvement of computational power, and (3) the algorithmic advancement to train a large, deep neural network.

To describe how the neurons in the brain work, McCulloch and Pitts [MP43] utilize electrical circuits to model the operation of a simple neural network. Hebb [Heb49] mentions that neural connections are enhanced every time they are used in the hypothesis of human learning. The perceptron algorithm of binary classification for supervised learning is introduced by Rosenblatt [Ros58]. The first neural network that is applied to real-world application, MADALINE, is developed from Widrow and Hoff [WH60]. Fukushima [Fuk75] develops the first multilayer neural network for unsupervised learning. The backpropagation algorithm is proposed by Rumelhart, Hinton, and Williams [RHW86b], which serves as the key milestone of training a multilayer artificial neural network through backpropagating errors, and Funahashi [Fun89] mathematically proves the capability of multilayer network with sigmoid activation to approximate any continuous mapping (e.g., a binary classifier).

A series of challenges exist in training a deep neural network. From the perspective of activation functions, rectifier is first introduced by Hahnloser and Seung [HS00], Nair and Hinton [NH10] incorporate the rectifier as the activation for a hidden unit in Restricted Boltzmann Machine. Other types of activation functions include, Leaky ReLU from Maas, Hannun, and Ng [MHN], Parametric ReLU from He, Zhang, Ren, and Sun [HZRS15], ELU from Clevert, Unterthiner, and Hochreiter [CUH15], and Swish from Ramachandran, Zoph, and Le [RZL17]. From the optimization perspective, to reduce the burden of computation, the idea of stochastic approximation is incorporated into the gradient-descent-based optimization for deep neural networks (see Robbins and Monro [RM51], Kiefer and Wolfowitz [KW<sup>+</sup>52]), where the key idea is to replace the actual gradient with an estimation. To help acceler-

ate the computation of stochastic gradient descent (SGD), the momentum mechanism is utilized by Qian [Qia99], where it introduces a parameter to preserve the gradient of the previous step. Nesterov accelerated gradient (NAG) by Nesterov [Nes83] improves the gradient calculation by estimating the value of the loss function. In order to set an appropriate learning rate, several methods have been developed. To name a few, AdaGrad from Duchi, Hazan, and Singer [DHS11] adapts the learning rate to the magnitude of the parameter for each dimension; Adadelta from Zeiler [Zei12] aggregates the past gradients in a given window size; RMSProp from Hinton, Srivastava, and Swersky [HSS] further resolves the problem of diminishing learning rate of AdaGrad; Adam and AdaMax are developed by Kingma and Ba [KB14], where Adam combines the advantages of momentum and RMSprop, and AdaMax is a variant of Adam based on the infinity norm; NAdam from Dozat [Doz16] incorporates Nesterov momentum into Adam. Other techniques for optimizing gradient descent include managing the training samples in a meaningful order from curriculum learning by Bengio, Louradour, Collobert, and Weston [BLCW09]; batch normalization for each mini-batch by Ioffe and Szegedy [IS15], which makes it possible to use larger learning rate. Neelakantan et al. [NVL<sup>+</sup>15] add Gaussian noise to every gradient, which improves the training for deeper networks and makes the network more robust. For efficient training, new parameter initialization schemes are proposed by Glorot and Bengio [GB10] and He et al. [HZRS15].

To prevent the network from overfitting, Srivastava et al. [SHK<sup>+</sup>14] propose a method called dropout to randomly deactivate nodes during training. Related works include Wang and Manning [WM13], Ba and Frey [BF13], Pham, Bluche, Kermorvant, and Louradour [PBKL14], Dahl, Sainath, and Hinton [DSH13], Kingma, Salimans, and Welling [KSW15], and Gal and Ghahramani [GG16]. Prechelt [Pre98] introduces a technique called early stopping by leveraging validation. For pretraining used in deep learning, refer to Erhan, Courville, Bengio, and Vincent [ECBV10], Yu and Seltzer [YS11], Saxe, McClelland, and Ganguli [SMG13], Radford, Narasimhan, Salimans, and Sutskever [RNSS18], and Devlin, Chang, Lee, and Toutanova [DCLT18]. Besides supervised learning, for unsupervised tasks, see Barlow [Bar89], Sanger [San89], Baldi [Bal12], Figueiredo and Jain [FJ02], Radford, Meta, and Chintala [RMC15], and Le [Le13]. For semisupervised learning, see Chapelle, Scholkopf, and Zien [CSZ09] and Zhu [Zhu05] for an introduction.

Many variations of backpropagation have been proposed, involving, for example, dynamic adjustment of the network topology (Mézard and Nadal [MN89]; Fahlman and Lebiere [FL90]; Le Cun, Denker, and Solla [LDS89]; and Harp, Samad, and Guha [HSG89]); and dynamic adjustment of the learning rate and momentum parameters (Jacobs [Jac88]). Other variations are discussed in Chauvin and Rumelhart [CR95]. Books on neural networks include Rumelhart and McClelland [RM86]; Hecht-Nielsen [HN90]; Hertz, Krogh, and Palmer [HKP91]; Chauvin and Rumelhart [CR95]; Bishop [Bis95]; Ripley [Rip96]; and Haykin [Hay99]. Many books on machine learning, such as Mitchell [Mit97] and Russell and Norvig [RN95], also contain good explanations of the backpropagation algorithm.

Yang, Fu, Sidiropoulos, and Hong [YFSH17] propose to integrate autoencoder and  $K$ -means to simultaneously find clusters and embedding. Xie, Girshick, and Farhadi [XGF16] introduce KL-divergence in deep clustering. Yang, Fu, and Sidiropoulos [YFS16] propose joint nonnegative matrix factorization and  $K$ -means for latent clustering. For the application of deep clustering in image data, see Caron, Bojanowski, Joulin, and Douze [CBJD18], and Yang, Parikh, and Batra [YPB16].

Convolution, rooted in mathematics, has been widely used in various areas like signal processing. Convolutional neural networks (CNNs), inspired by the visual cortex [HW62], are one of the most successful architectures in deep learning. Waibel et al. [WHH<sup>+</sup>89] introduces time-delayed neural net-



work (TDNN) for speech recognition, which is the first CNN model. LeCun et al. [LBD<sup>+</sup>89] apply 2-D convolution operation for recognizing handwritten zip code. LeNet-5 by LeCun, Bottou, Bengio, and Haffner [LBBH98] represents a milestone of CNNs. Krizhevsky, Sutskever, and Hinton [KSH12] introduce the first Deep CNN (DCNN) architecture—AlexNet, which has achieved superior performance on the ImageNet data set [DDS<sup>+</sup>09]. VGGNets [SZ15] and GoogLeNets [SLJ<sup>+</sup>15] demonstrate that deeper architectures with small filters in convolutional layers could achieve better results. DCNNs are susceptible to the problems of vanishing gradients and exploding gradients. To address this, He, Zhang, Ren, and Sun [HZRS16] introduce deep residual learning and propose residual nets (ResNet). Huang, Liu, Van Der Maaten, and Weinberger [HLVDMW17] exploit the idea of skip-connections in ResNet and introduce dense convolutional network (DenseNet) with dense connectivity.

These CNN models have not only achieved superior performance in image classification tasks, but also been used as basic building blocks for other applications. DeepFace [TYRW14] and FaceNet [SKP15] adopt variants of AlexNet [KSH12] and GoogLeNets [SLJ<sup>+</sup>15] for face recognition respectively. R-CNN [GDDM14], Fast R-CNN [Gir15], Faster R-CNN [RHGS15], and Mask R-CNN [HGDG17] are some representative works in the field of object detection and segmentation. DeepPose [TS14] and two-stream CNNs [SZ14] adopt CNNs for human pose estimation and action recognition respectively. DCGAN [RMC16] introduces a deconvolutional neural network for generating images. Shen, Wu, and Suk [SWS17] review CNNs-based methods for medical image analysis. In addition to computer vision tasks, CNNs have also been used in speech recognition [AHMJ12], natural language processing [Kim14], and time-series analysis [WYO17].

Recurrent neural networks (RNNs) [RHW86a] are another classic neural network architecture, for modeling sequential data. Inspired by the memory mechanism in human brains, Little [Lit74] and Hopfield [Hop82] introduce (Little-)Hopfield Network, one of the earliest forms of RNNs with binary thresholds. Elman Network [Elm90] generalizes the Hopfield Network and is known as *simple recurrent network*. Schuster and Paliwal [SP97] introduce bi-directional RNNs (BRNNs) to allow the current state to gather the input information from not only the past but also the future. Hochreiter and Schmidhuber [HS97] propose long short-term memory networks (LSTM), one of the most popular RNNs, to solve the problem of vanishing gradients when training simple RNNs. Cho et al. [CVMG<sup>+</sup>14] simplify LSTM by introducing the gated recurrent unit (GRU).

RNNs are widely used for modeling sequential data such as natural languages and time series. In the field of natural language processing, RNN models have become cornerstones for most of the tasks, such as machine translation [SVL14,CVMG<sup>+</sup>14,BCB15], dialog system [LGB<sup>+</sup>16,WVM<sup>+</sup>17], named entity recognition [LBS<sup>+</sup>16], parsing [DBL<sup>+</sup>15], and speech recognition [GS05,GMH13]. RNNs have also been used for time series classification [FFW<sup>+</sup>19,LKEW16], missing value recovery [CPC<sup>+</sup>18], future value prediction [CMA94], and unsupervised feature learning [LKL14].

In addition to the independent usage of CNN and RNN models, these models are also used jointly for multimodal deep learning [BAM18], such as image captioning [VTBE15,XBK<sup>+</sup>15], image synthesis [RAY<sup>+</sup>16,OOS17], visual question answering [AAL<sup>+</sup>15,LYBP16], medical report generation [JXX18], and video analysis [SMS15,YHNHV<sup>+</sup>15].

Graph neural networks (GNNs) refer to the neural network structures on graph-structured data. One pioneering GNN architecture [SGT<sup>+</sup>08] is extended from recurrent models and iteratively updates each node's state by the propagation function of node neighborhood, node, and edge labels until the state equilibrium is reached. Gated GNNs proposed by Li, Zemel, Brockschmidt, and Tarlow [LTBZ15] use gated recurrent units as the recurrent propagation function. Seo, Defferrard, Vandergheynst, and

Bresson propose to combine graph CNN [DBV16] with recurrent units to capture the spatial and temporal patterns of graphs [SDVB18], and VGRNN [HHN<sup>+</sup>19] integrates variational graph autoencoder [KW16b] to learn latent representations of dynamic graphs. In addition, SSE [DKD<sup>+</sup>18] applies the stochastic updates of nodes' states. All these models are graph recurrent neural networks.

Graph CNNs are inspired by the classic CNNs and graph signal processing [SNF<sup>+</sup>13]. Bruna, Zaremba, Szlam, and LeCun propose to use spectral-based graph convolutions to learn the latent graph representations [BZSL13]. Defferrard, Bresson, and Van der Gheynst follow up to approximate the spectral graph convolution by Chebyshev polynomials [DBV16]. GCNs [KW16a] is a well-established model that fills the gap between spectral-based and spatial-based graph convolutional neural networks. Hamilton, Ying, and Leskovec propose an inductive model graphSage [HYL17] with multiple aggregation operators. FastGCN proposes to sample a fixed number of nodes at each convolutional layer [CMX18]. Graph attention networks (GATs) [VCC<sup>+</sup>17] apply an attention mechanism on the neighborhood aggregation. Graph isomorphism networks (GINs) [XHLJ18] use multilayer perceptrons as the aggregation function and are provable that it is as powerful as Weisfeiler Lehman graph isomorphism test by Shervashidze et al. [SSVL<sup>+</sup>11]. Other spatial-based models include [NAK16,GSR<sup>+</sup>17,MBM<sup>+</sup>17]. All these GNNs are for homogeneous networks. For heterogeneous networks, Zhang et al. propose the heterogeneous graph neural networks (HetGNNs) [ZSH<sup>+</sup>19] by designing the neighborhood aggregations that incorporate different node types. Graph CNNs are widely applied to the applications where networks are of multiple node types, such as text classification [YML19], and recommendation [WHW<sup>+</sup>19,YHC<sup>+</sup>18]. In addition to different graph convolutional layers, several graph pooling operators are designed to adaptively learn the hierarchical graph representations. Ying et al. propose DiffPool [YYM<sup>+</sup>18], which learns soft clustering assignment by GNNs and coarsens the input graphs at different resolutions. Gao and Ji propose Graph U-Nets that uses top-k node selection for graph pooling [GJ19], and SAGPool [LLK19] proposed by Lee, Lee, and Kang leverages GCNs [KW16a] to select important nodes for pooling.

For unsupervised learning, network embedding techniques, which sometimes are considered as shallow representation learning, have been extensively studied. Tang et al. propose LINE [TQW<sup>+</sup>15] that preserves first-order and second-order structural proximities and minimizes KL-divergence-based loss function. DeepWalk [PARS14] follows the idea of word2vec [MSC<sup>+</sup>13] and generate context nodes by truncated random walks, which are used in the Skip-gram type of loss function. Node2vec [GL16] combines breadth-first sampling (BFS) and depth-first sampling (DFS) for context node sampling. In addition to network embedding, Kipf and Welling propose variational graph autoencoder (VGAE) [KW16b], which uses GCNs [KW16a] as the encoder and reconstructs input graph in decoder. GraphSage [HYL17] can be also generalized to the unsupervised manner by using Skip-gram loss function. Adapted from deep infomax [HFLM<sup>+</sup>18], deep graph infomax (DGI) [VFH<sup>+</sup>18] maximizes mutual information between patch representations and the graph summaries.

In the age of big data, networks are often multisourced. Learning node representations of multisourced networks often suffer from the space disparity issue of the embedding vectors in different networks. To address this issue, Du and Tong propose MrMine that learns node embeddings of multiple networks in the same space based on the multiresolution characteristics of networks [DT19]. Origin [ZTX<sup>+</sup>19] is proposed to leverage nonrigid point-set registration to address the embedding space disparity issue. Another angle to tackle this issue is to learn graph-to-graph translation functions by GNNs. Jin et al. propose to leverage adversarial training to align the distributions of generated graphs [JYBJ19], and Guo et al. propose the graph-to-graph translation model to study network coevolution [GZN<sup>+</sup>19].

NetTrans [ZTX<sup>+</sup>20] proposed by Zhang et al. learns how a source network can be transformed into a target network and how nodes are associated across networks by an encoder-decoder architecture.

Recent years have witnessed an emerging research trend of generative models in deep learning. Generative adversarial networks (GANs) incorporate the idea of adversarial learning to generate new data samples and are proposed by Goodfellow et al. [GPAM<sup>+</sup>14]. GANs have been widely studied in a variety of applications, ranging from high-resolution image generation [RMC15,LTH<sup>+</sup>17,BSM17], image-to-image translation [ZPIE17,HZLH17], text/dialogue generation [YZWY17,GLC<sup>+</sup>18,LMS<sup>+</sup>17], text-to-image synthesis [ZXL<sup>+</sup>17], and graph representation learning [WWW<sup>+</sup>18], to medical record generation [CBM<sup>+</sup>17]. Some variants of generative models include variational autoencoder (VAE) from Kingma and Welling [KW13]. See Doersch [Doe16] for a detailed introduction.

Deep learning is also applied in other data mining problems. For example, in reinforcement learning, deep neural networks are utilized to estimate the state-value function (i.e., Q function) (Mnih et al. [MKS<sup>+</sup>15,MKS<sup>+</sup>13,MBM<sup>+</sup>16] and Lillicrap et al. [LHP<sup>+</sup>15]). Deep neural networks are found to be vulnerable to adversarial activities in the form of very subtle perturbation to the input data. Such adversarial attacks result in degraded performance of the neural networks, including misclassification of image data from Szegedy et al. [SZS<sup>+</sup>13], error in image reconstruction resulted from the generative model from Tabacof, Tavares, and Valle [TTV16], misclassification of sequences of words for LSTM from Papernot, McDaniel, Swami, and Harang [PMSH16], reducing reward in deep reinforcement learning from Lin et al. [LHL<sup>+</sup>17], and reducing the performance of GNNs in tasks such as node classification (Dai et al. [DLT<sup>+</sup>18] and Zügner, Akbarnejad, and Günnemann [ZAG18]). To further understand and interpret the results of machine learning models, research has been conducted toward this direction, see Du, Liu, and Hu [DLH19] for an introduction. In a decentralized scenario where the data samples are stored in multiple local devices, it is possible to collaboratively learn a shared machine learning model, and this refers to as federated learning [KMY<sup>+</sup>16,SCST17]. Automated machine learning (AutoML) studies the problem of applying effective feature processing and machine learning algorithms to a new data set (Feurer et al. [FKE<sup>+</sup>15] and Hutter, Kotthoff, and Vanschoren [HKV19]).

# Outlier detection

# 11

**Imagine that you are** a transaction auditor in a credit card company. To protect your customers from credit card fraud, you pay special attention to card usages that are rather different from typical cases. For example, if a purchase amount is much bigger than usual for a card owner, and if the purchase occurs far from the owner's resident city, then the purchase is suspicious. You want to detect such transactions as soon as they occur and contact the card owner for verification. This is common practice in many credit card companies. *What data mining techniques can help detect suspicious transactions?*

Most credit card transactions are normal. However, if a credit card is stolen, its transaction pattern usually changes dramatically—the locations of purchases and the items purchased are often very different from those of the authentic card owner and other customers. An essential idea behind credit card fraud detection is to identify those transactions that are very different from the norm.

*Outlier detection* (also known as *anomaly detection*) is the process of finding data objects with behaviors that are very different from expectation. Such objects are called **outliers** or **anomalies**. Outlier or anomaly detection is important in many applications in addition to fraud detection such as medical care, public safety and security, industry damage detection, image processing, sensor and video network surveillance, national security, and intrusion detection.

Outlier detection and clustering analysis are two highly related tasks. Clustering finds the majority patterns in a data set and organizes the data accordingly, whereas outlier detection tries to capture those exceptional cases that deviate substantially from the majority patterns. Outlier detection and clustering analysis serve different purposes. In terms of methodologies, outlier detection might also use supervision during the detection process, whereas clustering analysis is typically unsupervised in nature.

In this chapter, we study outlier detection techniques. Section 11.1 introduces the basic concepts, including different types of outliers and an overview of outlier detection methods. In the rest of the chapter, you will learn about outlier detection methods in detail. These approaches, organized here by category, are statistical (Section 11.2), proximity-based (Section 11.3), reconstruction-based (Section 11.4), and clustering-based vs. classification-based approaches (Section 11.5). In addition, you will learn about mining contextual and collective outliers (Section 11.6) and outlier detection in high-dimensional data (Section 11.7).

---

## 11.1 Basic concepts

Let us first define what outliers are, categorize the different types of outliers, and then discuss the challenges in outlier detection at a general level, followed by an overview of outlier detection methods.

### 11.1.1 What are outliers?

Assume that a given statistical process is used to generate a set of data objects. An **outlier** is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism. For ease of presentation within this chapter, we may refer to data objects that are not outliers as “normal” or expected data. Similarly, we may refer to outliers as “abnormal” data.

**Example 11.1. Outliers.** In Fig. 11.1, most objects roughly follow a Gaussian distribution. However, the objects in region  $R$  are significantly different. It is unlikely that they follow the same distribution as the other objects in the data set. Thus, the objects in  $R$  are outliers in the data set.  $\square$

Outliers are different from noisy data. As mentioned in Chapter 2, noise is a random error or variance in a measured variable. In general, noise is not interesting in data analysis, including outlier detection. For example, in credit card fraud detection, a customer’s purchase behavior can be modeled as a random variable. A customer may generate some “noisy transactions” that may seem like “random errors” or “variance,” such as by buying a bigger lunch one day, or having one more cup of coffee than usual. Such transactions should not be treated as outliers; otherwise, the credit card company would incur heavy costs from verifying that many transactions. The company may also lose customers by bothering them with multiple false alarms. As in many other data analysis and data mining tasks, noise should be removed before outlier detection.

Outliers are interesting because they are suspected of not being generated by the same mechanism as the rest of the data. Therefore in outlier detection, it is important to justify *why* the detected outliers are generated by some other mechanisms. This is often achieved by making various assumptions on the rest of the data and showing that the detected outliers violate those assumptions significantly.

Outlier detection is also related to *novelty detection* in evolving data sets. For example, by monitoring a social media web site where new content is incoming, novelty detection may identify new topics and trends in a timely manner. Novel topics may initially appear as outliers. To this extent, outlier detection and novelty detection share some similarity in modeling and detection methods. However, a critical difference between the two is that in novelty detection, once new topics are confirmed, they are usually incorporated into the model of normal behavior so that follow-up instances are not treated as outliers anymore.

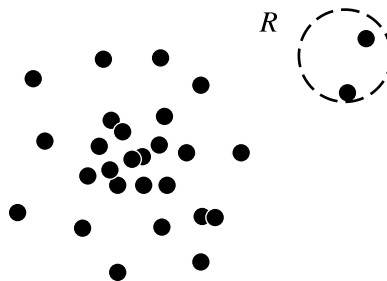


FIGURE 11.1

The objects in region  $R$  are outliers.

### 11.1.2 Types of outliers

In general, outliers can be classified into three categories, namely global outliers, contextual (or conditional) outliers, and collective outliers. Let's examine each of these categories.

#### **Global outliers**

In a given data set, a data object is a **global outlier** if it deviates significantly from the rest of the data set. Global outliers are sometimes called *point anomalies* and are the simplest type of outliers. Most outlier detection methods are aimed at finding global outliers.

**Example 11.2. Global outliers.** Consider the points in Fig. 11.1 again. The points in region  $R$  significantly deviate from the rest of the data set and hence are examples of global outliers.  $\square$

To detect global outliers, a critical issue is to find an appropriate measurement of deviation with respect to the application in question. Various measurements exist, and, based on these, outlier detection methods are partitioned into different categories. We will come to this issue in detail later.

Global outlier detection is important in many applications. Consider intrusion detection in computer networks, for example. If the communication behavior of a computer is very different from the normal patterns (e.g., a large number of packages is broadcast in a short time), this behavior may be considered as a global outlier, and the corresponding computer is a suspected victim of hacking. As another example, in trading transaction auditing systems, transactions that do not follow the regulations are considered as global outliers and should be held for further examination.

#### **Contextual outliers**

*“The temperature today is 28°C. Is it exceptional (i.e., an outlier)?”* It depends, for example, on the time and location! If it is in winter in Toronto, yes, it is an outlier. If it is a summer day in Toronto, then it is normal. Unlike global outlier detection, in this case, whether or not today's temperature value is an outlier depends on the context—the date, the location, and possibly some other factors.

In a given data set, a data object is a **contextual outlier** if it deviates significantly with respect to a specific context of the object. Contextual outliers are also known as *conditional outliers* because they are conditional on the selected context. Therefore, in contextual outlier detection, the context has to be specified as part of the problem definition. Generally, in contextual outlier detection, the attributes of the data objects in question are divided into two groups:

- **Contextual attributes:** The contextual attributes of a data object define the object's context. In the temperature example, the contextual attributes may be date and location.
- **Behavioral attributes:** These define the object's characteristics and are used to evaluate whether the object is an outlier in the context to which it belongs. In the temperature example, the behavioral attributes may be the temperature, humidity, and pressure.

Unlike global outlier detection, in contextual outlier detection, whether a data object is an outlier depends on not only the behavioral attributes but also the contextual attributes. A configuration of behavioral attribute values may be considered an outlier in one context (e.g., 28°C is an outlier for a Toronto winter) but not an outlier in another context (e.g., 28°C is not an outlier for a Toronto summer).

Contextual outliers are a generalization of local outliers, a notion introduced in density-based outlier analysis approaches. An object in a data set is a **local outlier** if its density significantly deviates from the local area in which it occurs. We will discuss local outlier analysis in greater detail in Section 11.3.2.

Global outlier detection can be regarded as a special case of contextual outlier detection where the set of contextual attributes is empty. In other words, global outlier detection uses the whole data set as the context. Contextual outlier analysis provides flexibility to users in that one can examine outliers in different contexts, which can be highly desirable in many applications.

**Example 11.3. Contextual outliers.** In credit card fraud detection, in addition to global outliers, an analyst may consider outliers in different contexts. Consider customers who use more than 90% of their credit limit. If one such customer is viewed as belonging to a group of customers with low credit limits, then such behavior may not be considered an outlier. However, similar behavior of customers from a high-income group may be considered outliers if their balance often exceeds their credit limit. Such outliers may lead to business opportunities—raising credit limits for such customers can bring in new revenue. □

The quality of contextual outlier detection in an application depends on the meaningfulness of the contextual attributes, in addition to the measurement of the deviation of an object to the majority in the space of behavioral attributes. More often than not, the contextual attributes should be determined by domain experts, which can be regarded as part of the input background knowledge. In many applications, neither obtaining sufficient information to determine contextual attributes nor collecting high-quality contextual attribute data is easy.

*“How can we formulate meaningful contexts in contextual outlier detection?”* A straightforward method simply uses group-bys of the contextual attributes as contexts. This may not be effective, however, because many group-bys may have insufficient data or noise. A more general method uses the proximity of data objects in the space of contextual attributes. We discuss this approach in detail in Section 11.3.

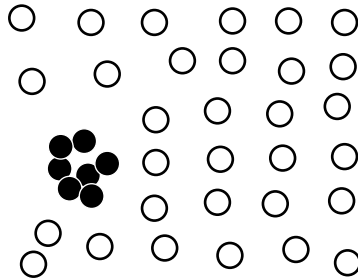
### **Collective outliers**

Suppose you are a supply-chain manager of an electronics store. You handle thousands of orders and shipments every day. If the shipment of an order is delayed, it may not be considered an outlier because, statistically, delays occur from time to time. However, you have to pay attention if 100 orders are delayed on a single day. Those 100 orders as a whole form an outlier, although each of them may not be regarded as an outlier if considered individually. You may have to take a close look at those orders collectively to understand the shipment problem.

Given a data set, a subset of data objects forms a **collective outlier** if the objects as a whole deviate significantly from the entire data set. Importantly, the individual data objects may not be outliers.

**Example 11.4. Collective outliers.** In Fig. 11.2, the black objects as a whole form a collective outlier because the density of those objects is much higher than the rest in the data set. However, every black object individually is not an outlier with respect to the whole data set. □

Collective outlier detection has many important applications. For example, in intrusion detection, a denial-of-service package from one computer to another is considered normal and not an outlier at all. However, if several computers keep sending denial-of-service packages to each other, they as a whole should be considered as a collective outlier. The computers involved may be suspected of being compromised by an attack. As another example, a stock transaction between two parties is considered normal. However, a large set of transactions of the same stock among a small party in a short period are collective outliers because they may be evidence of some people manipulating the market.



**FIGURE 11.2**

The black objects form a collective outlier.

Unlike global or contextual outlier detection, in collective outlier detection we have to consider not only the behavior of individual objects, but also that of groups of objects. Therefore to detect collective outliers, we need background knowledge of the relationship among data objects such as distance or similarity measurements between objects.

In summary, a data set can have multiple types of outliers. Moreover, an object may belong to more than one type of outlier. In business, different outliers may be used in various applications or for different purposes. Global outlier detection is the simplest. Contextual outlier detection requires background information to determine contextual attributes and contexts. Collective outlier detection requires background information to model the relationship among objects to find groups of outliers.

### 11.1.3 Challenges of outlier detection

Outlier detection is useful in many applications yet faces many challenges such as the following:

- **Modeling normal objects and outliers effectively.** Outlier detection quality highly depends on the modeling of normal (nonoutlier) objects and outliers. Often, building a comprehensive model for data normality is very challenging, if not impossible. This is partly because it is hard to enumerate all possible normal behaviors in an application. The border between data normality and abnormality (outliers) is often not clear-cut. Instead, there can be a wide range of gray area. Consequently, while some outlier detection methods assign to each object in the input data set a label of either “normal” or “outlier,” other methods assign to each object a score measuring the “outlier-ness.”<sup>1</sup>
- **Application-specific outlier detection.** Technically, choosing the similarity or distance measure and the relationship model to describe data objects is critical in outlier detection. Unfortunately, such choices are often application-dependent. Different applications may have very different requirements. For example, in clinic data analysis, a small deviation may be important enough to justify an outlier. In contrast, in marketing analysis, objects are often subjected to larger fluctuations, and consequently a substantially larger deviation is needed to justify an outlier. Outlier detection’s

<sup>1</sup> In some literature, it is also referred to as the “outlyingness” score.



high dependency on the application type makes it impossible to develop a universally applicable outlier detection method. Instead, individual outlier detection methods that are dedicated to specific applications must be developed.

- **Handling noise in outlier detection.** As mentioned earlier, outliers are different from noise. It is also well known that the quality of real data sets tends to be poor. Noise often unavoidably exists in data collected in many applications. Noise may be present as deviations in attribute values or even as missing values. Low data quality and the presence of noise bring a huge challenge to outlier detection. They can distort the data, blurring the distinction between normal objects and outliers. Moreover, noise and missing data may “hide” outliers and reduce the effectiveness of outlier detection—an outlier may appear “disguised” as a noisy point, and an outlier detection method may mistakenly identify a noisy point as an outlier.
- **Interpretability.** In some application scenarios, a user may want to not only detect outliers but also understand why the detected objects are outliers. To meet the understandability requirement, an outlier detection method has to provide some justification of the detection. For example, a statistical method can be used to justify the degree to which an object may be an outlier based on the likelihood that the object was generated by the same mechanism that generated the majority of the data. The smaller the likelihood, the more unlikely the object was generated by the same mechanism, and the more likely the object is an outlier.

### 11.1.4 An overview of outlier detection methods

There are many outlier detection methods in the literature and in practice. Here, we present two orthogonal ways to categorize outlier detection methods. First, we categorize outlier detection methods according to whether the sample of data for analysis is given with domain expert–provided labels that can be used to build an outlier detection model. Second, we divide methods into groups according to their assumptions regarding normal objects vs. outliers.

#### ***Supervised, semisupervised, and unsupervised methods***

If expert-labeled examples of normal or outlier objects can be obtained, they can be used to build outlier detection models. The methods used can be divided into supervised methods, semisupervised methods, and unsupervised methods.

*Supervised methods.* Supervised methods model data normality and abnormality. Domain experts examine and label a sample of the underlying data. Outlier detection can then be modeled as a classification problem (Chapters 6 and 7). The task is to learn a classifier that can recognize outliers. The sample is used for training and testing. In some applications, the experts may label just the normal objects, and any other objects not matching the model of normal objects are reported as outliers. Other methods model the outliers and treat objects not matching the model of outliers as normal.

Although many classification methods can be applied, challenges to supervised outlier detection include the following:

- The two classes (i.e., normal objects vs. outliers) are imbalanced. That is, the population of outliers is typically much smaller than that of normal objects. Therefore methods for handling imbalanced classes (Section 6.7.5) may be used, such as oversampling (i.e., replicating) outliers to increase their distribution in the training set used to construct the classifier. Due to the small population of

outliers in data, the sample data examined by domain experts and used in training may not even sufficiently represent the outlier distribution. The lack of outlier samples can limit the capability of the constructed classifiers. To tackle these problems, some methods “make up” artificial outliers.

- In many outlier detection applications, catching as many outliers as possible (i.e., the sensitivity or recall of outlier detection) is far more important than not mislabeling normal objects as outliers. Consequently, when a classification method is used for supervised outlier detection, it has to be interpreted appropriately so as to consider the application interest on recall.

In summary, supervised methods of outlier detection must be careful in how they train and how they interpret classification rates due to the fact that outliers are rare in comparison to the other data samples.

*Unsupervised methods.* In some application scenarios, objects labeled as “normal” or “outlier” are not available. Thus an unsupervised learning method has to be used.

Unsupervised outlier detection methods make an implicit assumption: The normal objects are somewhat “clustered.” In other words, an unsupervised outlier detection method expects that normal objects follow a pattern far more frequently than outliers. Normal objects do not have to fall into one group sharing high similarity. Instead, they can form multiple groups, where each group has distinct features. However, an outlier is typically expected to occur far away in feature space from any of those groups of normal objects.

This assumption may not be true all the time. For example, in Fig. 11.2, the normal objects do not share any strong patterns. Instead, they are uniformly distributed. The collective outliers, however, share high similarity in a small area. Unsupervised methods might not be able to detect such outliers effectively. In some applications, normal objects are diversely distributed, and many such objects do not follow strong patterns. For instance, in some intrusion detection and computer virus detection problems, normal activities are very diverse and many do not fall into high-quality clusters. In such scenarios, unsupervised methods may have a high false positive rate—they may mislabel many normal objects as outliers (intrusions or viruses in these applications), and let many actual outliers go undetected. Due to the high similarity between intrusions and viruses (i.e., they have to attack key resources in the target systems), modeling outliers using supervised methods may be far more effective.

Many clustering methods can be adapted to act as unsupervised outlier detection methods. The central idea is to find clusters first, and then the data objects not belonging to any cluster could be flagged as outliers. However, such methods suffer from two issues. First, a data object not belonging to any cluster may be noise instead of an outlier. Second, it is often costly to find clusters first and then find outliers. It is usually assumed that there are far fewer outliers than normal objects. Having to process a large population of nontarget data entries (i.e., the normal objects) before one can touch the real meat (i.e., the outliers) can be unappealing. More recent unsupervised outlier detection methods develop various smart ideas to tackle outliers directly without explicitly and completely finding clusters. You will learn more about these techniques in Sections 11.3 and 11.5.1 on proximity-based and clustering-based methods, respectively.

*Semisupervised methods.* In many applications, although obtaining some labeled examples is feasible, the number of such labeled examples is often small. We may encounter cases where only a small set of the normal and outlier objects are labeled, but most of the data are unlabeled. Semisupervised outlier detection methods were developed to tackle such scenarios.

Semisupervised outlier detection methods can be regarded as applications of semisupervised learning methods (Section 7.5.1). For example, when some labeled normal objects are available, we can use

them, together with unlabeled objects that are close by, to train a model for normal objects. The model of normal objects then can be used to detect outliers—those objects not fitting the model of normal objects are classified as outliers.

If only some labeled outliers are available, semisupervised outlier detection is trickier. A small number of labeled outliers are unlikely to represent all the possible outliers. Therefore building a model for outliers based on only a few labeled outliers is unlikely to be effective. To improve the quality of outlier detection, we can get help from models for normal objects learned from unsupervised methods.

For additional information on semisupervised methods, interested readers are referred to the bibliographic notes at the end of this chapter.

### ***Statistical methods, proximity-based methods, and reconstruction-based methods***

As discussed earlier, outlier detection methods make assumptions about outliers vs. the rest of the data. According to the assumptions made, we can categorize outlier detection methods into three types: statistical methods, proximity-based methods, and reconstruction-based methods.

*Statistical methods.* **Statistical methods** (also known as **model-based methods**) make assumptions of data normality. They assume that normal data objects are generated by a statistical (stochastic) model, and that data not following the model are outliers.

**Example 11.5. Detecting outliers using a statistical (Gaussian) model.** In Fig. 11.1, the data points except for those in region  $R$  fit a Gaussian distribution  $g_D$ , where for a location  $\mathbf{x}$  in the data space,  $g_D(\mathbf{x})$  gives the probability density at  $\mathbf{x}$ . Thus, the Gaussian distribution  $g_D$  can be used to model the normal data, that is, most of the data points in the data set. For each object  $\mathbf{y}$  in region  $R$ , we can estimate  $g_D(\mathbf{y})$ , the probability that this point fits the Gaussian distribution. Because  $g_D(\mathbf{y})$  is very low,  $\mathbf{y}$  is unlikely generated by the Gaussian model and thus is an outlier.  $\square$

The effectiveness of statistical methods highly depends on whether the assumptions made for the statistical model hold true for the given data. There are many kinds of statistical models. For example, the statistic models used in the methods may be parametric or nonparametric. Statistical methods for outlier detection are discussed in detail in Section 11.2.

*Proximity-based methods.* **Proximity-based methods** assume that an object is an outlier if the nearest neighbors of the object are far away in feature space, that is, the proximity of the object to its neighbors significantly deviates from the proximity of most of the other objects to their neighbors in the same data set.

**Example 11.6. Detecting outliers using proximity.** Consider the objects in Fig. 11.1 again. If we model the proximity of an object using its three nearest neighbors, then the objects in region  $R$  are substantially different from other objects in the data set. For the two objects in  $R$ , their second and third nearest neighbors are dramatically more remote than those of any other objects. Therefore we can label the objects in  $R$  as outliers based on proximity.  $\square$

The effectiveness of proximity-based methods relies heavily on the proximity (or distance) measure used. In some applications, such measures cannot be easily obtained. Moreover, proximity-based methods often have difficulty in detecting a group of outliers if the outliers are close to one another. There are two major types of proximity-based outlier detection, namely *distance-based* and *density-based* outlier detection. Proximity-based outlier detection is discussed in Section 11.3.

*Reconstruction-based methods.* Reconstruction-based outlier detection approaches are built upon the following idea. Since the normal data samples often share certain similarities, they can often be represented in a more succinct way compared with their original representation (e.g., an attribute vector for each data sample). With the succinct representation, we can well *reconstruct* the original representation of the normal samples. On the other hand, for samples that cannot be well reconstructed by such alternative, succinct representation, we flag them as outliers.

There are two major types of reconstruction-based outlier detection approaches, namely matrix-factorization based methods for numerical data; and pattern-based compression methods for categorical data. Reconstruction-based outlier detection is discussed in detail in Section 11.4.

The rest of this chapter discusses approaches to outlier detection.

---

## 11.2 Statistical approaches

As with statistical methods for clustering, statistical methods for outlier detection make assumptions about data normality. They assume that the normal objects in a data set are generated by a stochastic process (e.g., a generative model). Consequently, normal objects occur in the regions of high probability for the stochastic model, and objects in the regions of low probability are outliers.

The general idea behind statistical methods for outlier detection is to learn a generative model fitting the given data set and then identify those objects in low-probability regions of the model as outliers. However, there are many different ways to learn generative models. In general, statistical methods for outlier detection can be divided into two major categories: *parametric methods* and *nonparametric methods*, according to how the models are specified and learned.

A **parametric method** assumes that the normal data objects are generated by a parametric distribution with a finite number of parameters  $\Theta$ . The *probability density function* of the parametric distribution  $f(x, \Theta)$  gives the probability that object  $x$  is generated by the distribution. The smaller this value, the more likely  $x$  is an outlier.

A **nonparametric method** does not assume an a priori statistical model with a finite number of parameters. Instead, a nonparametric method tries to determine the model from the input data. Note that most nonparametric methods do not assume that the model is completely parameter-free. (Such an assumption would make learning the model from data almost a mission impossible.) Instead, nonparametric methods often take the position that the number and nature of the parameters are flexible and not fixed in advance. Examples of nonparametric methods include histogram and kernel density estimation.

### 11.2.1 Parametric methods

In this subsection, we introduce several simple yet practical parametric methods for outlier detection. We first discuss methods for univariate data based on normal distribution. We then discuss how to handle multivariate data using multiple parametric distributions.

#### ***Detection of univariate outliers based on normal distribution***

Data involving only one attribute or variable are called *univariate data*. For simplicity, we often choose to assume that data are generated from a normal distribution. We can then learn the parameters of the normal (i.e., Gaussian) distribution from the input data and identify the points with low probability as outliers.

Let's start with univariate data. We will try to detect outliers by assuming the data follow a normal distribution.

**Example 11.7. Univariate outlier detection using maximum likelihood method.** Suppose a city's average temperature values in July in the last 10 years are, in value-ascending order, 24.0°C, 28.9°C, 28.9°C, 29.0°C, 29.1°C, 29.1°C, 29.2°C, 29.2°C, 29.3°C, and 29.4°C. Let's assume that the average temperature follows a normal distribution, which is determined by two parameters: the mean,  $\mu$ , and the standard deviation,  $\sigma$ .

We can use the *maximum likelihood method* to estimate the parameters  $\mu$  and  $\sigma$ . That is, we maximize the *log-likelihood function*

$$\ln \mathcal{L}(\mu, \sigma^2) = \sum_{i=1}^n \ln f(x_i | (\mu, \sigma^2)) = -\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2, \quad (11.1)$$

where  $n$  is the total number of samples, which is 10 in this example.

Taking derivatives with respect to  $\mu$  and  $\sigma^2$  and solving the resulting system of first-order conditions leads to the following *maximum likelihood estimates*:

$$\hat{\mu} = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (11.2)$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (11.3)$$

In this example, we have

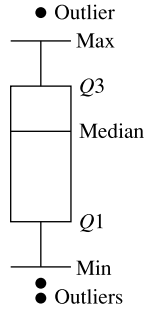
$$\begin{aligned} \hat{\mu} &= \frac{24.0 + 28.9 + 28.9 + 29.0 + 29.1 + 29.1 + 29.2 + 29.2 + 29.3 + 29.4}{10} = 28.61 \\ \hat{\sigma}^2 &= ((24.0 - 28.61)^2 + (28.9 - 28.61)^2 + (28.9 - 28.61)^2 + (29.0 - 28.61)^2 \\ &\quad + (29.1 - 28.61)^2 + (29.1 - 28.61)^2 + (29.2 - 28.61)^2 + (29.2 - 28.61)^2 \\ &\quad + (29.3 - 28.61)^2 + (29.4 - 28.61)^2) / 10 \simeq 2.29. \end{aligned}$$

Accordingly, we have  $\hat{\sigma} = \sqrt{2.29} = 1.51$ .

The most deviating value, 24.0°C, is 4.61°C away from the estimated mean. We know that the  $\mu \pm 3\sigma$  region contains 99.7% data under the assumption of normal distribution. Because  $\frac{4.61}{1.51} = 3.04 > 3$ , the probability that the value 24.0°C is generated by the normal distribution is less than 0.15% and thus can be identified as an outlier.  $\square$

Example 11.7 elaborates a simple yet practical outlier detection method. It simply labels any object as an outlier if it is more than  $3\sigma$  away from the mean of the estimated distribution, where  $\sigma$  is the standard deviation.

Such straightforward methods for statistical outlier detection can also be used in visualization. For example, the *boxplot method* (described in Chapter 2) plots the univariate input data using a five-number summary (Fig. 11.3): the smallest nonoutlier value (Min), the lower quartile ( $Q1$ ), the median ( $Q2$ ), the


**FIGURE 11.3**

Using a boxplot to visualize outliers.

upper quartile ( $Q3$ ), and the largest nonoutlier value ( $Max$ ). The *interquartile range* ( $IQR$ ) is defined as  $Q3 - Q1$ . Any object that is more than  $1.5 \times IQR$  smaller than  $Q1$  or  $1.5 \times IQR$  larger than  $Q3$  is treated as an outlier because the region between  $Q1 - 1.5 \times IQR$  and  $Q3 + 1.5 \times IQR$  contains 99.3% of the objects. The rationale is similar to using  $3\sigma$  as the threshold for normal distribution.

Another simple statistical method for univariate outlier detection using normal distribution is the *Grubb's test* (also known as the *maximum normed residual test*). For each object  $x$  in a data set, we define a  $z$ -score as

$$z = \frac{|x - \mu|}{\sigma}, \quad (11.4)$$

where  $\mu$  is the mean, and  $\sigma$  is the standard deviation of the input data. An object  $x$  is an outlier if

$$z \geq \frac{n-1}{\sqrt{n}} \sqrt{\frac{t_{\alpha/(2n), n-2}^2}{n-2 + t_{\alpha/(2n), n-2}^2}}, \quad (11.5)$$

where  $t_{\alpha/(2n), n-2}^2$  is the value taken by a  $t$ -distribution at a significance level of  $\alpha/(2n)$ , and  $n$  is the number of objects in the data set.

### Detection of multivariate outliers

Data involving two or more attributes or variables are *multivariate data*. Many univariate outlier detection methods can be extended to handle multivariate data. The central idea is to transform the multivariate outlier detection task into a univariate outlier detection problem. Here, we use two examples to illustrate this idea.

**Example 11.8. Multivariate outlier detection using the Mahalanobis distance.** For a multivariate data set, let  $\bar{o}$  be the sample mean vector. For an object  $o$  in the data set, the squared Mahalanobis distance from  $o$  to  $\bar{o}$  is

$$MDist(o, \bar{o}) = (o - \bar{o})^T S^{-1} (o - \bar{o}), \quad (11.6)$$

where  $S$  is the sample covariance matrix.

$MDist(\mathbf{o}, \bar{\mathbf{o}})$  is a univariate variable, and thus Grubb's test can be applied to this measure. Therefore we can transform the multivariate outlier detection tasks as follows:

1. Calculate the mean vector from the multivariate data set.
2. For each object  $\mathbf{o}$ , calculate  $MDist(\mathbf{o}, \bar{\mathbf{o}})$ , the squared Mahalanobis distance from  $\mathbf{o}$  to  $\bar{\mathbf{o}}$ .
3. Detect outliers in the transformed univariate data set,  $\{MDist(\mathbf{o}, \bar{\mathbf{o}}) | \mathbf{o} \in D\}$ .
4. If  $MDist(\mathbf{o}, \bar{\mathbf{o}})$  is determined to be an outlier, then  $\mathbf{o}$  is regarded as an outlier as well. □

Our second example uses the  $\chi^2$ -statistic to measure the distance between an object to the mean of the input data set.

**Example 11.9. Multivariate outlier detection using the  $\chi^2$ -statistic.** The  $\chi^2$ -statistic can also be used to capture multivariate outliers under the assumption of normal distribution. For an object,  $\mathbf{o}$ , the  $\chi^2$ -statistic is

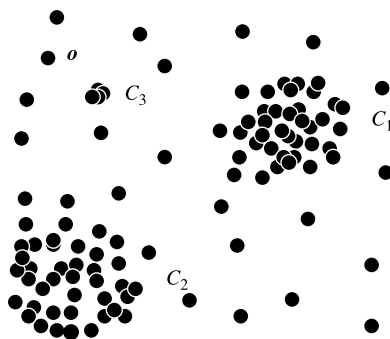
$$\chi^2 = \sum_{i=1}^n \frac{(\mathbf{o}_i - E_i)^2}{E_i}, \quad (11.7)$$

where  $\mathbf{o}_i$  is the value of  $\mathbf{o}$  on the  $i$ th dimension,  $E_i$  is the mean of the  $i$ -dimension among all objects, and  $n$  is the dimensionality. If the  $\chi^2$ -statistic is large, the object is an outlier. □

### **Using a mixture of parametric distributions**

If we assume that the data are generated by a normal distribution, this works well in many situations. However, this assumption may be overly simplified when the actual data distribution is complex. In such cases, we instead assume that the data are generated by a mixture of parametric distributions.

**Example 11.10. Multivariate outlier detection using multiple parametric distributions.** Consider the data set in Fig. 11.4. There are two big clusters,  $C_1$  and  $C_2$ . To assume that the data are generated by a normal distribution would not work well here. The estimated mean is located between the two clusters and not inside any cluster. The objects between the two clusters cannot be detected as outliers since they are close to the mean. □



**FIGURE 11.4**

A complex data set.

To overcome this problem, we can instead assume that the normal data objects are generated by multiple normal distributions, two in this case. That is, we assume two normal distributions,  $\Theta_1(\mu_1, \sigma_1)$  and  $\Theta_2(\mu_2, \sigma_2)$ . For any object  $\mathbf{o}$  in the data set, the probability that  $\mathbf{o}$  is generated by the mixture of the two distributions is given by

$$Pr(\mathbf{o}|\Theta_1, \Theta_2) = w_1 f_{\Theta_1}(\mathbf{o}) + w_2 f_{\Theta_2}(\mathbf{o}),$$

where  $f_{\Theta_1}$  and  $f_{\Theta_2}$  are the probability density functions of  $\Theta_1$  and  $\Theta_2$ , respectively, and  $w_1$  and  $w_2$  are the weights of two probability density functions. We can use the *expectation-maximization* (EM) algorithm (Chapter 9) to learn the parameters  $\mu_1, \sigma_1, \mu_2, \sigma_2, w_1$ , and  $w_2$  from the data, as we do in mixture models for clustering. Each cluster is represented by a learned normal distribution. An object  $\mathbf{o}$  is detected as an outlier if it does not belong to any cluster, that is, the probability is very low that it was generated by the combination of the two distributions.

**Example 11.11. Multivariate outlier detection using multiple clusters.** Most of the data objects shown in Fig. 11.4 are in either  $C_1$  or  $C_2$ . Other objects, representing noise, are uniformly distributed in the data space. A small cluster,  $C_3$ , is highly suspicious because it is not close to either of the two major clusters,  $C_1$  and  $C_2$ . The objects in  $C_3$  should therefore be detected as outliers.

Note that identifying the objects in  $C_3$  as outliers is difficult, whether or not we assume that the given data follow a normal distribution or a mixture of multiple distributions. This is because the probability of the objects in  $C_3$  will be higher than some of the noise objects, like  $\mathbf{o}$  in Fig. 11.4, due to a higher local density in  $C_3$ .  $\square$

To tackle the problem demonstrated in Example 11.11, we can assume that the normal data objects are generated by a normal distribution, or a mixture of normal distributions, whereas the outliers are generated by another distribution. Heuristically, we can add constraints on the distribution that is generating outliers. For example, it is reasonable to assume that this distribution has a larger variance if the outliers are distributed in a larger area. Technically, we can assign  $\sigma_{outlier} = k\sigma$ , where  $k$  is a user-specified parameter and  $\sigma$  is the standard deviation of the normal distribution generating the normal data. Again, the EM algorithm can be used to learn the parameters.

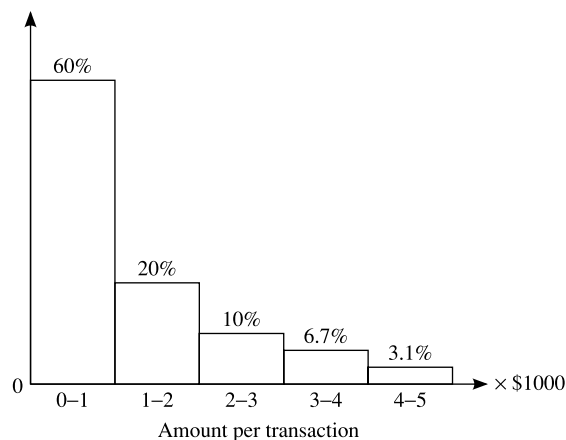
## 11.2.2 Nonparametric methods

In nonparametric methods for outlier detection, the model of “normal data” is learned from the input data, rather than assuming one a priori. Nonparametric methods often make fewer assumptions about the data, and thus can be applicable in more scenarios.

**Example 11.12. Outlier detection using a histogram.** An electronics store records the purchase amount for every customer transaction. Fig. 11.5 uses a histogram (refer to Chapter 2) to graph these amounts as percentages, given all transactions. For example, 60% of the transaction amounts are between \$0.00 and \$1000.

We can use the histogram as a nonparametric statistical model to capture outliers. For example, a transaction in the amount of \$7500 can be regarded as an outlier because only  $1 - (60\% + 20\% + 10\% + 6.7\% + 3.1\%) = 0.2\%$  of transactions have an amount higher than \$5000. On the other hand, a transaction amount of \$385 can be treated as normal because it falls into the bin (or bucket) holding 60% of the transactions.  $\square$



**FIGURE 11.5**

Histogram of purchase amounts in transactions.

As illustrated in the previous example, the histogram is a frequently used nonparametric statistical model that can be used to detect outliers. The procedure involves the following two steps.

**Step 1: Histogram construction.** In this step, we construct a histogram using the input data (training data). The histogram may be univariate as in Example 11.12 or multivariate if the input data are multidimensional.

Note that although nonparametric methods do not assume a priori statistical model, they often do require user-specified parameters to learn models from data. For example, to construct a good histogram, a user has to specify the type of histogram (e.g., equal width or equal depth) and other parameters (e.g., the number of bins in the histogram or the size of each bin). Unlike parametric methods, these parameters do not specify types of data distribution (e.g., Gaussian).

**Step 2: Outlier detection.** To determine whether an object  $o$  is an outlier, we can check it against the histogram. In the simplest approach, if the object falls in one of the histogram's bins, the object is regarded as normal. Otherwise, it is considered an outlier.

For a more sophisticated approach, we can use the histogram to assign an outlier-ness score to the object. In Example 11.12, we can let an object's outlier-ness score be the reciprocal of the volume of the bin in which the object falls. For example, the outlier-ness score for a transaction amount of \$7500 is  $\frac{1}{0.2\%} = 500$  and that for a transaction amount of \$385 is  $\frac{1}{60\%} = 1.67$ . The scores indicate that the transaction amount of \$7500 is much more likely to be an outlier than that of \$385.

A drawback of using histograms as a nonparametric model for outlier detection is that it is hard to choose an appropriate bin size. On the one hand, if the bin size is set too small, many normal objects may end up in empty or rare bins and thus be misidentified as outliers. This leads to a high false positive rate and low precision. On the other hand, if the bin size is set too high, outlier objects may infiltrate into some frequent bins and thus be "disguised" as normal. This leads to a high false negative rate and low recall.

To overcome this problem, we can adopt kernel density estimation to estimate the probability density distribution of the data. We treat an observed object as an indicator of high probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects. We use a *kernel function* to model the influence of a sample point within its neighborhood. A kernel  $K()$  is a nonnegative real-valued integrable function that satisfies the following two conditions:

- $\int_{-\infty}^{+\infty} K(u)du = 1$ .
- $K(-u) = K(u)$  for all values of  $u$ .

A frequently used kernel is a standard Gaussian function with mean 0 and variance 1:

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}. \quad (11.8)$$

Let  $x_1, \dots, x_n$  be an independent and identically distributed samples of a random variable  $f$ . The kernel density approximation of the probability density function is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right), \quad (11.9)$$

where  $K()$  is a kernel and  $h$  is the bandwidth serving as a smoothing parameter.

Once the probability density function of a data set is approximated through kernel density estimation, we can use the estimated density function  $\hat{f}$  to detect outliers. For an object  $o$ ,  $\hat{f}(o)$  gives the estimated probability that the object is generated by the stochastic process. If  $\hat{f}(o)$  is high, then the object is likely normal. Otherwise,  $o$  is likely an outlier. This step is often similar to the corresponding step in parametric methods.

In summary, statistical methods for outlier detection learn models from data to distinguish outliers from normal data objects. An advantage of using statistical methods is that the outlier detection may be statistically justifiable. Of course, this is true only if the statistical assumption made about the underlying data meets the constraints in reality.

The data distribution of high-dimensional data is often complicated and hard to be fully understood. Consequently, statistical methods for outlier detection on high-dimensional data remain a big challenge. Outlier detection for high-dimensional data is further addressed in Section 11.7.

The computational cost of statistical methods depends on the models. When simple parametric models are used (e.g., a Gaussian), fitting the parameters typically takes linear time. When more sophisticated models are used (e.g., mixture models, where the EM algorithm is used in learning), approximating the best parameter values often takes several iterations. Each iteration, however, is typically linear with respect to the data set's size. For kernel density estimation, the model learning cost can be up to quadratic. Once the model is learned, the outlier detection cost is often very small per object.

### 11.3 Proximity-based approaches

Given a set of objects in feature space, a distance measure can be used to quantify the similarity between objects. Intuitively, objects that are far from others can be regarded as outliers. Proximity-based approaches assume that the proximity of an outlier object to its nearest neighbors significantly deviates from the proximity of most other objects to their nearest neighbors in the data set.

There are two types of proximity-based outlier detection methods: distance-based and density-based methods. A *distance-based outlier detection method* consults the **neighborhood** of an object, which is defined by a given radius. An object is then considered as an outlier if its neighborhood does not have enough other points. A *density-based outlier detection method* investigates the density of an object and that of its neighbors. Here, an object is identified as an outlier if its density is relatively much lower than that of its neighbors.

Let's start with distance-based outliers.

#### 11.3.1 Distance-based outlier detection

A representative method of proximity-based outlier detection uses the concept of **distance-based outliers**. For a set,  $D$ , of data objects to be analyzed, a user can specify a distance threshold,  $r$ , to define a reasonable neighborhood of an object. For each object,  $\mathbf{o}$ , we can examine the number of other objects in the  $r$ -neighborhood of  $\mathbf{o}$ . If most of the objects in  $D$  are far from  $\mathbf{o}$ , that is, not in the  $r$ -neighborhood of  $\mathbf{o}$ , then  $\mathbf{o}$  can be regarded as an outlier.

Formally, let  $r$  ( $r \geq 0$ ) be a *distance threshold* and  $\pi$  ( $0 < \pi \leq 1$ ) be a fraction threshold. An object,  $\mathbf{o}$ , is a  $DB(r, \pi)$ -outlier if

$$\frac{\|\{\mathbf{o}' \mid \text{dist}(\mathbf{o}, \mathbf{o}') \leq r\}\|}{\|D\|} \leq \pi, \quad (11.10)$$

where  $\text{dist}(\cdot, \cdot)$  is a distance measure.

Equivalently, we can determine whether an object,  $\mathbf{o}$ , is a  $DB(r, \pi)$ -outlier by checking the distance between  $\mathbf{o}$  and its  $k$ -nearest neighbor,  $\mathbf{o}_k$ , where  $k = \lceil \pi \|D\| \rceil$ . Object  $\mathbf{o}$  is an outlier if  $\text{dist}(\mathbf{o}, \mathbf{o}_k) > r$ , because in such a case, there are fewer than  $k$  objects except for  $\mathbf{o}$  that are in the  $r$ -neighborhood of  $\mathbf{o}$ .

“How can we compute  $DB(r, \pi)$ -outliers?” A straightforward approach is to use nested loops to check the  $r$ -neighborhood for every object, as shown in Fig. 11.6. For any object,  $\mathbf{o}_i$  ( $1 \leq i \leq n$ ), we calculate the distance between  $\mathbf{o}_i$  and the other object and count the number of other objects in the  $r$ -neighborhood of  $\mathbf{o}_i$ . Once we find  $\pi \cdot n$  other objects within a distance  $r$  from  $\mathbf{o}_i$ , the inner loop can be terminated because  $\mathbf{o}_i$  already violates (Eq. (11.10)), and thus is not a  $DB(r, \pi)$ -outlier. On the other hand, if the inner loop completes for  $\mathbf{o}_i$ , this means that  $\mathbf{o}_i$  has less than  $\pi \cdot n$  neighbors in a radius of  $r$ , and thus is a  $DB(r, \pi)$ -outlier.

The straightforward nested loop approach takes  $O(n^2)$  time. Surprisingly, the actual CPU runtime is often linear with respect to the data set size. For most nonoutlier (i.e., normal) objects, the inner loop terminates early when the number of outliers in the data set is small, which should be the case most of the time. Correspondingly, only a small fraction of the data set is examined.

**Algorithm:** Distance-based outlier detection.

**Input:**

- a set of objects  $D = \{o_1, \dots, o_n\}$ , threshold  $r$  ( $r > 0$ ) and  $\pi$  ( $0 < \pi \leq 1$ );

**Output:**  $DB(r, \pi)$  outliers in  $D$ .

**Method:**

```

for  $i = 1$  to  $n$  do
   $count \leftarrow 0$ 
  for  $j = 1$  to  $n$  do
    if  $i \neq j$  and  $dist(o_i, o_j) \leq r$  then
       $count \leftarrow count + 1$ 
      if  $count \geq \pi \cdot n$  then
        exit  $\{o_i$  cannot be a  $DB(r, \pi)$  outlier $\}$ 
      endif
    endif
  endfor
  print  $o_i$   $\{o_i$  is a  $DB(r, \pi)$  outlier according to (Eq. 11.10) $\}$ 
endfor;

```

**FIGURE 11.6**

Nested loop algorithm for  $DB(r, \pi)$ -outlier detection.

### 11.3.2 Density-based outlier detection

Distance-based outliers, such as  $DB(r, \pi)$ -outliers, are just one type of outlier. Specifically, distance-based outlier detection takes a global view of the data set. Such outliers can be regarded as “global outliers” for two reasons:

- A  $DB(r, \pi)$ -outlier, for example, is far (as quantified by parameter  $r$ ) from at least  $(1 - \pi) \times 100\%$  of the objects in the data set. In other words, an outlier as such is remote from the majority of the data.
- To detect distance-based outliers, we need two global parameters,  $r$  and  $\pi$ , which are applied to every outlier object.

Many real-world data sets demonstrate a more complex structure, where objects may be considered outliers with respect to their local neighborhoods rather than with respect to the global data distribution. Let’s look at an example.

**Example 11.13. Local proximity-based outliers.** Consider the data points in Fig. 11.7. There are two clusters:  $C_1$  is dense, and  $C_2$  is sparse. Object  $o_3$  can be detected as a distance-based outlier because it is far from the majority of the data set.

Now, let’s consider objects  $o_1$  and  $o_2$ . Are they outliers? On the one hand, the distance from  $o_1$  and  $o_2$  to the objects in the dense cluster,  $C_1$ , is smaller than the average distance between an object in cluster  $C_2$  and its nearest neighbor. Thus,  $o_1$  and  $o_2$  are not distance-based outliers. In fact, if we were to categorize  $o_1$  and  $o_2$  as  $DB(r, \pi)$ -outliers, we would have to classify all the objects in cluster  $C_2$  as  $DB(r, \pi)$ -outliers.

On the other hand,  $o_1$  and  $o_2$  can be identified as outliers when they are considered locally with respect to cluster  $C_1$  because  $o_1$  and  $o_2$  deviate significantly from the objects in  $C_1$ . Moreover,  $o_1$  and  $o_2$  are also far from the objects in  $C_2$ .

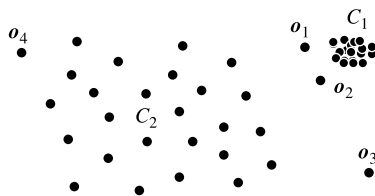


FIGURE 11.7

Global outliers and local outliers.

To summarize, distance-based outlier detection methods cannot capture local outliers like  $o_1$  and  $o_2$ . Note that the distance between object  $o_4$  and its nearest neighbors is much greater than the distance between  $o_1$  and its nearest neighbors. However, because  $o_4$  is local to cluster  $C_2$  (which is sparse),  $o_4$  is not considered a local outlier.  $\square$

“How can we formulate the local outliers as illustrated in Example 11.13?” The critical idea here is that we need to compare the density around an object with the density around its local neighbors. The basic assumption of density-based outlier detection methods is that the density around a nonoutlier object is similar to the density around its neighbors, while the density around an outlier object is significantly different from the density around its neighbors.

Based on the preceding, density-based outlier detection methods use the relative density of an object against its neighbors to indicate the degree to which an object is an outlier.

Now, let’s consider how to measure the *relative density* of an object,  $o$ , given a set of objects,  $D$ . The  $k$ -distance of  $o$ , denoted by  $dist_k(o)$ , is the distance,  $dist(o, p)$ , between  $o$  and another object,  $p \in D$ , such that

- There are at least  $k$  objects  $o' \in D \setminus \{o\}$  such that  $dist(o, o') \leq dist(o, p)$ .
- There are at most  $k - 1$  objects  $o'' \in D \setminus \{o\}$  such that  $dist(o, o'') < dist(o, p)$ .

In other words,  $dist_k(o)$  is the distance between  $o$  and its  $k$ -nearest neighbor. Consequently, the  $k$ -distance neighborhood of  $o$  contains all objects of which the distance to  $o$  is not greater than  $dist_k(o)$ , the  $k$ -distance of  $o$ , denoted by

$$N_k(o) = \{o' \mid o' \in D, dist(o, o') \leq dist_k(o)\}. \quad (11.11)$$

Note that  $N_k(o)$  may contain more than  $k$  objects because multiple objects may each be the same distance away from  $o$ .

We can use the average distance from the objects in  $N_k(o)$  to  $o$  as the measure of the local density of  $o$ . However, such a straightforward measure has a problem: If  $o$  has very close neighbors  $o'$  such that  $dist(o, o')$  is very small, the statistical fluctuations of the distance measure can be undesirably high. To overcome this problem, we can switch to the following reachability distance measure by adding a smoothing effect.

For two objects,  $o$  and  $o'$ , the *reachability distance* from  $o'$  to  $o$  is  $dist(o, o')$  if  $dist(o, o') > dist_k(o)$ , and  $dist_k(o)$  otherwise. That is,

$$reachdist_k(o \leftarrow o') = \max\{dist_k(o), dist(o, o')\}. \quad (11.12)$$

Here,  $k$  is a user-specified parameter that controls the smoothing effect. Essentially,  $k$  specifies the minimum neighborhood to be examined to determine the local density of an object. Importantly, reachability distance is not symmetric, that is, in general,  $reachdist_k(\mathbf{o} \leftarrow \mathbf{o}') \neq reachdist_k(\mathbf{o}' \leftarrow \mathbf{o})$ .

Now, we can define the *local reachability density* of an object,  $\mathbf{o}$ , as

$$lrd_k(\mathbf{o}) = \frac{\|N_k(\mathbf{o})\|}{\sum_{\mathbf{o}' \in N_k(\mathbf{o})} reachdist_k(\mathbf{o}' \leftarrow \mathbf{o})}. \quad (11.13)$$

There is a critical difference between the density measure here for outlier detection and that in density-based clustering (Section 11.5.1). In density-based clustering, to determine whether an object can be considered a core object in a density-based cluster, we use two parameters: a radius parameter,  $r$ , to specify the range of the neighborhood, and the minimum number of points in the  $r$ -neighborhood. Both parameters are global and are applied to every object. In contrast, as motivated by the observation that relative density is the key to finding local outliers, we use the parameter  $k$  to quantify the neighborhood and do not need to specify the minimum number of objects in the neighborhood as a requirement of density. We instead calculate the local reachability density for an object and compare it with that of its neighbors to quantify the degree to which the object is considered an outlier.

Specifically, we define the *local outlier factor* of an object  $\mathbf{o}$  as

$$LOF_k(\mathbf{o}) = \frac{\sum_{\mathbf{o}' \in N_k(\mathbf{o})} \frac{lrd_k(\mathbf{o}')}{lrd_k(\mathbf{o})}}{\|N_k(\mathbf{o})\|} = \sum_{\mathbf{o}' \in N_k(\mathbf{o})} lrd_k(\mathbf{o}') \cdot \sum_{\mathbf{o}' \in N_k(\mathbf{o})} reachdist_k(\mathbf{o}' \leftarrow \mathbf{o}). \quad (11.14)$$

In other words, the local outlier factor is the average of the ratio of the local reachability density of  $\mathbf{o}$  and those of  $\mathbf{o}$ 's  $k$ -nearest neighbors. The lower the local reachability density of  $\mathbf{o}$  (i.e., the smaller the item  $\sum_{\mathbf{o}' \in N_k(\mathbf{o})} reachdist_k(\mathbf{o}' \leftarrow \mathbf{o})$ ) and the higher the local reachability densities of the  $k$ -nearest neighbors of  $\mathbf{o}$ , the higher the LOF value is. This exactly captures a local outlier of which the local density is relatively low compared to the local densities of its  $k$ -nearest neighbors.

The local outlier factor has some nice properties. First, for an object deep within a consistent cluster, such as the points in the center of cluster  $C_2$  in Fig. 11.7, the local outlier factor is close to 1. This property ensures that objects inside clusters, no matter whether the cluster is dense or sparse, will not be mislabeled as outliers.

Second, for an object  $\mathbf{o}$ , the meaning of  $LOF(\mathbf{o})$  is easy to understand. Consider the objects in Fig. 11.8, for example. For object  $\mathbf{o}$ , let

$$direct_{min}(\mathbf{o}) = \min\{reachdist_k(\mathbf{o}' \leftarrow \mathbf{o}) \mid \mathbf{o}' \in N_k(\mathbf{o})\} \quad (11.15)$$

be the minimum reachability distance from  $\mathbf{o}$  to its  $k$ -nearest neighbors. Similarly, we can define

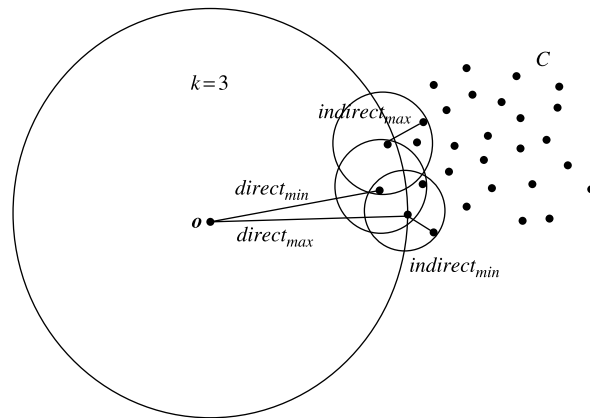
$$direct_{max}(\mathbf{o}) = \max\{reachdist_k(\mathbf{o}' \leftarrow \mathbf{o}) \mid \mathbf{o}' \in N_k(\mathbf{o})\}. \quad (11.16)$$

We also consider the neighbors of  $\mathbf{o}$ 's  $k$ -nearest neighbors. Let

$$indirect_{min}(\mathbf{o}) = \min\{reachdist_k(\mathbf{o}'' \leftarrow \mathbf{o}') \mid \mathbf{o}' \in N_k(\mathbf{o}) \text{ and } \mathbf{o}'' \in N_k(\mathbf{o}')\} \quad (11.17)$$

and

$$indirect_{max}(\mathbf{o}) = \max\{reachdist_k(\mathbf{o}'' \leftarrow \mathbf{o}') \mid \mathbf{o}' \in N_k(\mathbf{o}) \text{ and } \mathbf{o}'' \in N_k(\mathbf{o}')\}. \quad (11.18)$$




---

**FIGURE 11.8**

A property of  $LOF(o)$ .

Then, it can be shown that  $LOF(o)$  is bounded as

$$\frac{direct_{min}(o)}{indirect_{max}(o)} \leq LOF(o) \leq \frac{direct_{max}(o)}{indirect_{min}(o)}. \quad (11.19)$$

This result clearly shows that LOF captures the relative density of an object.

---

## 11.4 Reconstruction-based approaches

The central idea behind reconstruction-based outlier detection approaches is as follows. Since the normal data samples share certain similarities, they can often be represented in a more succinct way, compared with their original representation (e.g., an attribute vector for each data sample). With the succinct representation, we can well *reconstruct* the original representation of the normal samples. On the other hand, for samples that cannot be well reconstructed by such alternative, succinct representation, we flag them as outliers.

**Example 11.14.** Given a set of researchers in Fig. 11.9(a), where each researcher is described by the list of venues the corresponding author publishes. Alternatively, we use a more succinct way, namely the research areas, to represent each author, as shown in Fig. 11.9(b). For most authors, we can use such succinct representation to perfectly reconstruct their original representation in Fig. 11.9(a). For example, given the succinct representation “Data Mining” for John, Tom and Bob, we can infer (i.e., reconstruct) that their publication venues are “KDD” and “ICDM.” Likewise, given the succinct representation “Software Engineering” for Van and Roy, we can infer (i.e., reconstruct) that their publication venues are “FSE” and “ICSE.” Therefore we conclude that they are all “normal” researchers, which in this example means that they all focus on a specific research area (e.g., either “Data Mining” or “Software Engineering”). However, for Carl, we can only infer that his publication venues would be

| Researchers | Publication Venues (Original) | Researchers | Publication Areas (Succinct) |
|-------------|-------------------------------|-------------|------------------------------|
| John        | KDD, ICDM                     | John        | Data Mining                  |
| Tom         | KDD, ICDM                     | Tom         | Data Mining                  |
| Bob         | KDD, ICDM                     | Bob         | Data Mining                  |
| Carl        | ICDM, FSE, ICSE               | Carl        | Software Engineering         |
| Van         | FSE, ICSE                     | Van         | Software Engineering         |
| Roy         | FSE, ICSE                     | Roy         | Software Engineering         |

(a) Original representation

(b) Succinct representation

**FIGURE 11.9**

An example of using reconstruction-based approaches to detect outlying researchers. “KDD” and “ICDM” are two conferences in the area of data mining; and “FSE” and “ICSE” are two conferences in software engineering. (a) The original representation of authors, using the publication venues. (b) The succinct representation of authors, using research areas.

“FSE” and “ICSE” based on his succinct representation (“Software Engineering”), which misses an important venue he actually publishes (i.e., “ICDM”). In other words, there is a discrepancy between Carl’s original representation and reconstructed one. We say that the reconstruction quality for Carl is low. Therefore, we flag him as an outlier, which in this example means that he might be a multi-disciplinary researcher who publishes mainly “Software Engineering” but also publishes in some data mining venues (e.g., “ICDM”). Conceptually, if we treat each succinct representation (e.g., “Data Mining” vs. “Software Engineering”) as a cluster, Carl is flagged as an outlier in this example, because he is the only researcher who belongs to both clusters, whereas each of the remaining researchers only belongs to a single cluster (i.e., either “Data Mining” or “Software Engineering”).<sup>2</sup> □

The key questions in a reconstruction-based outlier detection method include (Q1) how to find the succinct representation; (Q2) how to use the succinct representation to reconstruct the original data samples; and (Q3) how to measure the quality (i.e., goodness) of reconstruction. In this section, you will learn two types of reconstruction-based outlier detection approaches, including (1) matrix-factorization based methods for numerical data and (2) pattern-based compression methods for categorical data.

### 11.4.1 Matrix factorization–based methods for numerical data

For data with numerical attributes (i.e., features), we represent each data sample as an attribute vector and the entire input data set is then represented by a *data matrix*, whose rows are different data samples, columns are different attributes, and entries of the data matrix indicate the attribute values of the corresponding data samples. In this setting, a powerful technique for detecting outliers is *matrix factorization*. In this method, we approximate the data matrix by two or more *low-rank* matrices, which serve as the succinct representation of the original data matrix and meanwhile, provide a natural way

<sup>2</sup> Notice that this is different from the clustering-based outlier detection that will be introduced in Section 11.5.1, where an outlier is often defined as a data tuple that either does not belong to any cluster or is far away from the cluster center it belongs to.



**Algorithm: Matrix factorization** based outlier detection

**Input:**

- $X_i = (X_{i,1}, \dots, X_{i,m})$  ( $i = 1, \dots, n$ ),  $n$  data samples each of which is represented by an  $m$  dimensional numerical attribute vector;
- $r$ , the rank;
- $k$ , the number of outliers.

**Output:** A set of  $k$  outliers.

**Method:**

- //each row of  $X$  is a data sample
- (1) Represent the input data samples as an  $n \times m$  data matrix  $X$ ;
  - (2) Approximate the data matrix by two rank  $r$ -matrices:  $X \approx FG$ ;
  - (3) **for** ( $i = 1, \dots, n$ ) { // for each sample  
     //compute the reconstruction error  
     // $F(i, j)$  is the element of  $F$  at the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  column  
     // $G(j, :)$  is the  $j^{\text{th}}$  row of  $G$
  - (4)     Compute  $r_i = \|X_i - \hat{X}_i\|^2 = \|X_i - \sum_{j=1}^r F(i, j)G(j, :)\|^2$ ;
  - (5)     }
  - (5) Return  $k$  samples with the largest reconstruction errors  $r_i$  ( $i = 1, \dots, n$ ).

**FIGURE 11.10**

Matrix factorization–based outlier detection.

to reconstruct the original data matrix. The *reconstruction error* of each data sample is used as an indicator of the outlier-ness: the higher the reconstruction error, the more likely the given data sample is an outlier. The algorithm is summarized in Fig. 11.10 and the details are depicted as follows.

We start with representing the input data samples in the form of a data matrix  $X$  (step 1), whose rows represent data samples and columns are attributes (i.e., features). Then (step 2), we approximate the data matrix  $X$  by the multiplication of two matrices  $F$  and  $G$ . Here, an important point is that the rank of  $F$  and  $G$  should be much smaller than the data matrix (i.e.,  $r \ll m$ ). In this way, the  $r$  rows of matrix  $G$  provide a more succinct way to represent the input data samples, that is,  $G(j, :)$  ( $j = 1, \dots, r$ ) are the new, succinct representation. In the meanwhile, matrix  $F$  tells us how to use such succinct representation to *reconstruct* the original data sample, that is,  $\hat{X}_i = \sum_{j=1}^r F(i, j)G(j, :)$ , where  $F(i, j)$  is the element of  $F$  at the  $i$ th row and the  $j$ th column. The squared distance between the original data sample  $X_i$  and the reconstructed one  $\hat{X}_i$  is called *reconstruction error* (step 4), and it measures the outlier-ness of the corresponding data sample. Finally, in step 5, we return  $k$  data samples with the highest reconstruction errors as the outliers.

**Example 11.15.** For Example 11.14, we represent each author as a 4-D binary vector, indicating if the author publishes in the corresponding conferences, where the four dimensions are the four conferences, including “KDD,” “ICDM,” “FSE,” and “ICSE,” respectively. For example, John is represented as (1, 1, 0, 0), meaning that he publishes in both “KDD” and “ICDM”; Carl is represented as (0, 1, 1, 1), meaning that he publishes in all conferences but “KDD.” We organize the feature vectors of all six authors as a  $6 \times 4$  data matrix  $X$  (the left part of Fig. 11.11). Then, we approximate the data matrix  $X$  by the multiplication of two low-rank matrices  $F$  and  $G$ , both with a rank of 2 (the middle part of Fig. 11.11). The two rows of matrix  $G$  provide an alternative, succinct representation compared with the original four features in the data matrix  $X$ . For example, the first row of

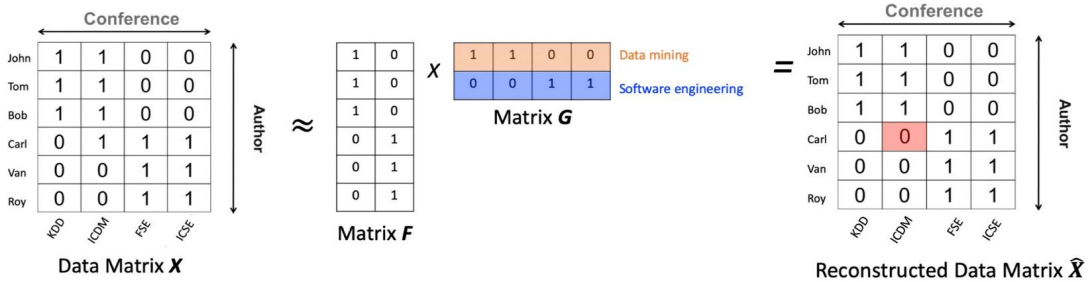


FIGURE 11.11

Matrix factorization based method to detect outliers for Example 11.14. By approximating the original data matrix (on the left) by the multiplication of two low-rank matrices  $F$  and  $G$  (in the middle), it provides an alternative, more succinct representation of the original features. The reconstruction error of each author by comparing the original and the reconstructed representation (on the right) provides an indicator of the outlier-ness of each author. In this example, the reconstruction errors for all researchers except Carl is 0, and thus Carl is flagged as an outlying author.

$G$  represents data mining research area, and the second row of  $G$  represents the software engineering research area. With the help of the rows of  $F$ , we can use the succinct representation in  $G$  to reconstruct the original representation of each author. For example, the reconstructed representation for John is  $1 \times (1, 1, 0, 0) + 0 \times (0, 0, 1, 1) = (1, 1, 0, 0)$ , and the reconstructed representation for Carl is  $0 \times (1, 1, 0, 0) + 1 \times (0, 0, 1, 1) = (0, 0, 1, 1)$ . We put the reconstructed representation of all six authors in another matrix  $\hat{X}$  (the right part of Fig. 11.11). Then, we compute the squared distance between the rows of the original data matrix  $X$  and that of the reconstructed data matrix  $\hat{X}$ , which are used as the outlier-ness scores of the corresponding authors. In this example, the reconstruction error is zero for all authors but Carl, who has a reconstruction error of 1. Therefore we flag Carl as an outlying author.  $\square$

“So, how can we find these two magic low-rank matrices  $F$  and  $G$ ?” Many approaches exist. A popular choice is to use *singular value decomposition* (SVD). Given an input data matrix  $X$ , SVD approximates it by the multiplication of three rank- $r$  matrices

$$X \approx U \Sigma V', \tag{11.20}$$

where  $U$  and  $V$  are *orthonormal matrices* and their columns are called left singular vectors and right singular vectors of data matrix  $X$ , respectively,  $V'$  is the transpose of matrix  $V$ , and  $\Sigma$  is a diagonal with nonnegative diagonal elements called singular values. For the purpose of outlier detection, we can set  $F = U \Sigma$  and  $G = V'$ . See Fig. 11.12 for an illustration.

The full mathematical details of SVD are outside the scope of this textbook. Many software packages exist to compute SVD for a given data matrix. An appealing property of SVD lies in its *optimal approximation*. This means that among all rank- $r$  matrices, SVD provides the best approximation of the original data matrix in terms of both  $L_2$  norm and Frobenius norm. Another interesting property of SVD lies in its close connection with PCA. In particular, if the data matrix  $X$  is “centered,” meaning that each column (i.e., feature) of  $X$  has a zero sample mean (we can achieve this, say by z-score normalization), the columns of matrix  $V$  (i.e., the right singular vectors) are exactly the first  $r$  principle components of the input data. (Why this is true is left as an exercise.)

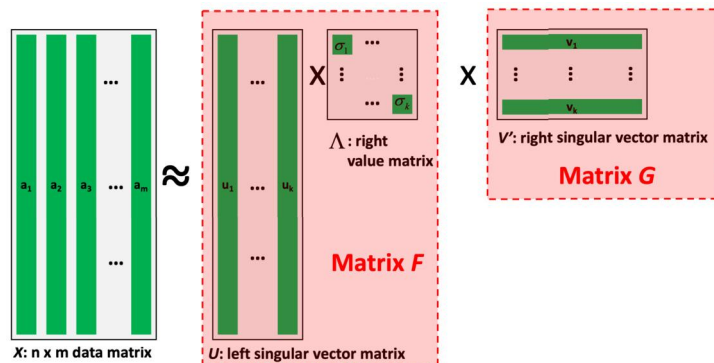


FIGURE 11.12

An illustration of SVD. Given a data matrix  $X$  (on the left), SVD approximates it by the multiplication of three rank- $r$  matrices. The multiplication of the left singular vector matrix  $U$  and the singular value matrix  $\Sigma$  becomes matrix  $F$  in Fig. 11.11 and the transpose of the right singular vector matrix  $V'$  becomes matrix  $G$  in Fig. 11.11. When the data matrix  $X$  is centered (i.e., each of its columns has zero sample mean), the columns of  $V$  are the first  $r$  principal components of  $X$ .

A potential limitation of SVD is that the low-rank matrices ( $F$  and  $G$ ) are often dense, even when the input data matrix  $X$  itself is sparse, which makes the detection results somehow hard for the end user to interpret. To address this issue, *example-based matrix factorization* has been developed. For instance, CX-decomposition first samples (with replacement) a few columns of the data matrix  $X$  to form matrix  $F$ , and then finds matrix  $G$  by projecting the data matrix  $X$  onto the column space spanned by matrix  $F$ .<sup>3</sup> In CX-decomposition, there might exist redundancy among the columns in matrix  $F$  with duplicated or linearly correlated columns that have no impact on improving the reconstruction error (the indicator for outlier detection) yet waste both time and space. An improved example-based matrix factorization called *Colibri* only uses linearly independent columns of the data matrix  $X$  to construct the matrix  $F$ . Both CX-decomposition and Colibri were found to improve the efficiency and interpretability of outlier detection.

**Example 11.16.** Suppose we are given a set of authors, each of whom is represented by how many papers she has published in two conferences, including “KDD”—a data mining conference, and “ISBM”—a computational biology conference. Therefore the input data matrix  $X$  is an  $n \times 2$  matrix, where each row (i.e., a circle in Fig. 11.13(a)) is an author and  $n$  is the number of authors. If we use SVD to factorize the data matrix  $X \approx FG$ , each column of matrix  $G$  is a right singular vector of  $X$  that is a linear combination of *all* rows of the data matrix (Fig. 11.13(b)). Therefore the resulting matrix  $G$  is dense, and so is matrix  $F$ . In contrast, example-based factorization (e.g., CX-decomposition and Colibri in Fig. 11.13(c-d)) uses actual data samples (e.g., sampled authors) to construct matrix  $G$ , which is more interpretable and computationally more efficient than SVD. In order to detect outlying authors,

<sup>3</sup> Mathematically, let  $F$  consist of columns sampled from the data matrix  $X$ . We approximate  $X$  as  $X \approx F(F'F)^+F'X$ , where  $+$  is the matrix pseudo-inverse. We set matrix  $G = (F'F)^+F'X$ .

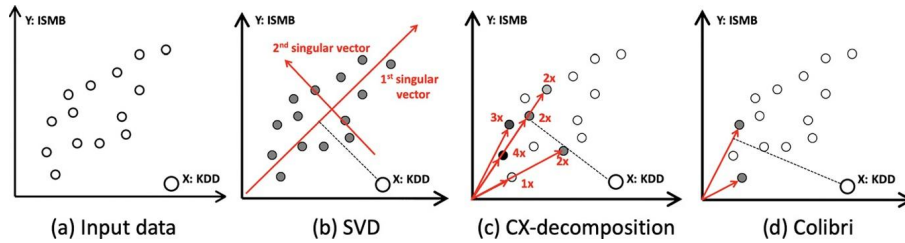


FIGURE 11.13

A pictorial comparison between SVD, CX-decomposition, and Colibri. (a) Input data: each circle is an author who is represented by two features (“KDD” and “ISMB”). (b) SVD finds optimal factorization, where the two right singular vectors (two red (gray in print version) arrows), together with the two corresponding singular values, form the matrix  $G$ . Each right singular vector is a linear combination of all input authors, resulting in a dense matrix  $G$ . (c) CX-decomposition randomly samples some actual authors (gray circles) to form matrix  $G$ , and some sampled columns might be duplicated (indicated by the number of the corresponding circle). (d) Colibri uses two linearly independent, sampled authors (two gray circles) to form the matrix  $G$ . Using the first singular vector (b), or one sampled data point by either CX-decomposition (c) or Colibri (d), we might flag the author on the right bottom corner as an outlier, with a high reconstruction error (the dashed line).

we can use the first singular vector (b), or one sampled data point by either CX-decomposition (c) or Colibri (d) as the succinct representation. Then, an author (e.g., the one on the right bottom corner) with a high reconstruction error (indicated by the length of the dashed line in the figure) is flagged as an outlier. Notice that the meaning of outliers found in this example is quite different from Example 11.14. In Example 11.14, most (i.e., normal) authors belong to one of the two research areas (“Data Mining” or “Software Engineering”), and thus Carl, who belongs to both areas, is flagged as an outlying author. In contrast, in this example, for most (i.e., normal) authors, the numbers of publications in “ISMB” are positively correlated with that in “KDD,” which is captured by the succinct representation (e.g., the first singular vector). Therefore the author on the right bottom corner, with a significant number of publications in “KDD” yet almost none in “ISMB,” is flagged as an outlier.  $\square$

When the input data matrix  $X$  is *nonnegative*, a natural choice is to use nonnegative matrix factorization (NMF, which was introduced in Chapter 9). In general, nonnegative entries are easier to interpret since they often correspond to the actual “parts” of the input data (e.g., a nose, an eye if the input data is face image). A typical NMF method imposes the nonnegativity constraints on the two factorized matrices ( $F$  and  $G$ ). However, in Fig. 11.11, we use the *residual matrix*  $R = X - FG$  to detect outlying data samples (e.g., if the squared norm of a row of the residual matrix  $R$  is high, the corresponding data are flagged as an outlier). Therefore it is natural to require the residual matrix  $R$ , instead of the factorized matrices  $F$  and  $G$ , to be nonnegative. Matrix factorization with this type of constraint (i.e., the entries of the residual matrix must be nonnegative) is called *nonnegative residual matrix factorization* (NrMF). NrMF was found to be more interpretable to detect outliers which corresponds to some actual behaviors or activities of certain data samples.

**Example 11.17.** An IP source might be detected as a suspicious port-scanner if it *sends* packages to a lot of destinations in an IP traffic network; we might flag a group of users who always give good ratings

to another group of users as a collusion type of fake reviewers. If we map such behaviors or activities (e.g., “sends packages,” “gives good ratings,” etc.) to the language of matrix factorization, it suggests that the corresponding entries in the residual matrix should be nonnegative. □

Another method, called *robust PCA*, requires that the residual matrix  $\mathbf{R}$  to be sparse with most of the rows being empty. The intuition is as follows. For the vast majority of normal data tuples, they can be perfectly reconstructed by the factorized matrices, whereas for a small number of outlying tuples, they bear nonzero reconstruction errors indicated by the corresponding, nonempty rows in the residual matrix  $\mathbf{R}$ .

Matrix factorization based methods are linear methods in that the new, more succinct representation is a linear combination of the original features. For example, in SVD, each right singular vector is a linear combination of the original features of the input data. For the example in Fig. 11.9, the new representation “Data Mining” is the result of a linear combination of the two conferences including “KDD” and “ICDM.” If we wish to capture the *nonlinear* relationship between the original features, we can use *autoencoder*, an unsupervised deep learning architecture introduced in Chapter 10, to reconstruct the original data samples. Again, the data samples with high reconstruction errors are flagged as outliers.

### 11.4.2 Pattern-based compression methods for categorical data

For input data with categorical attributes, we could convert the categorical attributes to binary attributes and then apply the matrix factorization based methods to detect outliers.

**Example 11.18.** Given an input data set in Fig. 11.14, where each row represents a customer who is described by three categorical attributes, namely “income,” “credit,” and “purchase.” In order to detect abnormal customers, we first convert the input data set into a binary data matrix  $\mathbf{B}$  in Fig. 11.15(a). Since each categorical attribute has three possible values, including “High,” “Medium,” and “Low,” it is converted into three binary attributes. (How to convert a categorical attribute to binary ones was introduced in Chapter 2.) Next, we approximate the binary data matrix  $\mathbf{B}$  by the multiplication of two low-rank matrices, say the ones in Fig. 11.15(b), which are in turn used to reconstruct the original data matrix  $\mathbf{B}$ . By comparing the original and reconstructed data matrices, we find out that the reconstruction errors for the first four customers are all zeros. On the other hand, the reconstruction errors for Tom and Jim are 2.5 and 1.34, respectively, both of whom are thus flagged as outliers. □

|      | Income | Credit | Purchase |
|------|--------|--------|----------|
| John | High   | High   | High     |
| Amy  | High   | High   | High     |
| Carl | Low    | Low    | Low      |
| Mary | Low    | Low    | Low      |
| Tom  | High   | Low    | Medium   |
| Jim  | High   | Low    | Low      |

FIGURE 11.14

Input data with categorical attributes, including “Income,” “Credit,” and “Purchase.” Each categorical attribute has three values, including “High,” “Medium,” and “Low.”

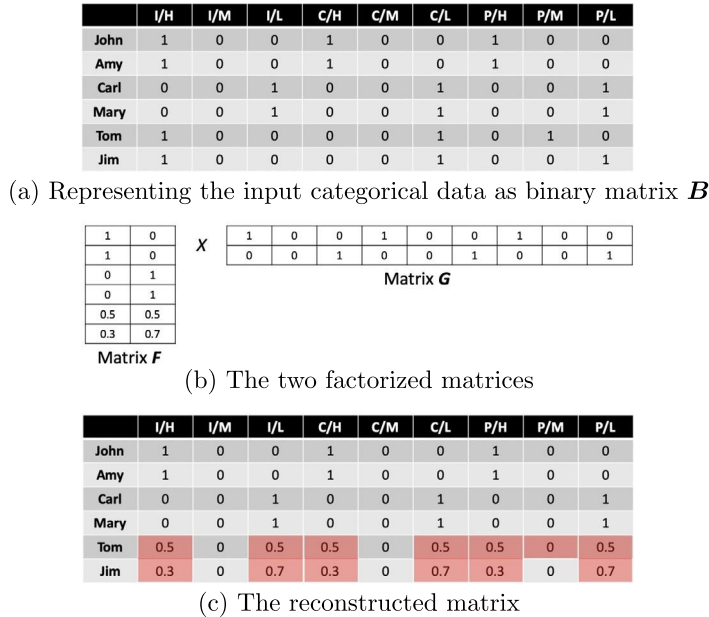


FIGURE 11.15

Using matrix factorization based method to detect outliers of input data with categorical attributes. (a) Representing the input data by a binary data matrix  $B$  by converting the categorical attributes to binary attributes. (b) The two factorized matrices  $F$  and  $G$  to approximate the binary data matrix  $B$ . “I/H,” “I/M,” and “I/L” represent the attribute values of “Income” being “High,” “Medium,” and “Low,” respectively. Likewise, “C/H,” “C/M,” and “C/L” represent the attribute values of “Credit” being “High,” “Medium,” and “Low,” respectively; and “P/H,” “P/M,” and “P/L” represent the attribute values of “Purchase” being “High,” “Medium,” and “Low,” respectively. (c) The reconstructed matrix.

When a categorical attribute has many different values, the binary data matrix  $B$  would have many columns since we need to create a separate column (i.e., binary feature) for each categorical attribute value. This might make the matrix factorization process computationally intensive and the detection results hard to interpret. For example, it might be hard to tell why “Tom” and “Jim” are flagged as outliers by just looking at the factorized matrices  $F$  and  $G$  in Fig. 11.15(b). *So, how can we do better?*

An alternative method is to use **pattern-based compression** methods to detect outliers for input data with categorical attributes.

For the input data set in Fig. 11.14, we can construct a *code table* (also known as *dictionary*), shown in Fig. 11.16. At the first glance, the table might look a bit abstract to you. Let us explain the details.

The first column of Fig. 11.16 consists of a set of *code words*, also known as a *pattern*. Each code word or pattern (e.g., “[I/H, C/H, P/H]”) consists of one or more *items*, which are essentially an attribute-value pairs (e.g., “I/H” meaning the value of “Income” is “High”). (Recall that we have used the similar notations for pattern-based classification in Chapter 7.) The second column of Fig. 11.16 contains the binary codes that are used to represent the corresponding code words, and the last column is

| Code word       | Code | Usage | Code Length |
|-----------------|------|-------|-------------|
| [I/H, C/H, P/H] | 01   | 2     | 2           |
| [I/L, C/L, P/L] | 10   | 2     | 2           |
| [I/H, C/L]      | 11   | 2     | 2           |
| [P/M]           | 001  | 1     | 3           |
| [P/L]           | 010  | 1     | 3           |

**FIGURE 11.16**

Code table for the input data in Fig. 11.14.

simply the length of each code (i.e., how many bits are used in the corresponding code).<sup>4</sup> Using the code words in the first column, we can now represent the original input data samples in an alternative, more succinct way. For example, for John and Amy, we can just use the first code word “[I/H, C/H, P/H]” to represent them respectively. Likewise, for Carl and Mary, we use the second code word “[I/L, C/L, P/L]” to represent them respectively. On the other hand, for Tom, we need two code words, including “[I/H, C/L]” and “[P/M],” to represent him; and for Jim, we also need two code words to represent him, including “[I/H, C/L]” and “[P/L].” If a given code word is *used* to represent a data sample, we say that the data sample *covers* the corresponding code word. The total number of times that a given code word is used to represent the entire data set is the *usage* of that code word (i.e., the third column of Fig. 11.16). For example, the usage of “[I/H, C/H, P/H]” is 2, since it is used to represent both John and Amy; the usage of “[P/M]” is one since it is only used to represent Tom.

Once we have the alternative representation of the input data sample, we can use the *encoding length* as an indicator of the outlier-ness. The longer the encoding length, the more likely the given data sample is an outlier. For the example in Fig. 11.14, John is represented by a single code word “[I/H, C/H, P/H]” with the encoding length of 2 (i.e., “01”). Likewise, the encoding length for Amy, Carl and Mary is also 2. On the other hand, Tom is represented by two code words, including “[I/H, C/L]” and “[P/M]” with the total encoding length of 5; Jim is also represented by two code words, including “[I/H, C/L]” and “[P/L]” with the total encoding length of 5. Therefore we flag both Tom and Jim as outlying customers. Compared with matrix-factorization methods, the pattern-compression based methods have an advantage for detecting outliers with categorical attributes in terms of interpretability. For example, by looking at the code words that Tom and Jim cover, an atypical pattern, namely “[I/H, C/L]” (high income but low credit score), stands out. Therefore the low purchase activity of Tom and Jim might be attributed to their low credit score despite that the income of both of them is high.

*But, you might wonder, how can we obtain the code table from the input data set? Why is certain code word (e.g., “[I/H, C/H, P/H]”) assigned by a shorter code than others (e.g., “[I/H, C/L],” “[P/L]”)? Why is the encoding length a reasonable indicator of outlier-ness? Pattern-based compression methods often use Minimum Description Length (MDL) principle to search for the optimal code table. The full mathematical details of MDL are outside the scope of the textbook and interested readers can refer to the bibliographic notes. Simply put, MDL principles favors the model that can most*

<sup>4</sup> It turns out that for the outlier detection task, it is the length of codes, not the actual codes, of the code words that matters with the detection results.

succinctly describe the input data set. That is, it favors a model  $M$  that minimizes the following cost:

$$L(M) + L(D|M), \quad (11.21)$$

where  $L(M)$  is the length in bits of the description of the model itself, and  $L(D|M)$  is the length in bits of the description of the data set  $D$  using the given model  $M$ .

In the outlier detection setting, the models are all possible code tables and thus  $L(M)$  is the encoding cost to describe the first two columns in Fig. 11.16, including the encoding cost of the actual code words (e.g., “[I/H, C/H, P/H]”) and the corresponding codes (e.g., “01”). Intuitively, in order to minimize  $L(M)$ , we would favor a concise code table (e.g., with a small number of short code words).  $L(D|M)$  is the total cost to describe the input data set using the given code table  $M$ , which is the summation of the encoding length of each data sample using the given code table. Intuitively, in order to minimize  $L(D|M)$ , on the one side, we should choose a comprehensive set of code words so that each input data sample can be represented by a subset of chosen code words. On the other hand, we should choose frequent code words, and if a code word is frequently *used* to represent different input data samples, we should assign it with a shorter code.<sup>5</sup> In this way, the total encoding cost  $L(D|M)$  could be minimized. Therefore if the encoding length of a given data sample is long, it suggests that it is represented by rare, infrequent patterns (code words), and thus looks outlying.

---

## 11.5 Clustering- vs. classification-based approaches

Depending on the availability of *supervision*, which is in the form of the labels of training tuples regarding whether they are normal or outlying samples, we can categorize outlier detection techniques into **clustering-based** vs. **classification-based** methods. Clustering-based approaches are unsupervised methods, which detect outliers by examining the relationship between objects and clusters. Intuitively, an outlier could be an object that belongs to a small and remote cluster or does not belong to any cluster. The notion of outliers is highly related to that of clusters. Classification-based approaches are essentially supervised methods, which treat outlier detection as a classification problem. The general idea of classification-based outlier detection methods is to train a classification model that can distinguish normal data from outliers.

### 11.5.1 Clustering-based approaches

Generally speaking, there are three approaches to clustering-based outlier detection. Consider an object.

- Does the object belong to any cluster? If not, then it is identified as an outlier.
- Is there a large distance between the object and the cluster to which it is closest? If yes, it is an outlier.
- Is the object part of a small or sparse cluster? If yes, then all the objects in that cluster are outliers.

---

<sup>5</sup> According to Shannon entropy, the optimal encoding length of a given code  $c$  is  $\log \frac{\sum_i \text{usage}(i)}{\text{usage}(c)}$ . The higher the usage of the code  $c$ , the shorter its encoding length.



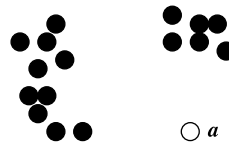


FIGURE 11.17

Object  $a$  is an outlier because it does not belong to any cluster.

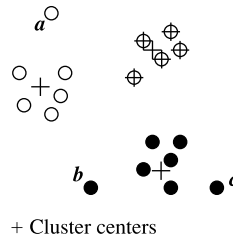


FIGURE 11.18

Outliers ( $a$ ,  $b$ ,  $c$ ) are far from the clusters to which they are closest (with respect to the cluster centers).

Let's look at examples of each of these approaches.

**Example 11.19. Detecting outliers as objects that do not belong to any cluster.** Gregarious animals (e.g., goats and deer) live and move in flocks. Using outlier detection, we can identify outliers as animals that are not part of a flock. Such animals may be either lost or wounded.

In Fig. 11.17, each point represents an animal living in a group. Using a density-based clustering method, such as DBSCAN, we note that the black points belong to clusters. The white point,  $a$ , does not belong to any cluster, and thus is declared an outlier.  $\square$

The second approach to clustering-based outlier detection considers the distance between an object and the cluster to which it is closest. If the distance is large, then the object is likely an outlier with respect to the cluster. Thus this approach detects individual outliers with respect to clusters.

**Example 11.20. Clustering-based outlier detection using distance to the closest cluster.** Using the  $k$ -means clustering method, we can partition the data points shown in Fig. 11.18 into three clusters, as shown using different symbols. The center of each cluster is marked with a +.

For each object  $o$ , we can assign an outlier-ness score according to the distance between the object and the center that is closest to the object. Suppose the closest center to  $o$  is  $c_o$ ; then the distance between  $o$  and  $c_o$  is  $dist(o, c_o)$ , and the average distance between  $c_o$  and the objects assigned to  $c_o$  is  $l_{c_o}$ . The ratio  $\frac{dist(o, c_o)}{l_{c_o}}$  measures how  $dist(o, c_o)$  stands out from the average. The larger the ratio, the farther away  $o$  is relative from the center, and the more likely  $o$  is an outlier. In Fig. 11.18, points  $a$ ,  $b$ , and  $c$  are relatively far away from their corresponding centers and thus are suspected of being outliers.  $\square$

This approach can also be used for intrusion detection, as described in Example 11.21.

**Example 11.21. Intrusion detection by clustering-based outlier detection.** A bootstrap method was developed to detect intrusions in TCP connection data by considering the similarity between data points and the clusters in a training data set. The method consists of three steps.

1. A training data set is used to find patterns of normal data. Specifically, the TCP connection data are segmented according to, say, dates. Frequent itemsets are found in each segment. The frequent itemsets that are in a majority of the segments are considered as patterns of normal data and are referred to as “base connections.”
2. Connections in the training data that contain base connections are treated as attack-free. Such connections are clustered into groups.
3. The data points in the original data set are compared with the clusters mined in step 2. Any point that is deemed an outlier with respect to the clusters is declared as a possible attack.  $\square$

Note that each of the approaches we have seen so far detects only individual objects as outliers because they compare objects one at a time against clusters in the data set. However, in a large data set, some outliers may be similar and form a small cluster. In intrusion detection, for example, hackers who use similar tactics to attack a system may form a cluster. The approaches discussed so far may be deceived by such outliers.

To overcome this problem, a third approach to cluster-based outlier detection identifies small or sparse clusters and declares the objects in those clusters to be outliers as well. An example of this approach is the *FindCBLOF* algorithm, which works as follows.

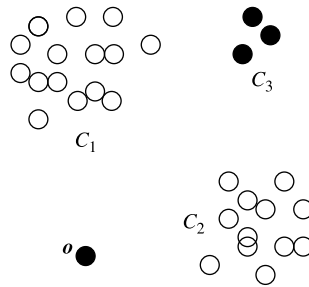
1. Find clusters in a data set and sort them according to decreasing size. The algorithm assumes that most of the data points are not outliers. It uses a parameter  $\alpha$  ( $0 \leq \alpha \leq 1$ ) to distinguish large from small clusters. Any cluster that contains at least a percentage  $\alpha$  (e.g.,  $\alpha = 30\%$ ) of the data set is considered a “large cluster.” The remaining clusters are referred to as “small clusters.”
2. To each data point, assign a *cluster-based local outlier factor* (CBLOF). For a point belonging to a large cluster, its CBLOF is the product of the cluster’s size and the similarity between the point and the cluster. For a point belonging to a small cluster, its CBLOF is calculated as the product of the size of the small cluster and the similarity between the point and the closest large cluster.

CBLOF defines the similarity between a point and a cluster in a statistical way that represents the probability that the point belongs to the cluster. The larger the value, the more similar the point and the cluster are. The CBLOF score can detect outlier points that are far from any clusters. In addition, small clusters that are far from any large cluster are considered to consist of outliers. The points with the lowest CBLOF scores are suspected outliers.

**Example 11.22. Detecting outliers in small clusters.** The data points in Fig. 11.19 form three clusters: large clusters,  $C_1$  and  $C_2$ , and a small cluster,  $C_3$ . Object  $o$  does not belong to any cluster.

Using CBLOF, *FindCBLOF* can identify  $o$  and the points in cluster  $C_3$  as outliers. For  $o$ , the closest large cluster is  $C_1$ . The CBLOF score is simply the similarity between  $o$  and  $C_1$ , which is small. For the points in  $C_3$ , the closest large cluster is  $C_2$ . Although there are three points in cluster  $C_3$ , the similarity between those points and cluster  $C_2$  is low, and  $|C_3| = 3$  is small; thus, the CBLOF scores of points in  $C_3$  are small.  $\square$

Clustering-based approaches may incur high computational costs if they have to find clusters before detecting outliers. Several techniques have been developed for improved efficiency. For example, **fixed-width clustering** is a linear-time technique that is used in some outlier detection methods. The idea is




---

**FIGURE 11.19**

Outliers in small clusters.

simple yet efficient. A point is assigned to a cluster if the center of the cluster is within a predefined distance threshold from the point. If a point cannot be assigned to any existing cluster, a new cluster is created. The distance threshold may be learned from the training data under certain conditions.

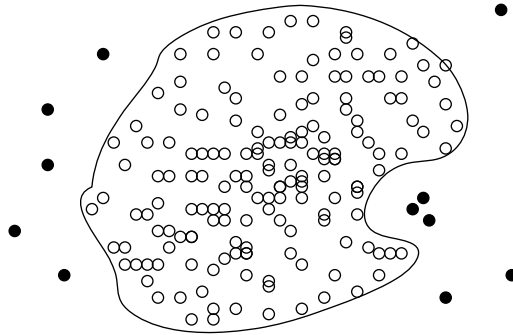
Clustering-based outlier detection methods have the following advantages. First, they can detect outliers without requiring any labeled data, that is, in an unsupervised way. They work for many data types. Clusters can be regarded as summaries of the data. Once the clusters are obtained, clustering-based methods need only compare any object against the clusters to determine whether the object is an outlier. This process is typically fast because the number of clusters is usually small compared to the total number of objects.

A weakness of clustering-based outlier detection is its effectiveness, which highly depends on the clustering method used. Such methods may not be optimized for outlier detection. Clustering methods are often costly for large data sets, which could become a bottleneck.

### 11.5.2 Classification-based approaches

Let us consider a training set that contains samples labeled as “normal” and others labeled as “outlier.” A classifier can then be constructed based on the training set. Any classification method can be used (Chapters 6 and 7). This kind of brute-force approach, however, does not work well for outlier detection because the training set is typically heavily biased. That is, the number of normal samples likely far exceeds the number of outlier samples. This imbalance, where the number of outlier samples may be insufficient, can prevent us from building an accurate classifier. Consider intrusion detection in a system, for example. Because most system accesses are normal, it is easy to obtain a good representation of the normal events. However, it is infeasible to enumerate all potential intrusions, as new and unexpected attempts occur from time to time. Hence, we are left with an insufficient representation of the outlier (or intrusion) samples.

To overcome this challenge, classification-based outlier detection methods often use a *one-class model*. That is, a classifier is built to describe only the normal class. Any samples that do not belong to the normal class are regarded as outliers. For example, in **one-class SVM**, it seeks to find a max-margin hyperplane in the transformed high-dimensional space (called reproducing kernel Hilbert space or RKHS for short) that separates the normal data tuples and the origin that represents the outlier. In



**FIGURE 11.20**

Learning a model for the normal class.

**Support Vector Data Description (SVDD)**, it seeks to find a minimum hypersphere that contains all the normal tuples.

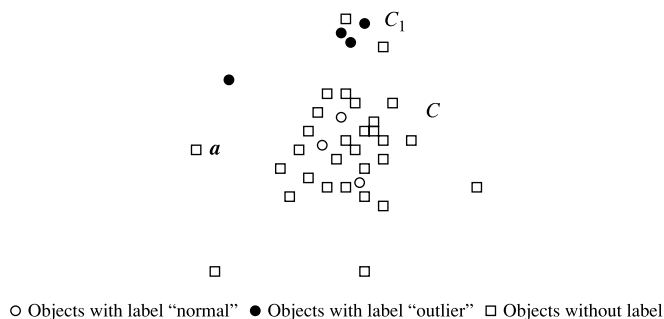
**Example 11.23. Outlier detection using a one-class model.** Consider the training set shown in Fig. 11.20, where white points are samples labeled as “normal” and black points are samples labeled as “outlier.” To build a model for outlier detection, we can learn the decision boundary of the normal class using classification methods such as SVM (Chapter 7), as illustrated. Given a new object, if the object is within the decision boundary of the normal class, it is treated as a normal case. If the object is outside the decision boundary, it is declared an outlier.

An advantage of using only the model of the normal class to detect outliers is that the model can detect new outliers that may not appear close to any outlier objects in the training set. This occurs as long as such new outliers fall outside the decision boundary of the normal class. □

The idea of using the decision boundary of the normal class can be extended to handle situations where the normal objects may belong to multiple classes. For example, an electronics store accepts returned items. Customers can return items for a number of reasons (corresponding to class categories) such as “product design defects” and “product damaged during shipment.” Each such class is regarded as normal. To detect outlier cases, the store can learn a model for each normal class. To determine whether a case is an outlier, we can run each model on the case. If the case does not fit any of the models, then it is declared an outlier.

Classification-based methods and clustering-based methods can be further combined to detect outliers in a semisupervised learning way.

**Example 11.24. Outlier detection by semisupervised learning.** Consider Fig. 11.21, where objects are labeled as either “normal” or “outlier” or have no label at all. Using a clustering-based approach, we find a large cluster,  $C$ , and a small cluster,  $C_1$ . Because some objects in  $C$  carry the label “normal,” we can treat all objects in this cluster (including those without labels) as normal objects. We use the one-class model of this cluster to identify normal objects in outlier detection. Similarly, because some objects in cluster  $C_1$  carry the label “outlier,” we declare all objects in  $C_1$  as outliers. Any object that does not fall into the model for  $C$  (e.g.,  $a$ ) is considered an outlier as well. □




---

**FIGURE 11.21**

Detecting outliers by semisupervised learning.

Classification-based methods can incorporate human domain knowledge into the detection process by learning from the labeled samples. Once the classification model is constructed, the outlier detection process is fast. It only needs to compare the objects to be examined against the model learned from the training data. The quality of classification-based methods heavily depends on the availability and quality of the training set. In many applications, it is difficult to obtain representative and high-quality training data, which limits the applicability of classification-based methods. A promising solution is to combine outlier detection and active learning (introduced in Chapter 7) to obtain a few labeled training tuples for each class (e.g., normal class, outlying class), with the help of a human annotator.

---

## 11.6 Mining contextual and collective outliers

An object in a given data set is a **contextual outlier** (or *conditional outlier*) if it deviates significantly with respect to a specific context of the object (Section 11.1). The context is defined using **contextual attributes**. These depend heavily on the application and are often provided by users as part of the contextual outlier detection task. Contextual attributes can include spatial attributes, time, network locations, and sophisticated structured attributes. In addition, **behavioral attributes** define characteristics of the object and are used to evaluate whether the object is an outlier in the context to which it belongs.

**Example 11.25. Contextual outliers.** To determine whether the temperature of a location is exceptional (i.e., an outlier), the attributes specifying information about the location can serve as contextual attributes. These attributes may be spatial attributes (e.g., longitude and latitude) or location attributes in a graph or network. The attribute *time* can also be used. In customer-relationship management, whether a customer is an outlier may depend on other customers with similar profiles. Here, the attributes defining customer profiles provide the context for outlier detection. □

In comparison to outlier detection in general, identifying contextual outliers requires analyzing the corresponding contextual information. Contextual outlier detection methods can be divided into two categories according to whether the contexts can be clearly identified.

### 11.6.1 Transforming contextual outlier detection to conventional outlier detection

This category of methods is for situations where the contexts can be clearly identified. The idea is to transform the contextual outlier detection problem into a typical outlier detection problem. Specifically, for a given data object, we can evaluate whether the object is an outlier in two steps. In the first step, we identify the context of the object using the contextual attributes. In the second step, we calculate the outlier-ness score for the object in the context using a conventional outlier detection method.

**Example 11.26. Contextual outlier detection when the context can be clearly identified.** In customer-relationship management, we can detect outlier customers in the context of customer groups. Suppose an electronics store maintains customer information on four attributes, namely *age\_group* (i.e., under 25, 25–45, 45–65, and over 65), *postal\_code*, *number\_of\_transactions\_per\_year*, and *annual\_total\_transaction\_amount*. The attributes *age\_group* and *postal\_code* serve as contextual attributes, and the attributes *number\_of\_transactions\_per\_year* and *annual\_total\_transaction\_amount* are behavioral attributes.  $\square$

To detect contextual outliers in this setting, for a customer,  $c$ , we can first locate the context of  $c$  using the attributes *age\_group* and *postal\_code*. We can then compare  $c$  with the other customers in the same group and use a conventional outlier detection method, such as some of the ones discussed earlier, to determine whether  $c$  is an outlier.

Contexts may be specified at different levels of granularity. Suppose the same electronics store also maintains customer information at a more detailed level for the attributes *age*, *postal\_code*, *number\_of\_transactions\_per\_year*, and *annual\_total\_transaction\_amount*. We can still group customers on *age* and *postal\_code* and then mine outliers in each group. What if the number of customers falling into a group is very small or even zero? For a customer,  $c$ , if the corresponding context contains very few or even no other customers, the evaluation of whether  $c$  is an outlier using the exact context is unreliable or even impossible.

To overcome this challenge, we can assume that customers of similar age and who live within the same area should have similar normal behavior. This assumption can help to generalize contexts and makes for more effective outlier detection. For example, using a set of training data, we may learn a mixture model,  $U$ , of the data on the contextual attributes, and another mixture model,  $V$ , of the data on the behavior attributes. A mapping  $p(V_i|U_j)$  is also learned to capture the probability that a data object  $o$  belonging to cluster  $U_j$  on the contextual attributes is generated by cluster  $V_i$  on the behavior attributes. The outlier-ness score can then be calculated as

$$S(o) = \sum_{U_j} p(o \in U_j) \sum_{V_i} p(o \in V_i) p(V_i|U_j). \quad (11.22)$$

Thus the contextual outlier problem is transformed into outlier detection using mixture models.

### 11.6.2 Modeling normal behavior with respect to contexts

In some applications, it is inconvenient or infeasible to clearly partition the data into contexts. For example, consider the situation where an online store records customer browsing behavior in a search log. For each customer, the data log contains the sequence of products searched for and browsed by the customer. The store is interested in contextual outlier behavior, such as if a customer suddenly purchased

a product that is unrelated to those she recently browsed. However, in this application, contexts cannot be easily specified because it is unclear how many products browsed earlier should be considered as the context, and this number will likely differ for each product.

This second category of contextual outlier detection methods models the normal behavior with respect to contexts. Using a training data set, such a method trains a model that predicts the expected behavior attribute values with respect to the contextual attribute values. To determine whether a data object is a contextual outlier, we can then apply the model to the contextual attributes of the object. If the behavior attribute values of the object significantly deviate from the values predicted by the model, then the object can be declared a contextual outlier.

By using a prediction model that links the contexts and behavior, these methods avoid the explicit identification of specific contexts. A number of classification and prediction techniques can be used to build such models such as regression, Markov models, and finite state automaton. Interested readers are referred to Chapters 6 and 7 on classification and the bibliographic notes for further details (Section 11.10).

In summary, contextual outlier detection enhances conventional outlier detection by considering contexts, which are important in many applications. We may be able to detect outliers that cannot be detected otherwise. Consider a credit card user whose income level is low but whose expenditure patterns are similar to those of millionaires. This user can be detected as a contextual outlier if the income level is used to define context. Such a user may not be detected as an outlier without contextual information because she does share expenditure patterns with many millionaires. Considering contexts in outlier detection can also help to avoid false alarms. Without considering the context, a millionaire's purchase transaction may be falsely detected as an outlier if the majority of customers in the training set are not millionaires. This can be corrected by incorporating contextual information in outlier detection.

### 11.6.3 Mining collective outliers

A group of data objects forms a **collective outlier** if the objects as a whole deviate significantly from the entire data set, even though each individual object in the group may not be an outlier (Section 11.1). To detect collective outliers, we have to examine the *structure* of the data set, that is, the relationships between multiple data objects. This makes the problem more difficult than conventional and contextual outlier detection.

*“How can we explore the data set structure?”* This typically depends on the nature of the data. For outlier detection in temporal data (e.g., time series and sequences), we explore the structures formed by time, which occur in segments of the time series or subsequences. To detect collective outliers in spatial data, we explore local areas. Similarly, in graph and network data, we explore subgraphs. Each of these structures is inherent to its respective data type.

Contextual outlier detection and collective outlier detection are similar in that they both explore structures. In contextual outlier detection, the structures are the contexts, as specified by the contextual attributes explicitly. The critical difference in collective outlier detection is that the structures are often not explicitly defined and have to be discovered as part of the outlier detection process.

As with contextual outlier detection, collective outlier detection methods can also be divided into two categories. The first category consists of methods that reduce the problem to conventional outlier detection. Its strategy is to identify *structure units*, treat each structure unit (e.g., a subsequence, a time-series segment, a local area, or a subgraph) as a data object, and extract features. The problem of

collective outlier detection is thus transformed into outlier detection on the set of “structured objects” constructed as such using the extracted features. A structure unit, which represents a group of objects in the original data set, is a collective outlier if the structure unit deviates significantly from the expected trend in the space of the extracted features.

**Example 11.27. Collective outlier detection on graph data.** Let’s see how we can detect collective outliers in a store’s online social network of customers. Suppose we treat the social network as an unlabeled graph. We then treat each possible subgraph of the network as a structure unit. For each subgraph,  $S$ , let  $|S|$  be the number of vertices in  $S$ , and  $freq(S)$  be the frequency of  $S$  in the network. That is,  $freq(S)$  is the number of different subgraphs in the network that are isomorphic to  $S$ . We can use these two features to detect outlier subgraphs. An *outlier subgraph* is a collective outlier that contains multiple vertices.

In general, a small subgraph (e.g., a single vertex or a pair of vertices connected by an edge or a dense subgraph) is expected to be frequent, and a large subgraph is expected to be infrequent. Using the preceding simple method, we can detect small subgraphs that are of very low frequency or large subgraphs that are surprisingly frequent. These are outlier structures in the social network.  $\square$

Predefining the structure units for collective outlier detection can be difficult or impossible. Consequently, the second category of methods models the expected behavior of structure units directly. For example, to detect collective outliers in temporal sequences, one method is to learn a Markov model from the sequences. A subsequence can then be declared as a collective outlier if it significantly deviates from the model.

In summary, collective outlier detection is subtle due to the challenge of exploring the structures in data. The exploration typically uses heuristics and thus may be application-dependent. The computational cost is often high due to the sophisticated mining process. While highly useful in practice, collective outlier detection remains a challenging direction that calls for further research and development.

---

## 11.7 Outlier detection in high-dimensional data

In some applications, we may need to detect outliers in high-dimensional data. The curse of dimensionality poses huge challenges for effective outlier detection. As the dimensionality increases, the distance between objects may be heavily dominated by noise. That is, the distance and similarity between two points in a high-dimensional space may not reflect the real relationship between the points. Consequently, conventional outlier detection methods, which mainly use proximity or density to identify outliers, deteriorate as dimensionality increases.

Ideally, outlier detection methods for high-dimensional data should meet the challenges that follow.

- **Interpretation of outliers:** They should be able to not only detect outliers but also provide an interpretation of the outliers. Because many features (or dimensions) are involved in a high-dimensional data set, detecting outliers without providing any interpretation as to why they are outliers is not very useful. The interpretation of outliers may come from, for example, specific subspaces that manifest the outliers or an assessment regarding the “outlier-ness” of the objects. Such interpretation can help users understand the possible meaning and significance of the outliers.



- **Data sparsity:** The methods should be capable of handling sparsity in high-dimensional spaces. The distance between objects becomes heavily dominated by noise as the dimensionality increases. Therefore data in high-dimensional spaces are often sparse.
- **Data subspaces:** They should model outliers appropriately, for example, adaptive to the subspaces signifying the outliers and capturing the local behavior of data. Using a fixed-distance threshold against all subspaces to detect outliers is not a good idea because the distance between two objects monotonically increases as the dimensionality increases.
- **Scalability with respect to dimensionality:** As the dimensionality increases, the number of subspaces increases exponentially. An exhaustive combinatorial exploration of the search space, which contains all possible subspaces, is not a scalable choice.

Outlier detection methods for high-dimensional data can be divided into the following approaches. These include extending conventional outlier detection (Section 11.7.1), finding outliers in subspaces (Section 11.7.2), outlier detection ensemble (Section 11.7.3), deep learning-based approaches (Section 11.7.4) and modeling high-dimensional outliers (Section 11.7.5).

### 11.7.1 Extending conventional outlier detection

One approach for outlier detection in high-dimensional data extends conventional outlier detection methods. It uses the conventional proximity-based models of outliers. However, to overcome the deterioration of proximity measures in high-dimensional spaces, it uses alternative measures or constructs subspaces and detects outliers there.

The **HilOut** algorithm is an example of this approach. HilOut finds distance-based outliers, but uses the ranks of distance instead of the absolute distance in outlier detection. Specifically, for each object,  $\mathbf{o}$ , HilOut finds the  $k$ -nearest neighbors of  $\mathbf{o}$ , denoted by  $nn_1(\mathbf{o}), \dots, nn_k(\mathbf{o})$ , where  $k$  is an application-dependent parameter. The weight of object  $\mathbf{o}$  is defined as

$$w(\mathbf{o}) = \sum_{i=1}^k dist(\mathbf{o}, nn_i(\mathbf{o})). \quad (11.23)$$

All objects are ranked in weight-descending order. The top- $l$  objects in weight are output as outliers, where  $l$  is another user-specified parameter.

Computing the  $k$ -nearest neighbors for every object is costly and does not scale up when the dimensionality is high and the database is large. To address the scalability issue, HilOut employs space-filling curves to achieve an approximation algorithm, which is scalable in both running time and space with respect to database size and dimensionality.

While some methods like HilOut detect outliers in the full space despite the high dimensionality, other methods reduce the high-dimensional outlier detection problem to a lower-dimensional one by dimensionality reduction (Chapter 2). The idea is to reduce the high-dimensional space to a lower-dimensional space where normal instances can still be distinguished from outliers. If such a lower-dimensional space can be found, then conventional outlier detection methods can be applied. In principle, the matrix factorization based approaches introduced in Section 11.4 can be used to find such lower-dimensional space. For example, the rows of the right matrix  $\mathbf{G}$  in Fig. 11.10 provide the representation of the input data tuples in the lower-dimensional space.

To reduce the dimensionality, general feature selection and extraction methods may be used or extended for outlier detection. For example, principal components analysis (PCA) can be used to extract a lower-dimensional space. Heuristically, the principal components with *low* variance are preferred because, on such dimensions, normal objects are likely close to each other and outliers often deviate from the majority. Notice that this is different from the case when PCA is used as a general-purposed dimensionality reduction tool, where the principal components with *high* variance are favored.

By extending conventional outlier detection methods, we can reuse much of the experience gained from research in the field. These new methods, however, are limited. First, they cannot detect outliers with respect to subspaces and thus have limited interpretability. Second, dimensionality reduction is feasible only if there exists a lower-dimensional space where normal objects and outliers are well separated. This assumption may not hold true.

### 11.7.2 Finding outliers in subspaces

Another approach for outlier detection in high-dimensional data is to search for outliers in various subspaces. A unique advantage is that, if an object is found to be an outlier in a subspace of much lower dimensionality, the subspace provides critical information for interpreting *why* and *to what extent* the object is an outlier. This insight is highly valuable in applications with high-dimensional data due to the overwhelming number of dimensions.

**Example 11.28. Outliers in subspaces.** As a customer-relationship manager at an electronics store, you are interested in finding outlier customers. The store maintains an extensive customer information database, which contains many attributes and the transaction history of customers. The database is high dimensional.

Suppose you find that a customer, Alice, is an outlier in a lower-dimensional subspace that contains the dimensions *average\_transaction\_amount* and *purchase\_frequency*, such that her average transaction amount is substantially larger than the majority of the customers, and her purchase frequency is dramatically lower. The subspace itself speaks for why and to what extent Alice is an outlier. Using this information, you strategically decide to approach Alice by suggesting options that could improve her purchase frequency at the store.  $\square$

“How can we detect outliers in subspaces?” We use a *grid-based subspace outlier detection method* to illustrate. The major ideas are as follows. We consider projections of the data onto various subspaces. If, in a subspace, we find an area that has a density that is much lower than average, then the area may contain outliers. To find such projections, we first discretize the data into a grid in an equal-depth way. That is, each dimension is partitioned into  $\phi$  equal-depth ranges, where each range contains a fraction,  $f$ , of the objects ( $f = \frac{1}{\phi}$ ). Equal-depth partitioning is used because data along different dimensions may have different localities. An equal-width partitioning of the space may not be able to reflect such differences in locality.

Next, we search for regions defined by ranges in subspaces that are significantly sparse. To quantify what we mean by “significantly sparse,” let’s consider a  $k$ -dimensional cube formed by  $k$  ranges on  $k$  dimensions. Suppose the data set contains  $n$  objects. If the objects are independently distributed, the expected number of objects falling into a  $k$ -dimensional region is  $\left(\frac{1}{\phi}\right)^k n = f^k n$ . The standard deviation of the number of points in a  $k$ -dimensional region is  $\sqrt{f^k(1 - f^k)n}$ . Suppose a specific  $k$ -dimensional

cube  $C$  has  $n(C)$  objects. We can define the **sparsity coefficient** of  $C$  as

$$S(C) = \frac{n(C) - f^k n}{\sqrt{f^k(1 - f^k)n}}. \quad (11.24)$$

If  $S(C) < 0$ , then  $C$  contains fewer objects than expected. The smaller the value of  $S(C)$  (i.e., the more negative), the sparser  $C$  is and the more likely the objects in  $C$  are outliers in the subspace.

By assuming  $S(C)$  follows a normal distribution, we can use normal distribution tables to determine the significance level for an object that deviates dramatically from the average for a priori assumption of the data following a uniform distribution. In general, the assumption of uniform distribution does not hold. However, the sparsity coefficient still provides an intuitive measure of the “outlier-ness” of a region.

To find cubes of significantly small sparsity coefficient values, a brute-force approach is to search every cube in every possible subspace. The cost of this, however, is immediately exponential. An *evolutionary search* can be conducted, which improves efficiency at the expense of accuracy. For details, please refer to the bibliographic notes (Section 11.10). The objects contained by cubes of very small sparsity coefficient values are output as outliers. In addition, *genetic algorithm* has been found to be effective in searching for desirable cubes.

In summary, searching for outliers in subspaces is advantageous in that the outliers found tend to be better understood, owing to the context provided by the subspaces.<sup>6</sup> Challenges include making the search efficient and scalable.

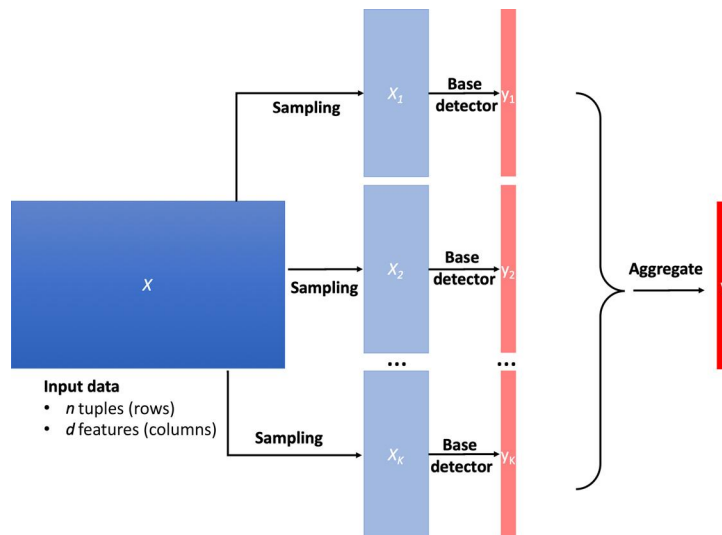
### 11.7.3 Outlier detection ensemble

Another effective way to detect outliers in high-dimensional data is via *ensemble*. Its general procedure is as follows and a pictorial illustration is given in Fig. 11.22. Given an input data set  $\mathbf{X}$  (the left of Fig. 11.22), with  $n$  data tuples (rows) in  $d$ -dimensional space (columns), the ensemble methods first create  $K$  base detectors (the middle of Fig. 11.22). For each base detector, we first represent the input data set in a *random subspace*  $\mathbf{X}_i$  ( $i = 1, \dots, K$ ) whose dimensionality  $d'$  is much smaller than the original feature dimensionality (i.e.,  $d' \ll d$ ), and then use an off-the-shelf outlier detection method (e.g., LOF) to assign each data tuple with an outlier-ness score. We represent the outlier-ness scores of all  $n$  data tuples in the form of a vector  $\mathbf{y}_i$  of length  $n$  ( $i = 1, \dots, K$ ). After that, we aggregate the detection results from the  $K$  base detectors to obtain the overall outlier-ness scores for all input data tuples in the form of another vector  $\mathbf{y}$  of length  $n$  (the right of Fig. 11.22). Finally, we flag data tuples with the highest overall outlier-ness scores as outliers.

There are two key issues in outlier detection ensemble, including (1) how to represent the input data in a random subspace and (2) how to aggregate the detection results of base detectors. For (1), there are two commonly used methods, including *feature bagging* and *rotated bagging*. For feature bagging, we randomly select a few original features to form the representation of the input data in a random subspace; and for rotated bagging, we first generate a random subspace of dimensionality  $d'$  ( $d' \ll d$ ) and then project the input data into this subspace. Conceptually, if we represent the input

---

<sup>6</sup> For this reason, we can see that subspace-based outlier detection and contextual outlier detection are closely related with each other.



**FIGURE 11.22**

A pictorial illustration of using ensemble methods to detect outliers in high-dimensional data. (Left) The input data set with  $n$  data tuples (rows) in  $d$ -dimensional space (columns). (Middle)  $K$  base detectors. (Right) The aggregated detection results.

data set as an  $n \times d$  data matrix  $\mathbf{X}$ , feature bagging selects a few  $d'$  ( $d' \ll d$ ) actual columns of  $\mathbf{X}$  as the representation of the input data set in a random subspace, whereas rotated bagging first generates  $d'$  mutually orthonormal vectors and then projects the input data matrix  $\mathbf{X}$  onto the subspace spanned by these  $d'$  vectors.

In order to aggregate the detection results from the base detectors, we can either use *mean aggregation* where the overall outlier-ness score of a given data tuple is the average of its outlier-ness scores from  $K$  base detectors, or *max aggregation* where the overall outlier-ness score of a given data tuple is the maximum of its outlier-ness scores from  $K$  base detectors. For both aggregation methods, it is important to normalize (e.g., min-max normalization, z-score normalization) the outlier-ness scores of the base detectors *before* aggregation, so that the outlier-ness score of a base detector will not overwhelm the scores of other base detectors.

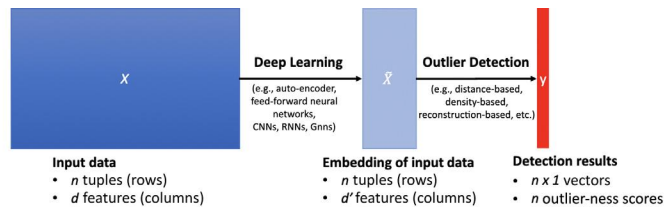
### 11.7.4 Taming high dimensionality by deep learning

In the context of outlier detection in high-dimensional data, deep learning–based approaches offer two appealing advantages. First, deep learning methods (e.g., autoencoder, feed-forward neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), graphical neural networks (GNNs)) produce vector representation (i.e., *embedding*) of the input data tuples with a much smaller number of dimensions than the original ones and thus naturally alleviate the high-dimensionality challenge. Second, owing to deep learning’s strong ability to learning semantically meaningful representation, the produced embedding often captures the complicate (e.g., nonlinear) interaction between

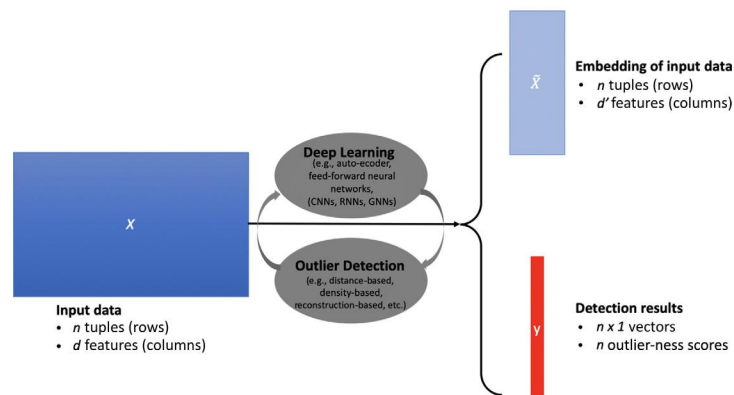
different input features and thus is capable of detecting outliers that might be overlooked by the alternative methods (e.g., linear matrix factorization–based approaches).

Generally speaking, there are two basic strategies to leverage deep learning for high-dimensional outlier detection, which we will introduce next. Fig. 11.23 presents an illustration.

The first strategy uses the deep learning methods as a *preprocessing* step. For example, if we feed the input data set into an autoencoder, the output from the encoder produces the embedding of the input data tuples in a much lower-dimensional space. Then, with such embedding as the input, we can use outlier detection methods (e.g., proximity-based approaches, reconstruction-based approaches) to detect outlying tuples. The advantage of this type of strategy lies in its simplicity. In principle, we can use a variety of deep learning models to produce embedding of the input data in a lower-dimensional space, such as feed-forward neural networks for spatial data, convolutional neural networks for grid data, recurrent neural networks for sequential data, and graph neural networks for graph data. The learned embedding can in turn be fed into a variety of off-the-shelf outlier detection methods, without the need of retraining or modifications. However, this strategy separates the embedding learning stage and outlier detection stage and thus might lead to suboptimal detection results. For example, the outliers in the learned embedding space might be mixed with that of the normal tuples and thus are missed by the outlier detection methods.



(a) Deep learning as a pre-processing step



(b) Integrating deep learning with outlier detection

**FIGURE 11.23**

An illustration of two basic strategies of using deep learning for high-dimensional outlier detection.

The second strategy aims to integrate the embedding learning and outlier detection together. In other words, it tries to find embedding of the input data tuples that is tailored for spotting the outliers and simultaneously produce the embedding and detection results. Compared with the first strategy, it often leads to a higher detection accuracy. The general idea of methods using this strategy is to replace certain component of an existing outlier detection method by a deep learning model. Let us take one-class SVM as an example. Recall that in the traditional one-class SVM (Section 11.5.2), it seeks to find a max-margin hyperplane in the transformed high-dimensional space (i.e., RKHS) that separates the normal data tuples from the origin which represents the outlier. To this end, it solves an optimization problem that involves a component of  $w^T \phi(x)$  representing the output of the hyperplane, where  $w$  is the weight vector and  $\phi(x)$  in the feature vector of the input data  $x$  in the RKHS. In the **One-Class Neural Network (OC-NN)**, it replaces the hyperplane output (i.e.,  $w^T \phi(x)$ ) by a feed-forward neural network, whose output layer produces an outlier-ness score, and the last hidden layer produces the embedding of the input data. By combining the embedding learning and outlier detection together, it was found that **OC-NN** leads to better performance than the traditional one-class SVM.

Another example is **Deviation Networks (DevNet)**. Recall that for univariate outlier detection (Section 11.2.1), a simple yet effective outlier detection method is Grubb's test based on z-score:  $z = \frac{|x-\mu|}{\sigma}$ , where  $x$  is the input feature value, and  $\mu$  and  $\sigma$  are the sample mean and the sample standard deviation, respectively. For high-dimensional data, **DevNet** replaces the raw feature value  $x$  by its outlier-ness score  $f(x, \Theta)$ . The outlier-ness score  $f(x, \Theta)$  is produced by a deep learning model (e.g., a feed-forward neural network with a linear output layer) with  $x$  as the input and  $\Theta$  being the model parameters of the deep learning model. In addition, **DevNet** replaces  $\mu$  and  $\sigma$  by the sample mean and the sample standard deviation of the outlier-ness scores of the normal tuples, respectively.<sup>7</sup> DevNet was found to obtain significant detection accuracy improvement over alternative methods.

### 11.7.5 Modeling high-dimensional outliers

Another alternative approach for outlier detection methods in high-dimensional data tries to develop new models for high-dimensional outliers directly. Such models typically avoid proximity measures and instead adopt new heuristics to detect outliers, which do not deteriorate in high-dimensional data.

Let's examine *angle-based outlier detection (ABOD)* as an example.

**Example 11.29. Angle-based outliers.** Fig. 11.24 contains a set of points forming a cluster, with the exception of  $c$ , which is an outlier. For each point  $o$ , we examine the angle  $\angle xoy$  for every pair of points  $x, y$  such that  $x \neq o, y \neq o$ . The figure shows angle  $\angle dae$  as an example.

Note that for a point in the center of a cluster (e.g.,  $a$ ), the angles formed as such differ widely. For a point that is at the border of a cluster (e.g.,  $b$ ), the angle variation is smaller. For a point that is an outlier (e.g.,  $c$ ), the angle variable is substantially smaller. This observation suggests that we can use the variance of angles for a point to determine whether a point is an outlier.  $\square$

We can combine angles and distance to model outliers. Mathematically, for each point  $o$ , we use the distance-weighted angle variance as the outlier-ness score. That is, given a set of points,  $D$ , for a point,

<sup>7</sup> Another subtle change DevNet made is to remove the absolute value sign in the z-score definition. In other words, it only looks at the upper-tail of the outlier-ness scores produced by the deep learning model. The larger the outlier-ness score, the more likely the given tuple is an outlier.

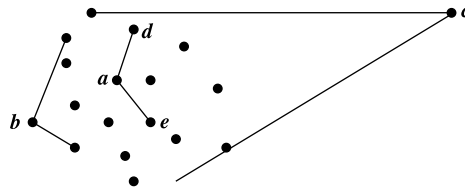


FIGURE 11.24

Angle-based outliers.

$o \in D$ , we define the **angle-based outlier factor** (ABOF) as

$$ABOF(o) = \text{VAR}_{x,y \in D, x \neq o, y \neq o} \frac{\langle \vec{ox}, \vec{oy} \rangle}{\text{dist}(o, x)^2 \text{dist}(o, y)^2}, \quad (11.25)$$

where  $\langle \cdot, \cdot \rangle$  is the scalar product (i.e., dot product) operator, VAR is the variance, and  $\text{dist}(\cdot)$  is a norm distance.

Clearly, the farther away a point is from clusters and the smaller the variance of the angles of a point, the smaller the ABOF. The ABOD computes the ABOF for each point and outputs a list of the points in the data set in ABOF-ascending order.

Computing the exact ABOF for every point in a database is costly, requiring a time complexity of  $O(n^3)$ , where  $n$  is the number of points in the database. Obviously, this exact algorithm does not scale up for large data sets. Approximation methods have been developed to speed up the computation. The angle-based outlier detection idea has been generalized to handle arbitrary data types. For additional details, see the bibliographic notes (Section 11.10).

Developing native models for high-dimensional outliers can lead to effective methods. However, finding good heuristics for detecting high-dimensional outliers is difficult. Efficiency and scalability on large and high-dimensional data sets are major challenges.

## 11.8 Summary

- Assume that a given statistical process is used to generate a set of data objects. An **outlier** is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism.
- **Types of outliers** include global outliers, contextual outliers, and collective outliers. An object may be more than one type of outlier.
- **Global outliers** are the simplest form of outlier and the easiest to detect. A **contextual outlier** deviates significantly with respect to a specific context of the object (e.g., a Toronto temperature value of 28°C is an outlier if it occurs in the context of winter). A subset of data objects forms a **collective outlier** if the objects as a whole deviate significantly from the entire data set, even though the individual data objects may not be outliers. Collective outlier detection requires background information to model the relationships among objects to find outlier groups.

- **Challenges** in outlier detection include finding appropriate data models, the dependence of outlier detection systems on the application involved, finding ways to distinguish outliers from noise, and providing justification for identifying outliers as such.
- Outlier detection methods can be **categorized** according to whether the sample of data for analysis is given with expert-provided labels that can be used to build an outlier detection model. In this case, the detection methods are *supervised*, *semisupervised*, or *unsupervised*. Alternatively, outlier detection methods may be organized according to their assumptions regarding normal objects vs. outliers. This categorization includes *statistical* methods, *proximity-based* methods, and *reconstruction-based* methods.
- **Statistical outlier detection methods** (or **model-based methods**) assume that the normal data objects follow a statistical model, where data not following the model are considered outliers. Such methods may be *parametric* (they assume that the data are generated by a parametric distribution) or *nonparametric* (they learn a model for the data rather than assuming one a priori). Parametric methods for multivariate data may employ the Mahalanobis distance, the  $\chi^2$  statistic, or a mixture of multiple parametric models. Histograms and kernel density estimation are examples of nonparametric methods.
- **Proximity-based outlier detection methods** assume that an object is an outlier if the proximity of the object to its nearest neighbors significantly deviates from the proximity of most of the other objects to their neighbors in the same data set. *Distance-based outlier detection methods* consult the *neighborhood* of an object, defined by a given radius. An object is an outlier if its neighborhood does not have enough other points. In *density-based outlier detection methods*, an object is an outlier if its density is relatively much lower than that of its neighbors.
- **Clustering-based outlier detection methods** assume that the normal data objects belong to large and dense clusters, whereas outliers belong to small or sparse clusters or do not belong to any clusters.
- **Classification-based outlier detection methods** often use a one-class model. That is, a classifier is built to describe only the normal class. Any samples that do not belong to the normal class are regarded as outliers.
- **Contextual outlier detection** and **collective outlier detection** explore structures in the data. In contextual outlier detection, the structures are defined as contexts using contextual attributes. In collective outlier detection, the structures are implicit and are explored as part of the mining process. To detect such outliers, one approach transforms the problem into one of conventional outlier detection. Another approach models the structures directly.
- **Outlier detection methods for high-dimensional data** can be divided into five main approaches, including extending conventional outlier detection, finding outliers in subspaces, outlier detection ensemble, deep learning-based approaches, and modeling high-dimensional outliers.

---

## 11.9 Exercises

- 11.1. Give an application example where global outliers, contextual outliers, and collective outliers are all interesting. What are the attributes, and what are the contextual and behavioral attributes? How is the relationship among objects modeled in collective outlier detection?



- 11.2. Give an application example of where the border between normal objects and outliers is often unclear, so that the degree to which an object is an outlier has to be well estimated.
- 11.3. Adapt a simple semisupervised method for outlier detection. Discuss the scenario where you have (a) only some labeled examples of normal objects and (b) only some labeled examples of outliers.
- 11.4. Using an equal-depth histogram, design a way to assign an object an outlier-ness score.
- 11.5. Consider the nested loop approach to mining distance-based outliers (Fig. 11.6). Suppose the objects in a data set are arranged randomly; that is, each object has the same probability to appear in a position. Show that when the number of outlier objects is small with respect to the total number of objects in the whole data set, the expected number of distance calculations is linear to the number of objects.
- 11.6. In the density-based outlier detection method of Section 11.3.2, the definition of local reachability density has a potential problem:  $lrd_k(o) = \infty$  may occur. Explain why this may occur and propose a fix to the issue.
- 11.7. Because clusters may form a hierarchy, outliers may belong to different granularity levels. Propose a clustering-based outlier detection method that can find outliers at different levels.
- 11.8. In outlier detection by semisupervised learning, what is the advantage of using objects without labels in the training data set?
- 11.9. Given a user-community bipartite graph, where the nodes are users and communities, and links indicate the membership between users and communities. We can represent this bipartite graph by its adjacency matrix  $\mathbf{A}$ , where  $\mathbf{A}(i, j) = 1$  means user  $i$  belongs to community  $j$ ; and  $\mathbf{A}(i, j) = 0$  otherwise. We further approximate the adjacency matrix  $\mathbf{A}$  by the multiplication of two low-rank matrices, that is,  $\mathbf{A} \approx \mathbf{FG}$ , where  $\mathbf{F}$  and  $\mathbf{G}$  are two low-rank matrices. Describe how you can leverage the above low-rank approximation result to detect (a) outlying users and (b) outlying user-community memberships, respectively.
- 11.10. Describe a method to integrate support vector data description (SVDD) and deep learning for outlier detection.
- 11.11. To understand why angle-based outlier detection is a heuristic method, give an example where it does not work well. Can you come up with a method to overcome this issue?

---

## 11.10 Bibliographic notes

Hawkins [Haw80] defined outliers from a statistics angle. For surveys or tutorials on the subject of outlier and anomaly detection, see Chandola, Banerjee, and Kumar [CBK09]; Hodge and Austin [HA04]; Agyemang, Barker, and Alhaji [ABA06]; Markou and Singh [MS03a,MS03b]; Patcha and Park [PP07]; Beckman and Cook [BC83]; Ben-Gal [BG05]; and Bakar, Mohemad, Ahmad, and Deris [BMAD06]. For outlier detection on temporal data, see Gupta, Gao, Aggarwal, and Han [GGAH13]. For anomaly detection on graph data, see Akoglu, Tong, and Koutra [ATK15]. Song et al. [SWJR07] proposed the notion of conditional anomaly and contextual outlier detection. For a comprehensive textbook on outlier detection, see Aggarwal [Agg15c].

Fujimaki, Yairi, and Machida [FYM05] presented an example of semisupervised outlier detection using a set of labeled “normal” objects. For an example of semisupervised outlier detection using labeled outliers, see Dasgupta and Majumdar [DM02]. For outlier detection with active learning, see

He and Carbonell [HC08]; Prateek and Ashish [JK09]; He, Liu, and Richard [HLL08]; Hospedales, Gong, and Xiang [HGX11]; and Zhou and He [ZHCD15].

Shewhart [She31] assumed that most objects follow a Gaussian distribution and used  $3\sigma$  as the threshold for identifying outliers, where  $\sigma$  is the standard deviation. Boxplots are used to detect and visualize outliers in various applications such as medical data (Horn, Feng, Li, and Pesce [HFLP01]). Grubb's test was described by Grubbs [Gru69], Stefansky [Ste72], and Anscombe and Guttman [AG60]. Laurikkala, Juhola, and Kentalo [LJK00] and Aggarwal and Yu [AY01] extended Grubb's test to detect multivariate outliers. Use of the  $\chi^2$  statistic to detect multivariate outliers was studied by Ye and Chen [YC01].

Agarwal [Aga06] used Gaussian mixture models to capture "normal data." Abraham and Box [AB79] assumed that outliers are generated by a normal distribution with a substantially larger variance. Eskin [Esk00] used the EM algorithm to learn mixture models for normal data and outliers.

Histogram-based outlier detection methods are popular in the application domain of intrusion detection (Eskin [Esk00] and Eskin et al. [EAP<sup>+</sup>02]) and fault detection (Fawcett and Provost [FP97]).

The notion of distance-based outliers was developed by Knorr and Ng [KN97]. The index-based, nested loop-based, and grid-based approaches were explored (Knorr and Ng [KN98] and Knorr, Ng, and Tucakov [KNT00]) to speed up distance-based outlier detection. Bay and Schwabacher [BS03] and Jin, Tung, and Han [JTH01] pointed out that the CPU runtime of the nested loop method is often scalable with respect to database size. Tao, Xiao, and Zhou [TXZ06] presented an algorithm that finds all distance-based outliers by scanning the database three times with fixed main memory. For larger memory size, they proposed a method that uses only one or two scans.

The notion of density-based outliers was first developed by Breunig, Kriegel, Ng, and Sander [BKNS00]. Various methods proposed with the theme of density-based outlier detection include Jin, Tung, and Han [JTH01]; Jin, Tung, Han, and Wang [JTHW06]; and Papadimitriou et al. [PKGFO3]. The variations differ in how they estimate density.

SVD and related techniques were used in [IK04] to detect anomalies in computer systems. Incremental SVD was developed by Papadimitriou, Sun, and Faloutsos [PSF05] to find anomalies in coevolving time series data. Principal component analysis (PCA) was used to detect abnormal traffic in [BSM09]. For example-based matrix factorization, see Mahoney and Drineas [MD09] and Tong et al. [TPS<sup>+</sup>08]. For nonnegative residual matrix factorization, see Tong and Lin [TL11]. Xu, Constantine, and Sujay proposed robust PCA for outlier detection [XCS12]. Pattern-compression-based outlier detection was first developed by Vreeken, Leeuwen, and Siebes [VvLS11]. Akoglu, Tong, Vreeken, and Faloutsos [ATVF12] further improved the method by building multiple codetables.

The bootstrap method discussed in Example 11.21 was developed by Barbara et al. [BLC<sup>+</sup>03]. The FindCBOLF algorithm was given by He, Xu, and Deng [HXD03]. For the use of fixed-width clustering in outlier detection methods, see Eskin et al. [EAP<sup>+</sup>02]; Mahoney and Chan [MC03]; and He, Xu, and Deng [HXD03]. Barbara, Wu, and Jajodia [BWJ01] used multiclass classification in network intrusion detection.

Song et al. [SWJR07] and Fawcett and Provost [FP97] presented a method to reduce the problem of contextual outlier detection to one of conventional outlier detection. Yi et al. [YSJ<sup>+</sup>00] used regression techniques to detect contextual outliers in coevolving sequences. The idea in Example 11.26 for collective outlier detection on graph data is based on Noble and Cook [NC03].

The HilOut algorithm was proposed by Angiulli and Pizzuti [AP05]. Aggarwal and Yu [AY01] developed the sparsity coefficient–based subspace outlier detection method. Kriegel, Schubert, and Zimek [KSZ08] proposed angle-based outlier detection.

Zhou and Paffenroth introduced deep autoencoder for outlier detection [ZP17]. Chalapathy, Menon, and Chawla proposed to integrate one-class SVM with neural networks for effective outlier detection [CMC18]. Pang, Shen, and Hengel developed deviation networks for deep outlier detection. For surveys on deep learning–based outlier detection, see Chalapathy and Chawla [CC19] and Pang, Shen, Cao, and Hengel [PSCH20].

There are numerous applications of outlier and anomaly detection, ranging from finance [NHW<sup>+</sup>11, ZZY<sup>+</sup>17], healthcare [vCPM<sup>+</sup>16], accounting [MBA<sup>+</sup>09], intrusion detection [ZZ06], multiarmed bandit [ZWW17, BH20], to misinformation [ZG20].

# Data mining trends and research frontiers

# 12

**Still as a relatively young research field**, data mining has made significant progress and covered a broad spectrum of applications since its birth in the 1980s. Today, data mining is ubiquitous in a vast array of areas. Numerous commercial data mining systems and services are available. Many challenges, however, still remain. In this final chapter, we give a few examples of future trends and research frontiers in data mining as a prelude to further in-depth study that readers may choose to do. Section 12.1 presents an overview of methodologies for mining complex data types, which extend the concepts and tasks introduced in this book. Such mining includes mining text data, graphs and networks, and spatiotemporal data. In Section 12.2, you will learn more about data mining applications, including sentiment and opinion analysis, truth discovery and misinformation identification, information and disease propagation, and productivity and team science. Section 12.3 briefly introduces other approaches to data mining, including structuring unstructured data, data augmentation, causality analysis, network-as-a-context, and auto-ML. The social impacts of data mining are discussed in Section 12.4.

## 12.1 Mining rich data types

### 12.1.1 Mining text data

Data in the world can be organized in a highly structured form, such as that in a typical relational database, or in a semistructured data format, which is a blend between structured and unstructured data, such as XML, JSON, and HTML files. However, more than 80% of the world data reside in an unstructured format, such as text data from various written sources such as books, emails, reviews, articles, websites, news, product reviews, or social media or in rich media formats like video, image, and audio data.

Text (data) mining, also called text analytics, is the process of deriving high-quality information, such as structures, patterns, and summaries, from text. Text mining usually involves the process of mining structures from text, deriving patterns within the structured data, and evaluation and interpretation of the output. Typical text mining tasks include concept/entity extraction, extracting relations between named entities, taxonomy discovery, sentiment analysis, text categorization, text clustering, and document summarization.

A spectrum of methods have been developed to accomplish different text mining tasks. In general, these methods can be categorized into (i) supervised approach, (ii) semisupervised approach, (iii) distantly supervised approach, (iv) weakly supervised approach, and (v) self-supervised approach. Take document classification as an example. A supervised approach takes a good set of human-labeled documents as training data and applies some typical classification methods, such as support vector machine

or logistic regression, to build text classification models, which can then be used for classifying unseen documents. A semisupervised approach constructs models based on a combination of labeled and unlabeled documents. A distantly supervised approach constructs models based on the labels provided in some general or distant domains, such as those in Wikipedia or general knowledge bases. A weakly supervised approach builds models relying on a small set of labeled documents or keywords together with a large set of unlabeled data. Finally, self-supervised learning eliminates the necessity of data labeling by building models autonomously based on a large set of unlabeled data.

**Representation learning for natural language understanding.** A fundamentally important issue for text mining is how to represent text primitives (e.g., words, phrases, sentences, and documents) and how to learn effective representation from unstructured text. Early studies use *symbol-based representation*, such as using one-hot vectors for words (i.e., assigning 1 at the word appearing position and 0 at all other positions) and using bag-of-words model for documents. However, symbol-based representation encounters data sparsity and high-dimensionality challenges. Recent studies have been using *distributed representation* where each object (e.g., word) is represented (or encoded) with a low-dimensional real-valued dense vector. With a deep neural network architecture, distributed representation can be learned effectively with a large amount of unlabeled natural language text data. Distributed representation can represent data in a more compact and smooth way, carrying rich semantic meaning of an object by encoding its semantics based on the context of the object in the text. We first examine one such effort: word embedding.

**Word embedding.** With rich semantic meaning carried by a large number of words in a natural language, words essentially stand in very high-dimensional space. Word embedding encodes words from a space of high dimensionality to a continuous vector space with a much lower dimensionality so that the words similar in meaning are expected to be closer in the embedded space. Methods to generate such embeddings are language modeling and feature learning techniques including neural networks, dimensionality reduction on the word cooccurrence matrix, probabilistic models, explainable knowledge base methods, and explicit representation in terms of the context in which words appear.

*Word2vec* is a popular word embedding method developed by Mikolov et al. at Google around 2013. *Word2vec* is a two-layer neural network that processes text by taking a large corpus of text as its input and producing a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are embedded in close proximity in the vector space. For example, the word “*Sweden*” can be embedded close to *Norway*, *Denmark*, *Finland*, *Switzerland*, *Belgium*, *Netherlands*, and so on. Given enough data, usage, and contexts, *Word2vec* can make highly accurate guesses about a word’s meaning based on past appearances.

*Word2vec* can utilize either of two model architectures to produce a distributed representation of words (see Fig. 12.1<sup>1</sup>): (1) continuous bag-of-words (CBOW) or (2) continuous skip-gram (Skip-gram). Given a corpus, the CBOW architecture predicts the current word from a window of surrounding context words, where the order of context words does not influence prediction (bag-of-words assumption). On the other hand, the Skip-gram architecture uses the current word to predict the surrounding window of context words, and it weighs nearby context words more heavily than more distant context words.

---

<sup>1</sup> Figure adapted from T. Mikolov, K. Chen, G. Corrado, J. Dean, “Efficient estimation of word representations in vector space”: arXiv:1301.3781.

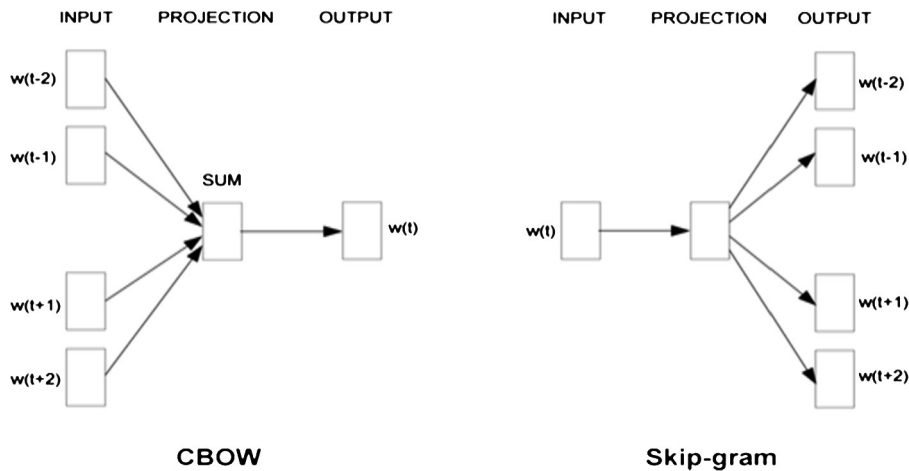


FIGURE 12.1

Two model architectures of Word2vec: CBOW and Skip-gram.

According to the authors, CBOW is faster than Skip-gram and has slightly better accuracy for the frequent words, whereas Skip-gram works well with a small amount of training data and represents well even rare words or phrases.

After Word2vec, there have been many word embedding methods proposed, notably, *GloVe* (coined from Global Vectors) developed at Stanford, by combining the features of two model families: the global matrix factorization and local context window methods, and *fastText*, developed at Facebook, which represents each word as an  $n$ -gram of characters, allows the embeddings to capture information about suffixes and prefixes, and enriches word vectors with subword information. Besides embedding words in Euclidean space, one can also embed words in hyperbolic space to model hierarchical structures, such as *Poincaré embedding*. Since cosine similarity is computed in spherical space, embedding in spherical space is recommended. Moreover, word embedding should also consider integration of its global context (e.g., in surrounding paragraphs or in the whole short document) with its local context. This leads to a new algorithm, *JoSE*, which conducts (local and global) joint spherical embedding. Besides computing word embedding only based on underlying corpus without any supervision, a user may like to influence the embedding computation based on his/her preference, such as making the embedding clustered according to topics (e.g., science vs. politics vs. sports) or according to regions (e.g., United States vs. Europe). A user-guided topic embedding method, *CatE*, takes user-provided category names as weak supervision in embedding computation.

**Pretrained language models.** Starting with ELMo and BERT around 2018, research in recent years has generated a good set of new methods under the umbrella of Pretrained Language Models (PLMs). Such pretrained language models use larger corpora, more parameters, and more computing resources than word embedding methods to pretrain large and effective models on large corpora. They use multilayer neural networks to calculate dynamic representations for the words based on their context. That is, they capture the subtlety that a word should have different meanings in different contexts, generate contextualized word embedding, and thus can distinguish multiple word senses in different contexts.

Moreover, many models adopt a pretraining-and-fine-tuning pipeline, that is, using the same neural network structure in both pretraining (for initialization) and fine-tuning on downstream tasks. Pretrained language models implicitly encode a variety of linguistic knowledge and patterns inside their multilayer network parameters and have achieved state-of-the-art performance on many NLP benchmarks. By exploring large-scale data and great computing power, new pretrained language models emerge rapidly, constantly advancing the frontiers of natural language processing and text mining.

Text mining has been focused on several tasks as outlined below.

**Information extraction (IE).** Information extraction is the task of automatically extracting structured information from unstructured and/or semistructured documents. The most typical subtasks in IE include (i) *named entity recognition* (NER), which recognizes known entity names (for people and organizations), locations, temporal expressions, and certain types of numerical expressions, by employing existing knowledge of the domain or information extracted from other sentences; (ii) *coreference resolution*, which detects coreference and anaphoric links between text spans (especially, finding links between previously-extracted named entities); and (iii) *relationship extraction* (RE), which identifies relations between entities (e.g., “Jeff lives in Chicago.”). *Event extraction* is also a popular subtask of IE, which extracts events often represented by verbs and their associated objects, such as “arrest a suspect,” or “vaccinate a child.” Phrase mining, introduced in Section 5.6 as a way of finding meaningful single or multiword phrases from documents helps entity/event extraction as well. Although human-annotated documents have been typically used for training NER models, recent studies have been exploring weakly or distantly supervised approaches to reduce human annotation efforts using Wikipedia or other existing general or domain-specific knowledge bases, pretrained language models, and text embedding methods.

**Taxonomy construction:** A taxonomy is a structured (and often hierarchical) organization or classification of things or concepts. Different applications may need to organize objects/concepts differently or even organize multiple layers differently according to some user-defined facets (e.g., region vs. theme). Thus it is desirable to construct different multifaceted taxonomies based on different corpora or applications. Manually developing and maintaining a taxonomy is labor-intensive and costly. Moreover, domain modelers may have their own points of view, which may inevitably lead to inconsistency. Automatic taxonomy construction uses text analysis techniques to automatically generate a taxonomy for a domain or a corpus to avoid the above problems. A taxonomy may also be automatically maintained or expanded based on the dynamic updates of corpus. A taxonomy can be used to organize and index knowledge (stored as documents, articles, videos, etc.), classify documents, help users search and comprehend knowledge stored in a large corpus, and provide structured guidance in many other knowledge engineering tasks.

**Text clustering and topic modeling.** *Text clustering* is to group a set of unlabeled texts (e.g., documents, sentences, words) in such a way that texts in the same cluster are more similar to each other than to those in other clusters. Text clustering algorithms process texts and determine if natural clusters (groups) exist in the data based on some similarity measures. Text clustering can benefit many text analysis tasks, such as document retrieval (adding similar documents to improve recall), hierarchical taxonomy generation, language translation, and social media analysis.

One special kind of text clustering analysis, called *topic modeling*, is to discover hidden semantic structures in a text body, especially a set of abstract “topics” (i.e., clusters of similar words) that occur in a collection of documents. Typical topic models include probabilistic topic models, such as LDA

(Latent Dirichlet Allocation) and PLSA (Probabilistic Latent Semantic Analysis), which are statistical algorithms for discovering the latent semantic structures of an extensive text body. Recent studies have been exploring different embedding and pretrained language models for topic modeling. Topic models can help organize and offer insights for us to understand large collections of unstructured text bodies, besides many other applications.

**Text classification.** *Text classification* (also known as *text categorization*), is to analyze open-ended texts (e.g., documents, messages, or webpages) and assign them to a set of predefined topics or categories. Since manual text classification by human annotators is time-consuming, expensive, and non-scalable, many algorithms have been developed for automated text classification adopting a supervised or semisupervised approach. A supervised approach takes a good amount of training data (i.e., annotated by experts or tagged with human feedback) to construct a quality classification model. A semisupervised approach takes a portion of labeled texts but a large amount of unlabeled data as training data. Its further development leads to *weakly supervised text classification* methods that reduce the labeled data to a very small amount (e.g., using class label name only) to conduct effective text classification. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text. It has broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection.

**Text summarization.** Automatic text summarization is the process of creating a subset of text that represents the most important or relevant information from a single piece or multiple pieces of text (typically, document). Based on the text to be summarized, text summarization can be categorized as *single-document* or *multidocument summarization*. There are two general approaches to text summarization: *extraction* and *abstraction*. *Extractive summarization* is to extract content (e.g., key sentences or key-phrases) from the original data, without modification. *Abstractive summarization* builds an internal semantic representation of the original content and then uses it to create a summary that is closer to what a human might express. Recent studies have also proposed *extract-then-rewrite* methods for text summarization, which can be regarded as a combination of extractive methods and abstractive methods. One may also perform *machine-aided human summarization* with human postprocessing of machine-generated summary candidates.

### **Bibliographic notes**

As an interdisciplinary research field, text mining is closely intertwined with natural language processing. Comprehensive books cover different themes on text mining (e.g., Agarwal and Zhai [AZ12], Zong, Xia, and Zhang [ZXZ21]), natural language processing (e.g., Jurafsky and Martin [JM09], Manning and Schuetze [MS01]), and representation learning for natural language processing (e.g., Liu, Lin and Sun [LLS20]).

Early researchers propose vector representation of words (e.g., Salton, Wong, and Yang [SWY75]). Recent research has been focusing on learning distributed representations for natural language processing, including an early overview by Bengio, Courville, and Vincent [BCV13]. *Word2vec* is proposed by Mikolov et al. [MSC<sup>+</sup>13] at Google, *GloVe* [PSM14] by Pennington, Socher, and Manning at Stanford, and *fastText* [BGJM16] by Bojanowski, Grave, Joulin, and Mikolov at Facebook. *Poincaré embeddings* for learning hierarchical representations is proposed by Nickel and Kiela [NK17]. A joint spherical embedding method, *JoSE*, is proposed by Meng et al. [MHW<sup>+</sup>19], which is further developed for user-guided topic embedding (e.g., *CatE* [MHW<sup>+</sup>20]) and user-guided hierarchical topic embedding (e.g., *JoSH* [MZH<sup>+</sup>20b]).



Pretrained language models (PLMs) have been studied popularly in recent years. Vaswani et al. have proposed a *Self-Attention* mechanism [VSP<sup>+</sup>17]. Started with ELMo by Peters et al. [PNI<sup>+</sup>18] and BERT by Devlin et al. [DCLT19], recent studies have developed many new methods, including RoBERTa [LOG<sup>+</sup>19], XLNet [YDY<sup>+</sup>19], and ELECTRA [CLLM20], among others. Most of these models are based on a deep learning architecture, Transformer [VSP<sup>+</sup>17]. An overview of Transformers proposed recently is in [LWLQ21].

Progress on various text mining tasks, including information extraction, taxonomy construction, text clustering and topic modeling, text classification, and text summarization, has been surveyed in various forums in the fields of natural language processing, machine learning, data mining, and data science, and will not be illustrated here in this short introduction.

### 12.1.2 Spatial-temporal data

Space and time are essential in many applications, such as smart cities, epidemiology, earth sciences, ecology, climatology, astronomy, and astronautics. In general, many data mining principles and methods, including those introduced in this book, can be adapted to apply on spatial and temporal data. At the same time, spatial-temporal data have some unique characteristics, which lead to noteworthy challenges and also opportunities for data mining techniques. Let us briefly discuss three important aspects of spatial and temporal data and applications, namely unique properties, data types, and data models for spatial-temporal data.

#### ***Auto-correlation and heterogeneity in spatial and temporal data***

One important property commonly exists in spatial and temporal data and applications is that dependencies and correlations often happen in proximate locations and time. Take road traffic as an example, the traffic status of an intersection at this moment is often highly correlated to the traffic statuses of the adjacent intersections at the same moment and also to the traffic status of this intersection at the last moment, such as five minutes ago. This is called the *auto-correlation property* of spatial and temporal data.

Another important property in spatial and temporal data and applications is *heterogeneity*; that is, spatial and temporal data are often heterogeneous and nonstationary in distribution. For example, the road traffic status of an intersection in a city may present to some extent a periodic pattern on weekdays and weekends. At the same time, there may be some more sophisticated changes with longer periodicity due to, for example, seasons and local and national economic development.

Clearly, the auto-correlation property and the heterogeneity in spatial and temporal data and applications require more sophisticated modeling and also lead to new opportunities for new data mining techniques.

#### ***Spatial and temporal data types***

Spatial and temporal data often have some advanced data types to represent more sophisticated semantics in applications. Particularly, events, trajectories, point reference data and raster data are the most important data types specific to spatial and temporal data and applications.

In general, an *event* happens at a spatial location and a time point. For example, in smart city management, a traffic accident happens at a location and a time. The representation of an event may be

extended to cover a spatial region, such as an area described by a polygon and/or a period between a starting time stamp and an end time stamp.

The frequently used operations on event data include intersections and similarity/difference between two events. For example, a forest fire and a rainfall can be recorded as two events, respectively, each covering an area and a period of time. The intersection of the two events returns the overlapping area and time of the two events and may be used to mine the effect of rainfall on alleviating forest fires. As another example, we may want to cluster accidents in highways and identify patterns in location, time, and other related attributes. To measure the similarity in location, in addition to the Euclidean distance, we may also consider the shortest path distance in the road network. To measure the similarity in time, in addition to the absolute difference in time, we may also consider the periodicities of day and week. For example, one accident happening at 12:30 pm last Tuesday may still be considered very similar to an accident happening at 1:05 pm this Thursday, since they both happened around weekday noon.

A *trajectory* is a path of an object over space and time. For example, tracing a vehicle produces a trajectory. Trajectory data are common in many spatial temporal applications, such as transportation, epidemiology, astronomy, and ecology. The frequently used operations on trajectory data include finding relationships between two trajectories. For example, to determine whether two trajectories intersect with each other, we need to check whether the two objects creating the trajectories are once close to each other in both space and time, such as within a spatial distance of at most  $\Delta d > 0$  and a time window of at most  $\Delta t > 0$ .

In many spatial and temporal data mining tasks, we are interested in a field evolving over time. For example, a meteorologist may want to maintain a field of temperature in space and time. There are two general ways to obtain observation data to reconstruct and estimate the field of the target variable.

The first way is to collect *point reference data*. For example, temperature data may often be collected using mobile sensors, such as weather balloons floating in space. While the field of temperature is continuous in both space and time, the data are collected using those discrete reference points, and then the field is reconstructed. Some frequently used operations on point reference data include reconstructing fields and modeling nonstationary spatial random processes. For example, based on the point reference data, one may use smoothing techniques to reconstruct the field of a spatial-temporal variable. Moreover, one may apply variogram model to describe and analyze the behavior of a nonstationary spatial random process.

Alternatively, one may collect *raster data* by recording observation data at fixed locations in space and at fixed points in time. In general, a raster data set can be written as  $S \times T$ , where  $S$  is a set of fixed locations, and  $T$  is a set of fixed time points. Every pair  $(s, t) \in S \times T$  is associated with the observation at location  $s$  at time  $t$ . Raster data are popularly used in many applications, such as medical imaging, demography, and remote sensing. Some frequently used operations on raster data include converting a raster to a finer or coarser resolution. For example, road traffic data as raster data are collected periodically (e.g., every minute) through road sensors and cameras set up on roadsides. However, in some applications, one may want to have traffic data at a finer resolution. For example, one may want to estimate the traffic of a segment of an alley where no sensors or cameras are deployed, although data may be collected at the major roads that the alley connects to. In order to convert the raster to a finer resolution, one may apply interpolation. Moreover, to obtain the traffic data at a coarser solution, such as at the larger road block level, one may apply aggregation.

### ***Spatial and temporal data models***

Using the four data types, events, trajectories, point reference data, and raster data, we can represent many data objects in spatial and temporal applications, such as points, trajectories, time series, spatial maps, and rasters. In general, there are three types of models for spatial data: the object model, the field model, and the spatial network model. The *object model* uses points, lines, and polygons to describe spatial data objects. More attributes may be associated with objects. For example, in road traffic mining, one may use points to describe vehicles and use lines to describe roads. A traffic jam may be described using a polygon. The *field model* describes spatial information as a function and thus is suitable for modeling continuous spatial data. For example, one may use the field model to describe temperature, humidity and some other meteorology variables over a geographical space. The *spatial network model* uses graphs to represent the relationship among spatial elements. For example, a road network models the road distance between points of interest.

To incorporate temporal data, three models can be used: the temporal snapshot model, the temporal change model, and the event/process model. In the *temporal snapshot model*, there are multiple spatial layers of the same theme associated with time stamps. For example, to model the development of wildfires, the remote sensing spatial images of wildfires at different time points are collected as different layers of the same theme. The *temporal change model* represents a spatial theme using a start time and the incremental changes. For example, to describe the moving of a vehicle using the temporal change model, one can describe the start time and the initial location and record the speed, direction, and acceleration. The *event and process model* records multiple events and processes over time using intervals. For example, one can represent volcanic activities over the world using the event and process model, which can facilitate the comparison among those activities in time and locations.

Due to the rich and unique data types and data models for spatial and temporal information, many generic data mining methods, while still valid on spatial and temporal applications, may have to be adapted to be applied on spatial and temporal data. Moreover, specific methods may need to be developed to mine spatial and temporal relationships, which are unique to spatial and temporal data.

### ***Bibliographic notes***

There are a few good surveys on mining spatial and temporal data (Atluri, Karpatne, and Kumar [AKK18] and Shekhar, Vatsavai, and Celik [SVC08]).

### **12.1.3 Graph and networks**

With the technology advancement of information, biology, industry, and beyond, our world has become increasingly connected than ever before. Many real-world complex systems consist of a vast number of interacting components like the users of the online social platforms, molecular regulators (e.g., DNA, RNA, etc.) in the gene relation systems, and sensors of different functionalities in the complex surveillance systems. Mathematically, such a kind of complex systems can be often abstracted and modeled by a graph or network,<sup>2</sup> which essentially consists of a collection of nodes (i.e., the objects in the system) interconnected with each other by a collection of edges (i.e., the interactions among the objects).

---

<sup>2</sup> Hereafter, we use graph and network interchangeably.

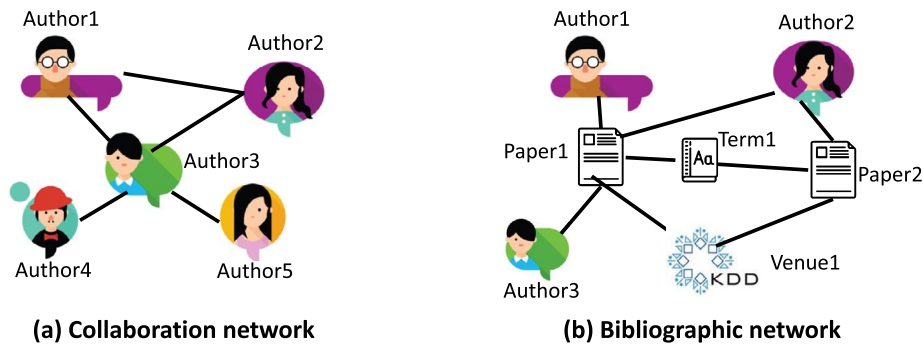


FIGURE 12.2

Examples of a homogeneous network and heterogeneous network.

Graphs can be categorized in different aspects. To name a few, based on whether node and edge attributes exist in the graphs, they can be categorized as plain graphs and attributed graphs. Based on whether graphs change over time, they can be categorized into static graphs and dynamic graphs. In addition, based on whether nodes and edges are of different types, they can be categorized into homogeneous and heterogeneous networks, respectively. Let us look at some examples to illustrate homogeneous and heterogeneous networks in Example 12.1.

**Example 12.1.** In a homogeneous collaboration network, each node represents an author and each edge indicates the coauthorship between two authors, shown in Fig. 12.2(a). In a heterogeneous bibliographic network shown in Fig. 12.2(b), in addition to the author nodes, other types of nodes (e.g., papers, venues, terms, etc.) exist. Accordingly, different types of edges describe different semantics. For example, edges between papers and venues indicate the “is published at” relation, whereas an edge between a paper and a term (e.g., a keyword) implies that the term appears in the paper. □

*How can we discover the insights and patterns from graphs?* Graph mining has been a very active research area. Here, we give three examples, including (1) graph modeling, (2) heterogeneous network mining, and (3) knowledge graph mining.

**Graph modeling.** Understanding how graphs are created and formed plays a critical role in discovering graph structural patterns and discerning the underlying governing mechanism of the graphs. In addition, it helps to anonymize social networks to protect user privacy. This motivates the graph modeling problem, which aims to study and simulate the generation mechanisms behind real-world graphs. It has been extensively studied in the past decades, and a variety of graph generation models has been proposed. Here, we introduce the following graph models, including (1) Erdős-Rényi (ER) graph model and its variants, (2) realistic graph generators, and (3) deep graph generative models.

*ER graph model.* The well-known ER graph model and its variants assume certain probability distributions, based on which edges are added randomly. The classic ER graph model assumes each edge has a fixed probability of being present and is independent of the existence of other edges. Although mathematically elegant, the ER model results in the exponential degree distribution, which is at odds with the degree distributions of real-world graphs, such as the power law degree distribution of social networks.

The power law random graph model, on the other hand, explicitly assumes the degree distribution to follow power law distribution.

*Realistic graph generators.* Different from the ER graph model, realistic graph generators aim to create succinct models that mimic one or more properties of real-world graphs. One class of these models are the preferential attachment based models that grow graphs by iteratively adding nodes and edges while preserving the rich-get-richer principle such as the Barabási-Albert model (i.e., BA model) and its variant AB model. Another realistic graph generator is the stochastic Kronecker graph model that grows the graph by iteratively applying Kronecker product starting from a small seed graph.

*Deep graph generative model.* The traditional models introduced above are hand-crafted to fit a few specific properties of the graphs but not others, so they might lack the flexibility of being able to capture various types of graphs (e.g., molecular graphs). With the advance of graph neural networks (introduced in Chapter 10), classic deep generative models can be extended to graphs. Different from the previous graph models, deep graph generative models directly learn to generate the synthetic graphs that mimic the real graphs with high fidelity by applying deep learning models such as variational auto-encoder, generative adversarial networks, and deep auto-regressive model.

Despite their power of modeling the latent characteristics of graphs owing to the learning capability, these deep graph generative models generally require high training overheads and thus are less efficient than the traditional graph models. Unlike these traditional graph modeling methods, the interpretability of the graph neural networks-based modeling often lacks (e.g., what aspects or patterns of the graphs these models focus on capturing). In addition, most, if not all, of the existing graph modeling methods focus on homogeneous networks while leaving the modeling on heterogeneous networks absent.

**Heterogeneous network mining.** With multiple types of nodes and edges, heterogeneous networks naturally bring the advantages of capturing rich semantics compared with the plain homogeneous networks. A core concept in heterogeneous networks is *metapath*, which is a sequence of relations that imply certain semantic meanings among multiple types of nodes. For example, in Fig. 12.2(b), Author1-Paper1-Author3 is an instance of the metapath Author-Paper-Author that indicates the coauthorship. The rich semantics benefit various data mining tasks on networks such as similarity search, classification, clustering and network embedding. Let us elaborate on this. First, similarity search aims to measure the similarities among nodes and return the top- $k$  most similar nodes to the given query node. Different from homogeneous networks, by considering metapaths with different physical meanings, the similarities among the nodes may vary and exhibit the ranking results in different aspects, which could be more accurate. Second, for the node classification task, the goal is to classify all types of nodes into different labels. Specifically, in the transductive setting, the similarity matrices derived by different metapaths can be used in the consistency assumption to infer node labels. Third, different semantics of metapaths could lead to different clustering results representing diverse meanings. Lastly, heterogeneous network embedding aims to learn the low-dimensional embedding vectors of nodes that can be used in other downstream tasks. In the vast majority of heterogeneous network embedding methods, metapaths are used as a core component to construct node neighborhood for representation learning.

Most of the existing methods on heterogeneous networks require domain knowledge to manually define meaningful metapaths. As such, poorly designed metapaths could weaken, instead of strengthen, the effectiveness of the mining model. In this way, a promising future direction is to avoid the manual predesigned of metapaths such that the structure and semantics of heterogeneous networks can be automatically inferred from the input data.

**Knowledge graph mining.** Knowledge graph, which is essentially a directed heterogeneous multi-graph, is often used to link concepts and entities of different types through human-interpretable semantics. It has been widely used in a wealth of applications. For example, the knowledge graph constructed upon DBpedia<sup>3</sup> is capable of linking different concepts and entities (e.g., person, location, etc.) from Wikipedia and has been used in many question answering and fact checking applications. Google knowledge graph with its information retrieved from many sources is used to enhance Google's search engine. Product knowledge graph, which describes the relations among products and their facts, facilitates many e-commerce services. Despite the tremendous benefits from knowledge graphs, it is nontrivial to construct and update them. Take knowledge graph construction from unstructured texts as an example. Traditional approaches rely on the prespecified ontology and a large amount of human labors for annotations, which makes it impractical to grow the knowledge graph when humans acquire new knowledge. It has been a long-standing task to reduce or even avoid human labors so that knowledge graphs can be automatically constructed. To be specific, the extractive construction usually contains information extraction as its key stage that involves many NLP and text mining tasks (e.g., semantic role labeling). Recent knowledge graph construction methods show a promising avenue of producing novel and diverse commonsense knowledge by training language models. Moreover, typical data mining tasks on knowledge graphs include link prediction (i.e., whether an entity has a specific relation with another entity), entity resolution (i.e., whether two entities represent the same object) and triplet classification. Most of the existing methods for these tasks are often based on representation learning.

Despite the extensive research on knowledge graphs, there still exist many limitations and challenges. For example, in addition to the specific challenges associated with the tasks for information extraction, the extractive construction of knowledge graphs is often limited to the knowledge mentioned explicitly in the texts. Besides, great efforts are still needed towards automatic high-precision knowledge graph construction and dynamic update. With the aid of knowledge graphs, many interesting data mining applications are currently underexplored, such as medical AI, conversational AI, and academic search engine.

### ***Bibliographic notes***

The ER graph model is first proposed by Erdős-Rényi in 1960, which assumes edges exist in the graph with the same probability (Erdős and Rényi [ER60]). With its simple and elegant rule, the ER graph model has been one of the most widely studied graph models and applied often as the starting point to many network analyses (Lyzinski, Fishkind, and Priebe [LFP14] and Guimera, Sales-Pardo, and Amaral [GSPA04]). The power law random graph model further extends the ER model to randomly insert edges to the nodes such that their degree sequence matches the power law distribution (Aiello, Chung, and Lu [ACL00]). The stochastic block model can be considered as a generalized graph model of the ER model and encodes the community structure into graphs (Holland, Laskey, and Leinhardt [HLL83]). Following the preferential attachment principle that the rich get richer, the BA graph model in each iteration adds a new node and connects it to some existing nodes with the probabilities proportional to their node degrees (Barabási and Albert [BA99]). Dangalchev, Mendes, and Samukhin propose to extend the BA model by considering the attractiveness based on the degree of both existing nodes and

---

<sup>3</sup> <https://wiki.dbpedia.org/>.

their corresponding neighbors (Dorogovtsev, Fernando, Mendes, and Samukhin [DMS00]). The AB model further applies the edge rewiring process to the BA model (Albert and Barabási [AB00]). The forest fire model also follows the preferential attachment principle by iteratively connecting the new nodes to some of the nodes that are connected with a uniformly selected ambassador node (Leskovec, Kleinberg, and Faloutsos [LKF05]). The Kronecker graph generator models graphs by recursively applying Kronecker multiplication between the intermediate adjacency matrix and itself (Leskovec et al. [LCK<sup>+</sup>10]). Another random graph generator, known as Watts–Strogatz model (Watts and Strogatz [WS98]), captures the small-world property of many real graphs (e.g., short average path lengths and high clustering coefficients). To improve the flexibility and expressiveness of the graph models, many deep graph generative models have been proposed. To name a few, GraphVAE leverages the variational autoencoder (VAE) with graph matching to learn how to construct graphs from their vector representations (Simonovsky and Komodakis [SK18]). Graphite utilizes an iterative refinement process on top of GraphVAE (Grover, Zweig, and Ermon [GZE19]). Moreover, NetGAN leverages the GANs to generate large-scale graphs that preserve many patterns in real graphs without explicitly specifying them (Bojchevski, Shchur, Zügner, and Günnemann [BSZG18]). GANs can be also used to generate molecular graphs (De Cao and Kipf [DCK18] and Wang et al. [WWW<sup>+</sup>18]). In addition, GraphRNN generates synthetic graphs in a sequential generation fashion with deep auto-regressive model (You et al. [YYR<sup>+</sup>18]).

One representative similarity measure based on metapaths is PathSim, which measures to what extent the nodes of the same type are connected and share similar visibility with the query node, defined on the symmetric metapath of interests (Sun et al. [SHY<sup>+</sup>11]). HeteSim is subsequently designed to measure the relevance not only among the nodes of the same type, but also those of different types (Shi et al. [SKH<sup>+</sup>14]). For node classification, GNetMine classifies nodes by preserving consistency over each type of edges (Ji et al. [JSD<sup>+</sup>10]). HetPathMine then extends the consistency over different types of edges to metapaths with different semantic meanings (Luo, Guan, Wang, and Lin [LGWL14]). For node clustering on heterogeneous networks, PathSelClus learns the weights of different metapaths, based on which generates node clusters (Sun et al. [SNH<sup>+</sup>13]). Regarding heterogeneous network embedding, metapath2vec uses metapath based random walks to construct contexts of nodes and then learns node embedding vectors by SkipGram model with negative sampling (Dong, Chawla, and Swami [DCS17]). In addition, by generating neighbors with metapaths, the attention mechanism can be applied to aggregate node representations among node neighbors (Wang et al. [WJS<sup>+</sup>19]).

Many knowledge graph construction methods are based on open information extraction (Fader, Soderland, and Etzioni [FSE11]; Schmitz et al. [SSB<sup>+</sup>12]; and Mehta, Singhal, and Karlapalem [MSK19]). COMET is a generative approach to knowledge graph construction by training transformer language models (Bosselut et al. [BRS<sup>+</sup>19]). For other mining tasks, embedding-based methods have been widely used. For example, by knowledge graph embedding methods, including the translation-based methods (Bordes et al. [BUGD<sup>+</sup>13] and Wang, Zhang, Feng, and Chen [WZFC14]), rotation-based methods (Sun, Deng, Nie, and Tang [SDNT19]), and neural network–based methods (Dettmers, Minervini, Stenetorp, and Riedel [DMSR18]), the learned entity and relation embeddings can be naturally used in link prediction, triplet classification, and so on. Knowledge graph alignment is another task that aims to align entities across different knowledge graphs and plays an important role in knowledge fusion. Many knowledge graph alignment methods exist, including the knowledge graph embedding–based methods (Zhu, Xie, Liu, and Sun [ZXLS17] and Sun, Hu, Zhang, and Qu [SHZQ18]) and graph neural network–based methods (Wang, Lv, Lan, and Zhang [WLLZ18]; Xu et al. [XWY<sup>+</sup>19]; and

Yan et al. [YLB<sup>+</sup>21]). There also exist many works on other knowledge-aware applications, including question answering (Chen, Wu, and Zaki [CWZ19] and Huang, Zhang, Li, and Li [HZLL19]), fact checking (Shi and Wenginger [SW17]), and recommendation (Zhang et al. [ZYL<sup>+</sup>16] and Wang et al. [WHC<sup>+</sup>19]).

---

## 12.2 Data mining applications

### 12.2.1 Data mining for sentiment and opinion

In many applications, we want to conduct data mining on user produced data, such as product reviews in e-commerce platforms. An important task in mining user produced data is to understand sentiment expressed in text. This is achieved by a specific branch of data mining, called sentiment analysis and opinion mining.

#### *What are sentiments and opinions?*

Literally, sentiment is “a view of attitude toward a situation or event.” It is a feeling or emotion. Opinion is “a view or judgment formed about something, not necessarily based on fact or knowledge.” Sentiment and opinion are heavily related and subtly different. For example, the sentence “I find data mining highly interesting” expresses a sentiment, whereas “I believe data mining is promising and useful for industry applications” is more an opinion. The subtle difference is that one may share the same sentiment expressed in the first sentence and agree or disagree with the opinion in the second sentence. However, after all, the underlying meanings of the two sentences are highly related. Moreover, while many opinions imply positive or negative sentiments, some do not. For example, the opinion “I think she will move to Canada after graduation” does not come with a positive or negative sentiment immediately.

#### *Sentiment analysis and opinion mining techniques*

Sentiment analysis and opinion mining is a highly interdisciplinary direction. First of all, as we have to handle text, natural language processing techniques are extensively used to process text in sentiment analysis and opinion mining. Moreover, many problems in sentiment analysis and opinion mining can be modeled as a classification problem, such as whether a product review is positive or negative. In many scenarios, one may also want to group users according to their opinions, and thus clustering analysis is needed. Outliers are useful in sentiment analysis and opinion mining, too. For example, an application of mental health assistant may analyze social media data, such as user posts, to identify possible extreme opinions. Correspondingly, machine learning and data mining techniques are frequently used in sentiment analysis and opinion mining. As sentiments and opinions are expressed and understood by people, naturally sentiment analysis and opinion mining has to leverage insights from psychology and social sciences, and, at the same time, can be useful for those sciences as technical tools.

Sentiment analysis and opinion mining is a highly dynamic application area, since many new types of media emerge and thus new ways of sentiment and opinion expressions come out. It is impossible to limit the data mining techniques used by this fast growing area. Instead, sentiment analysis and opinion mining will keep adopting and adapting the latest development in data mining to tackle the



novel application challenges. Here, let us briefly look at some technical directions popularly explored in sentiment analysis and opinion mining.

First, we can *conduct analysis of sentiments and opinions at different levels and granularities*. At the highest level, *document-level sentiment classification* determines whether a whole document, such as a review, expresses a positive or negative sentiment. At a lower granularity, *sentence-level analysis* looks at whether a sentence expresses a positive, negative, or neutral opinion. The third level frequently used is *aspect level*, which looks at an opinion and the associated target. For example, the sentence “I really like the pasta in this restaurant, but am not a big fan for the dessert here” expresses a positive opinion on the aspect “pasta” and a negative opinion on the aspect “dessert” of the restaurant. Aspect-level sentiment analysis and opinion mining can cross multiple sentences.

Second, to identify the sentiments and opinions expressed in a document or a sentence, we can *produce sentiment/opinion lexicons and conduct sentiment classification*. As an intuitive and practical idea, one can identify a series of words and phrases as *sentiment/opinion lexicons* that can serve as indicators of sentiments, such as “good,” “wonderful,” and “fantastic” for positive sentiments, and “poor,” “bad,” and “awful” for negative sentiments. Based on sentiment/opinion lexicons, sentiment classification can be conducted. However, in many situations, only those sentiment/opinion lexicons are not sufficient or may be subtle for specific domains. For example, in most domains, “suck” is a lexicon for negative sentiments. However, for vacuum cleaners, a sentence like “My new vacuum cleaner really sucks” in a review indeed is positive. Moreover, not every sentence containing sentiment lexicons expresses sentiments. For example, sentence “What is the best pasta in your restaurant” contains a popular sentiment lexicon “best” but does not express any sentiment. At the same time, a sentence not containing any sentiment lexicon may express some sentiment, such as “My new earbuds stop pairing with my laptop after a week” expressing a negative sentiment. A lot of research is dedicated to extract and compile sentiment/opinion lexicons and handle subtleties in conducting sentiment classification using those lexicons.

Third, in general there are two types of text content on which sentiment analysis and opinion mining may be conducted. The first type is the stand-alone posts, such as product reviews and blogs in social media. The second type is online conversations, such as discussions and debates. Analyzing conversational content is much more challenging, due to the interactive and interdependent nature of pieces in a multi-round conversation. There are also some new types of sentiments in conversational content, such as agreements and disagreements. Furthermore, more data mining tasks may be conducted on conversational content than on stand-alone posts, such as checking sentiment consistency or contradictions of one person in the conversation and finding groups of participants who share similar sentiments and opinions in a conversation.

Fourth, *mining intent* is a data mining task highly related to sentiment analysis and opinion mining, since in many situations, intent may imply sentiments and express opinions. For example, the sentence “I cannot wait to see the new movie next Monday” expresses a strong positive sentiment, whereas the sentence “I just want to return this microphone immediately to the store” conveys a clear negative sentiment. Indeed, the above two sentences represent a new type of sentiment, *aspiration*. Mining intent is a natural, interesting and also challenging technical direction in sentiment analysis and opinion mining.

Last, as sentiments and opinions may be abused, in order to achieve data mining for social good, it is important to explore opinion spam detection and assess review quality. Specifically, opinion spamming is the fake or malicious opinions under hidden agendas to attack vulnerable products, services, organizations, and individuals. One important challenge is that opinion spamming typically may happen not

just as individual and independent posts. Instead, opinion spamming often happens in a coordinated manner. Therefore successfully detecting opinion spamming involves not only techniques like natural language processing and the sentiment analysis and opinion mining techniques mentioned above, but also other data mining techniques like network mining.

### ***Sentiment analysis and opinion mining applications***

Sentiment analysis and opinion mining has many applications. It is impossible to enumerate all of them or even design a categorization accommodating all of them. Here, according to how and by whom the outcome of sentiment analysis and opinion mining is consumed, we briefly present three major categories of sentiment analysis and opinion mining applications.

The outcome of sentiment analysis and opinion mining can help human being to understand the advantages and disadvantages of target objects. For example, in e-commerce, applying sentiment analysis and opinion mining on product reviews, we can aggregate customers' opinions on products and their various aspects. The outcome of the analysis and mining can be used by customers in purchase decision making, by e-commerce platform and vendors in supply chain optimization, and by product producers and manufacturers in product improvements and new product development. The key tasks in sentiment analysis and opinion mining for this kind of applications include, for example, aspect level sentiment analysis and integration of sentiment analysis and customer categorization.

The outcome of sentiment analysis and opinion mining may also be consumed directly by other artificial intelligence agents. That is, the outcome is taken by an automation system to trigger actions. For example, in the applications of stock market prediction, sentiment analysis and opinion mining is extensively applied to message board posts, social media like Twitter messages, real-time news like Bloomberg news, and many other kinds of related media information sources. The analysis and mining can produce buy and sell signals directly and thus can be used by online trading systems to take actions in stock markets. There are several grand challenges in this type of real-time sentiment analysis and opinion mining for automation systems. First, such a sentiment analysis and opinion mining system has to be highly efficient and scalable, so that they can handle massive content in a real-time manner. Second, such a system has to be highly accurate, since any mistake may either lead to a loss in real-time action like trading or missed business opportunities. Last, such a system has to be highly robust, since a lot of various kinds of noise exist in different content, and there may even be malicious attacks from opinion spammers.

Sentiment analysis and opinion mining may also be used for monitoring and government administration purpose. In other words, this category of applications is about public sentiments and opinions. For example, in electoral politics, candidates and media may monitor the change of public sentiments and opinions on various parties and candidates over time. Some governments may also monitor sentiments and opinions in cyberspace as part of the efforts of fighting against cyber-violence, terrorism, and racism, for example. In addition to the common desiderata for general sentiment analysis and opinion mining, this category of tasks faces some new challenges, such as fairness and privacy protection.

### ***Bibliographic notes***

Sentiment analysis and opinion mining started in early 2000s and remains a vivid interdisciplinary area among data mining, natural language processing, machine learning, and some others. There are many publications on the subject. Some informative survey and review books include (Liu [Liu20]; Shanahan, Qu, and Wiebe [SQW06]; Liu [Liu12]; Pang and Lee [PL08]; and Cambria and Hussain [CH15a]).

## 12.2.2 Truth discovery and misinformation identification

### Truth discovery

With huge amounts of information provided by various information sources, it may happen that different values can be provided by different sources for the same data item. Naturally, one may hope there could be some automated methods that may help assess the quality and trustability of information on the web, especially when there are different values for the same data item. This leads to an important task in data mining, called *truth discovery* (also known as *truth finding* or *fact finding*), which is to assess and choose the actual true value for a data item when different data sources provide conflicting information on it. Truth discovery is an important task since we are increasingly relying on online resources to find trusted information and make critical decisions. Truth discovery is also important for data integration from multiple data sources, since assessing and choosing the quality values becomes crucial in constructing an integrated, reliable data storage.

Truth discovery problems can be divided into two subclasses: single-truth and multitrueth. In the first case, *single-truth discovery*, only one true value is allowed for a data item (e.g., birth date of a person or capital city of a country). In this case, different values provided for a given data item oppose to each other, and the values and information sources can either be correct or erroneous. However, in the second case, *multitrueth discovery*, multiple true values are allowed (e.g., authors of a paper or members of a team). In this case, the truth is a set of values, a different value could provide a partial truth, and the information source that provides more correct values and less incorrect values for a given data item is considered more valuable.

Early studies suggested some simple voting method, that is, each source votes for a value of a certain data item and, the value with the highest vote is selected as the true one. However, unreliable sources or sources simply copying information from others can mess things up since they may take the majority in voting and overwhelm the reliable sources. Therefore a truth discovery algorithm must take source reliability into consideration and estimate the trustworthiness of data sources.

Can we rely on supervised learning to assign a reliability score to information sources based on human labeling of the provided values? Unfortunately, this may not be realistic since there are too many information sources and too huge number of facts provided by such sources. Weakly supervised or self-supervised methods based on a huge amount of unlabeled data could be a more realistic solution.

Taking a simple example in Fig. 12.3, we illustrate a process on single-truth discovery, where the link  $s_k-f_j-o_i$  indicates that object  $o_i$  is supported by fact  $f_j$  provided by source  $s_k$ . Initially, we assume all sources are independent and have equal trustworthiness. Since for  $o_2$ ,  $f_3$  is supported by three sources  $s_1$ ,  $s_3$ , and  $s_4$ , but  $f_4$  is supported by only one source  $s_2$ ,  $f_3$  is likely more reliable. From this,

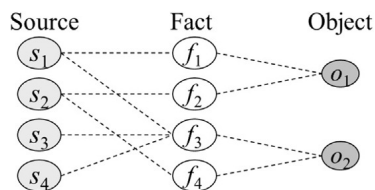


FIGURE 12.3

Truth finding.

we may assert that  $s_1$ ,  $s_3$ , and  $s_4$  are likely more trustable sources than  $s_2$ . Then for  $o_1$ ,  $f_1$  is likely more reliable than  $f_2$  since  $f_1$  is supported by  $s_1$  but  $f_2$  by  $s_2$ .

This example shows truth discovery needs to compute both source trustworthiness and claim reliability in a mutually enhanced manner. Source trustworthiness usually is not known a priori but estimated with an iterative approach. At each step of the truth discovery algorithm the trustworthiness score of each data source is recomputed, improving the assessment of the true values that in turn leads to a better estimation of the trustworthiness of the sources. This process usually ends when all the values reach a convergence state.

For effective truth discovery, it is important to detect copying behaviors since copying allows spreading false values easily, leading many sources to vote for the wrong values, and complicating the truth discovery process. Usually systems decrease the weight of votes associated with copied values or even do not count them at all. In some applications, data sources may have different trustworthiness for different categories of data. For example, a bookseller specialized on children books may provide high-quality information on children books but not as reliable information on other books, such as science and engineering. In this case, trustworthiness of data source can be computed separately for different categories of objects.

For multitrueth discovery, the evaluation of trustworthiness of a data source should consider multiple truth values for a data item (e.g., the authors of a book). A data source that provides more correct values for a data item should receive a reward but that provides incorrect values for the data item should receive a penalty. Iterative mutual enhancement methods, Bayesian-based inference methods and probabilistic graphical model-based methods have been developed for such analysis. Also, more sophisticated methods should also consider domain coverage and copying behaviors to better estimate source trustworthiness.

Some truth-finding methods evaluate the confidence of *numerical value-based facts* (e.g., the population of a city or the temperature of a time in a region). Even for trusted data sources, it is quite possible to provide slightly different numerical values (e.g., depending on the time and method of measurement). Thus the evaluation of source trustworthiness and fact confidence should be based on data distribution and treat only those answers that deviated substantially from the norm as wrong ones. Similar truth-finding mechanisms have been developed based on such observations.

### ***Identification of misinformation***

Different from truth discovery that identifies correct information from conflicting ones, a more challenging task is the identification of misinformation. *Misinformation* is false, inaccurate, or misleading information deliberately created by some data source and is propagated regardless of the intention of deception. Typical examples of misinformation include false rumors, insults, and hoaxes. Other related terms include *disinformation* that is a subset of misinformation that is deliberately deceptive (e.g., malicious hoaxes, fabricated information, and propaganda), and *fake news*, which refers to false information in the form of news. The principal effect of misinformation is to elicit fear and suspicion among a population since such population may take misinformation as credible or true.

In the information age, social networking sites have become a notable agent for the spread of misinformation, fake news, and propaganda. Misinformation on social media spreads quickly in comparison with traditional media because of the lack of regulation and examination before posting. Any users on such social media can generate and spread information quickly to other users without requiring the confirmation of its truth.

Ideas similar to those developed from the analysis of sources and claims in truth discovery can be used for misinformation identification. For example, one can explore the mutual enhancement between source trustworthiness and claim credibility. The consistent information provided by multiple news agencies and/or authoritative websites (i.e., those managed by responsible editors) can be considered credible. A source is more trustworthy if it provides more credible information and no or less noncredible information. For the conflicting information on the same events/facts, the piece provided consistently by multiple trustworthy sources is more likely to be credible, whereas those provided by less trustworthy sources and conflicting with the credible one are likely misinformation.

In today's world, massive social media are provided by multiple groups, parties, and countries of rival interests. Additional vigilance is needed to identify misinformation. It is insufficient to rely on simple votes of the claims since fabricated information, repeating hundreds of times or propagated by the groups of similar interests or biases, could be mistaken as true information by a population. Methods need to be developed to distinguish misinformation from genuine information, with additional measures, such as building up information literacy and media literacy through education, using commonsense knowledge and open-mindedness, developing critical thinking, holding nonbiased view, and avoiding motivated reasoning just based on one's own preference or belief. Eventually, identification of misinformation will be determined by individual's mental model, individual's worldview beliefs, taking into consideration of repetition of misinformation, time-lag between misinformation and correct information, and relative coherency between misinformation and corrective message.

Flagging or eliminating fake news and misinformation using algorithmic fact checkers is becoming the front line in the battle against the spread of misinformation. Methods that automatically detect misinformation are still under active research, by further developing methods in natural language processing and social network analysis, extracting, clustering, and classifying information for assessing the quality or credibility of the information provided. Nevertheless, effective software programs are being deployed by major information technology companies to detect and alert likely misinformation and supply supplemental information for fact checking.

### ***Bibliographic notes***

Truth discovery has been studied by many researchers in recent years [LGM<sup>+</sup>15]. A study on interaction between information source and information provided for webpage ranking can be traced back to Kleinberg [Kle99]. Yin, Han, and Yu [YHY08] propose TruthFinder, a truth discovery method based on the mutual enhancement between source trustworthiness and claim reliability. Li et al. study truth finding in structured data in deep web [LDL<sup>+</sup>12]. Dong and Srivastava carry out a series of studies on truth finding from data integration point of view [DS15]. Zhao et al. [ZRGH12] study truth finding under multitrueth assumption.

The history of misinformation can be traced back thousands of years ago when human started communications with different means. In the information age, social network sites have become a notable agent for the spread of misinformation, fake news, and propaganda and has attracted many data science researchers to study misinformation, its detection, and its handling. Some overviews of recent studies from data science point of view can be found at Berti-Équille and Borge-Holthoefter [BÉBH15], Shu and Liu [SL19], and Wu et al. [WMCL19].

### 12.2.3 Information and disease propagation

Data mining has been playing an important role in understanding information and disease propagation, mainly due to the availability of digital data trace at an unprecedented scale and speed. For example, Twitter was growing at a speed of 400M tweets per day in 2013, and Snapchat was producing 700M new photos and videos per day in 2014. The data mining research problems studied in this field can be divided into two categories, namely prediction problems and optimization problems. Underlying both prediction and optimization problems are the propagation models, which include information diffusion models and computational epidemiology models.

There are numerous applications of information and disease propagation. To name a few, for social media, it helps predict which piece of information (e.g., a Twitter post) is likely to go viral, detect the rumor source who started a misinformation campaign, and neutralize the propagation of misinformation before it goes viral (e.g., by appropriately disseminating a piece of true information or suspending the accounts of key spreaders, etc.). For computational epidemiology, it helps reveal the critical network condition (e.g., epidemic threshold) under which an epidemic is likely to happen.

**Prediction problems of propagation.** Various prediction scenarios exist for information and disease propagation, including the following:

*Classification and regression.* Basic classification and regression problems often aim at predicting the popularity of a piece of information via diffusion at a designated time in the future, by a predefined measure of popularity. The classification problem focuses on whether the information will be popular and the regression problem focuses on predicting the popularity score.

*Prediction around publication.* Predictions can be made before or soon after a piece of information is published. Before publishing, the available features and surrounding information are often very limited. For prediction after publication, the aim is to predict the future dissemination of the published information after observing the early stage of propagation, which provides more reliable information compared to prediction before publication.

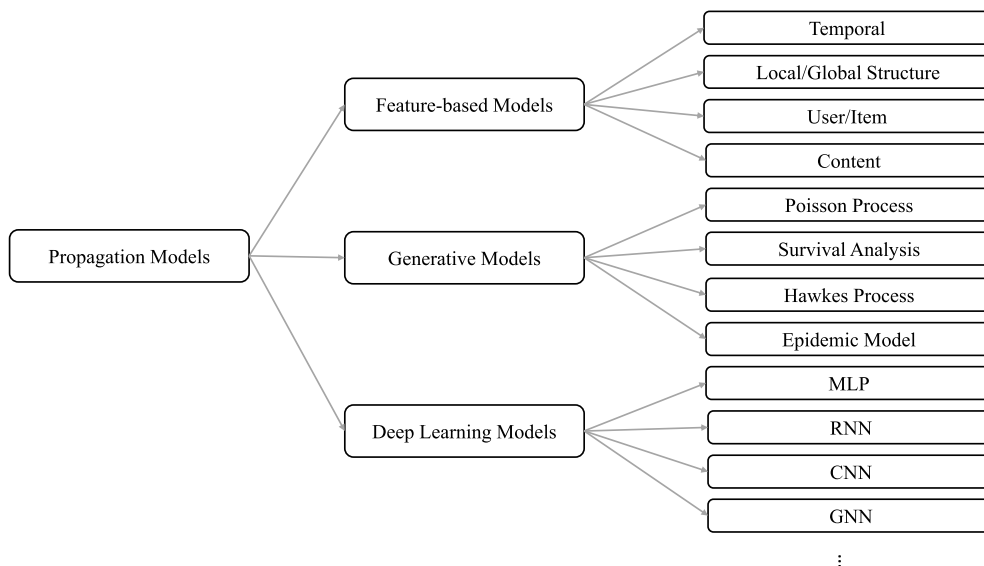
*Prediction at different granularities.* Information diffusion predictions can be conducted at different granularities of information and users. On a finer granularity, predictions can be performed on an individual piece of information or a user. On a coarser granularity, predictions can be performed on groups of users or clusters of information. Furthermore, in a more complex scenario, predictions at different granularities are performed simultaneously.

**Optimization problems of propagation.** Here, we discuss some representative optimization problems of information diffusion, usually in a (social) network scenario.

*Influence maximization.* The goal of influence maximization is to maximize the average or total number of infected nodes by choosing an optimal set of initially infected nodes to start the cascade process. Key challenges include how to incorporate the time information and how to handle model scalability.

*Source localization.* Generally speaking, the goal of source localization is to identify the source from the network with a partial observation of the cascade. A typical source localization method has two stages. First, the diffusion model parameters are inferred from historical cascade data. Second, given the diffusion model and possibly incomplete cascades, the source is identified. When the observed cascades are incomplete, which is often the case in some real applications, the formulated optimization objective for source localization is often difficult to solve.

*Activity shaping.* The activity shaping aims at steering users' activity. From the data mining perspective, the formal definition could vary, depending on the actual incentives and the specific goals. Compared

**FIGURE 12.4**

The taxonomy of information propagation models. Adapted from Zhou, Xu, Trajcevski, and Zhang [ZXTZ21].

with the influence maximization problem, activity shaping is different in the following three aspects. First, activity shaping has variable incentives, whereas influence maximization often only has a fixed incentive. Second, activity shaping usually uses multiple actions at multiple times, whereas influence maximization often uses the same action at a time. Third, activity shaping might include various objectives whereas the goal of influence maximization is primarily to maximize influence.

*Graph connectivity optimization.* The connectivity of the graph through which the information or disease is propagated has an profound impact on the propagation outcome. Graph connectivity optimization aims at manipulating the graph topology to affect the information propagation results. For example, a typical instance of graph connectivity optimization is to maximize (or minimize) the dissemination of information on graphs, by adding (or removing) a set of edges or nodes, under a fixed budget.

**Models for information diffusion problems.** A taxonomy of the propagation models for information diffusion problems is shown in Fig. 12.4.

*Feature-based model.* The feature-based model utilizes supervised data mining models for information diffusion prediction with various features, including temporal features, local and global structure features, user and item features, information content features, and so on.

*Generative model.* Many information diffusion processes can be regarded as event sequences in the continuous temporal domain, so that they can be naturally formulated as statistical generative approaches, such as epidemic models and various stochastic point processes. Representative generative models include the Poisson process, survival analysis, Hawkes process, epidemic model, and so on. Specifically,

for the Poisson process, it is often used as a reinforced Poisson process, where the reinforcement mechanism is added along with fitness of an item and temporal decay functions. For survival analysis, it is adopted for characterizing the information diffusion process by specific design of the hazard functions and survival probabilities. For the Hawkes process, it is generally difficult to distinguish cascades in the event data. The key idea to tackle this issue is to model each user's events as a counting process. Based on users' action sources, the activities are divided as exogenous (external to the network) and endogenous (user-user interaction) activities, and the exogenous intensity could be modeled as a Hawkes process. Epidemic models can often be used with self-exciting point process-based models to predict the rate of certain events as a function of time and the previous history of the events.

*Deep learning-based model.* Recently, the deep learning models have been adopted for information diffusion problems. Compared with traditional models, deep learning models do not have assumptions on the information diffusion process such as the generative mechanisms, so they are usually quite flexible. Moreover, deep learning models are able to incorporate multimodal data (such as text and images) with various neural modules (e.g. RNNs and CNNs). This direction has received tremendous attention, and it is developing rapidly.

**Computational epidemiology models.** There are many computational models in epidemiology for modeling the spread of disease, such as the well-known susceptible-infectious-susceptible (SIS), susceptible-infectious-recovered (SIR), and susceptible-exposed-infectious-removed (SEIR) models and their variants. The basic SIS model contains only susceptible (S) and infectious (I) states, with state transfer probabilities between these two states. As one of the most popular models, SEIR has four states in which an individual could be in, namely susceptible (S), exposed (E), infectious (I), and recovered (R). The states could only transit from  $S \rightarrow E \rightarrow I \rightarrow R$ . The transition between different states could be represented as a nonlinear dynamic process. The core tasks of modeling an epidemic case include (1) designing an appropriate epidemiology models with targeted variables for estimation (e.g. modifying SEIR by adding additional states and transition rules) and (2) estimating the targeted variables by solving the corresponding partial derivative equations of the nonlinear dynamic process. Note that a real-world epidemics is often more complex than these basic epidemiology models (e.g., incubation period, policy impact, etc.). Furthermore, the partial derivative equations often cannot be solved directly. Instead, the variables are estimated by the observed data or supervised data mining methods.

The research in this area is fast growing with many crucial challenges and promising future directions. For example, epidemics modeling has drawn vast attention since the breakout of COVID-19. Recently proposed techniques heavily concentrate on predicting transmission rate, number of death cases, and influences of government policies in controlling propagation, in order to assist people and the governments in understanding and acting during the global pandemic. Source localization and activity shaping still face many technical challenges. For example, how can we identify source with a small number of cascades, and how do we model more complex shaping behaviors? Furthermore, the predictability and interpretability of the propagation models still need further exploration. To name a few, what is the fundamental predictability of a prediction model? To what extent can a model predict information popularity? How do we interpret the end-to-end deep models for various prediction tasks?

### ***Bibliographic notes***

As one of the most influential works on influence maximization, Kempe, Kleinberg, and Tardos [KKT03] establish the first provable approximation guarantees for efficient algorithms in a social net-



work scenario. Gomez-Rodriguez et al. [GRSD<sup>+</sup>16] show that finding the set of source nodes that maximizes influence in the continuous time is NP-hard. To tackle this issue, Gomez-Rodriguez et al. [GRSD<sup>+</sup>16] find that the influence function bears submodularity property. Therefore one could maximize the submodular function of the original influence function by a greedy algorithm (ConTinEst), with  $\sim 63\%$  provable optimality guarantee. After Gomez-Rodriguez et al. [GRSD<sup>+</sup>16], Tong et al. [TWTD16] introduce a dynamic independent cascade model and propose an adaptive influence maximization method in dynamic social networks. Tang et al. [TSX15] develop IMM that achieves the state-of-the-art approximation guarantee and empirical efficiency with a novel algorithm design based on martingales.

For source localization, Farajtabar et al. [FRZ<sup>+</sup>15] propose a sampling strategy with two auxiliary distributions with the help of the diffusion model in order to approximate the incomplete cascade likelihood. It has been shown that it is difficult to locate sources when the number of cascades is small. Recently, Chen, Tong, and Ying [CTY19] study the problem of reconstructing the entire history of a diffusion process instead of only identifying the source of diffusion. A comprehensive survey is conducted by Yu and Jian [YP19].

For activity shaping, Farajtabar et al. [FDR<sup>+</sup>14] represent the overall activity as the summation of exogenous activity (by external factors) and endogenous activity (by user-user interactions); the exogenous intensity and the average overall intensity have a linear relationship, which opens the door to many optimization formulations for different activity shaping tasks. Recent research on pandemic modeling and controlling includes (Car et al. [CBŠA<sup>+</sup>20]; Ardabili et al. [AMG<sup>+</sup>20]; and Poirier et al. [PLC<sup>+</sup>20]). Zhou et al. [ZXTZ21] provide a comprehensive survey on feature-based, generative, and deep learning models of prediction problems for information diffusion.

### 12.2.4 Productivity and team science

Recent years have witnessed an increasing interest in understanding the performance of a team, the productivity of its team members, and the impact of the content that team produces. Teams, defined as a group of people with different roles and positions, serve as an integral function where members work collaboratively to achieve particular goals. In modern organizations (e.g., technology companies or government), it has become common for the organization to depend on a hierarchical structure of teams to boost productivity. More often than not, teams are often embedded in or operate on an underlying network such as a communication network or a social network.

**Example 12.2.** Different types of teams are observed in real-world scenarios. For example, in film production, directors, actors and actresses, designers, and editors collaborate with each other to make a movie within a period of time (Fig. 12.5(b)). In sports such as football or basketball, players and coach fight towards winning games with the support of other members like a physical therapist (Fig. 12.5(a)). Researchers or software developers that coordinate on a specific research project or product can also be viewed as a team where the members share the same goal and possess necessary skills. Furthermore, these teams can be embedded in a network. Specifically, for a movie crew network, two actors or actresses are connected if they have participated in the same film in the past. The attributes of movie crew network can be the genres (e.g., sci-fi, comedy, action) of movies that an actor or actress participates in. Likewise, for a research team, we can consider the coauthorship as the network structure and the expertise as the members' attributes (e.g., machine learning, system, computer vision).  $\square$

**FIGURE 12.5**

Examples of teams.

Data mining techniques have been leveraged to answer the following key questions in team science, including (1. team performance characterization) how to reveal the key characteristic patterns that differentiate a high-performing team from a struggling one; (2. team performance prediction) how to forecast the performance of the team before or soon after the start of the task; (3. team performance optimization) how to further enhance the team performance by adjusting the team composition; and (4. team performance explanation) how to interpret the team performance prediction and optimization results in an intuitive way.

**Team performance characterization.** In general, the key components in a team include (1) team leader and members, (2) the environment (e.g., networks) in which team members collaborate, and (3) the task on which the team works. Various patterns have been discovered to have a strong correlation with team performance, such as the collective intelligence as an outcome of collaboration and the average performance of top- $k$  members. Challenges in characterizing the team performance arise from all these three components. First, different team members possess different types of skills and social connectivity. Second, the nature of task varies depending on the specific application scenarios, such as collaborative tasks for research teams vs. competitive tasks in team sports. Third, the environment (e.g., networks) that the team is embedded in or operates on itself is often large in size, highly volatile in temporal dynamics, noisy and incomplete. Furthermore, it is also challenging to accurately quantify the team performance. For instance, it is very difficult, if not impossible, to find a single metric to precisely measure the performance of a research team. Instead, we often have to rely on a set of proxy performance measures, such as the citation counts of publications,  $h$ -index of members, and download counts. All these factors have precipitated the team performance characterization into a challenging problem. Existing literature characterizes the team performance as the outcome of collective intelligence from both virtual teams (e.g., online games) and teams with face-to-face interactions (e.g., sports or research teams).

**Team performance prediction.** Accurately forecasting the team performance is a key stepping stone to understand the underlying principles of constructing a high-performing team. In academia, it is desirable to predict the long-term performance of a scholarly entity (e.g., a researcher). An effective team performance prediction algorithm should be able to (1) identify crucial features (e.g., coauthorship network topology, research topics), (2) model the correlation between identified features and the team performance, and (3) encode the dynamics of team evolution. Complementary to the team performance prediction, it is also important to predict the impact of the content that team produces at a finer

granularity (e.g., forecasting the citation count of a research paper in each of the next 10 years upon its publication) by modeling the temporal correlation between content impact and the team dynamics. Furthermore, a team and its team members collectively constitute a specific instance of *part-whole* relationship, where the team is the whole entity and its team members are part entities. The part-whole relationship often goes beyond the linear correlation. For example, the performance of the team (the whole entity) is often not simply the (weighted) sum of the productivity of its members (part entities). By modeling the nonlinear correlation between the team performance and member productivity, it often leads to further prediction performance improvement.

**Team performance optimization.** In many real-world applications, the team leader often needs to optimize the team composition to maximize its performance. For example, during an NBA game, the rotation between players is dependent on the current strategy and the physical conditions of players, and we refer to this substitution as the team member replacement. Here, a key insight is that an effective team member replacement algorithm should consider not only the skills of the team members but also the network connectivity. In other words, the similarity between the teams before and after the replacement should be measured in the context of the network that the team operates on. Generalizations of such an approach have been used for other team optimization scenarios. To name a few, if the team leader wishes to downsize the team because of budget cut, a possible solution is to select a member whose departure has the least impact on the original team (i.e., team shrinkage); in other scenarios, we might want to bring in new team members with certain skills and collaboration structure according to the requirements of new tasks (i.e., team expansion); and if two or more team members do not get along in an existing team, we might consider to swap one of them with another team (i.e., team conflict resolution). Furthermore, real-world teams are complex and dynamic systems with time-evolving configuration and the team performance is likely to change over time. Hence, it is critical to model the dynamic correlations between the team performance and the team optimization strategies in order to sustain a high-performing team in real time (i.e., real-time team optimization), where reinforcement learning (introduced in Chapter 7) might provide an effective solution.

**Team performance explanation.** The vast majority of the literature on team performance prediction and optimization aims to answer questions like *which team* is most likely to succeed, *who* is the best candidate to replace a departure member in the team, or *what* is the best strategy to expand the team. On the other hand, the intuitive explanations for *why* the team performance prediction algorithm “thinks” a given team will succeed or struggle, or *why* the team optimization algorithm recommends a particular action for a given scenario are largely absent. The sparse literature on explainable team performance prediction and optimization often resort to *influence function*, a technique rooted in robust statistics, to identify key elements (e.g., team members, skills of team members, the connectivity between different team members) to interpret the team performance prediction or optimization results. For example, given the key components in a team (i.e., members, networks, tasks), the performance prediction results can be interpreted from the aspects of multiple components (e.g., which tasks are more critical compared to others?). The multilevel interpretation offers a comprehensive understanding of the prediction results. In terms of explaining team performance optimization, existing methods identify the network elements (e.g., edges and nodes) that are influential to the team optimization results. For instance, in team member replacement, the best candidate may have a nearly identical collaboration structure and possess the same important skills as the departure member does, which makes the candidate a favorable replacement.

### ***Bibliographic notes***

The definition of team is formally introduced by Hackman and Katz [HK10], which states that teams function as “purposive social systems” for collective objectives. Different types of teams exist in real-world applications, such as GitHub teams (Thung, Bissyand, Lo, and Jiang [TBLJ13]) and sports teams (Duch, Waitzman, and Amaral [DWA10]).

In team performance prediction, Uzzi, Mukherjee, Stringer, and Jones [UMSJ13] aim to forecast the impact of research works by evaluating the atypical combination of prior work, and Yan et al. [YTL<sup>+</sup>11] propose to leverage effective content and contextual features for citation counts prediction. Li and Tong [LT15] propose a joint predictive model for long-term scientific impact prediction under a more complex scenario with nonlinearity among feature and prediction, network dynamics, and domain heterogeneity. For performance trajectory forecasting, Li, Tong, Tang, and Fan [LTTTF16] introduce a new predictive model that can simultaneously fulfill two requirements, including prediction consistency and parameter smoothness. To further understand the relationship between individual members and the final outcome of the team, Li et al. [LTW<sup>+</sup>17] jointly model the part-whole correlation and the part-part interdependency.

In team performance optimization, existing works can achieve good performance in the task of static team member replacement using effective and efficient graph similarity based algorithms (Li et al. [LTC<sup>+</sup>15, LTC<sup>+</sup>17]) and dynamic team formation (Zhou, Li, and Tong [ZLT19]). In addition, Li, Tong, and Liu [LTL18] propose to interpret the networked prediction results from a multilevel perspective. For team optimization results, Zhou et al. [ZLC<sup>+</sup>18] aim to provide intuitive explanation for optimization results through visualization technique. Li and Tong [LT20] provide an introduction of computational foundation for network science of teams ranging from prediction, optimization and interpretation.

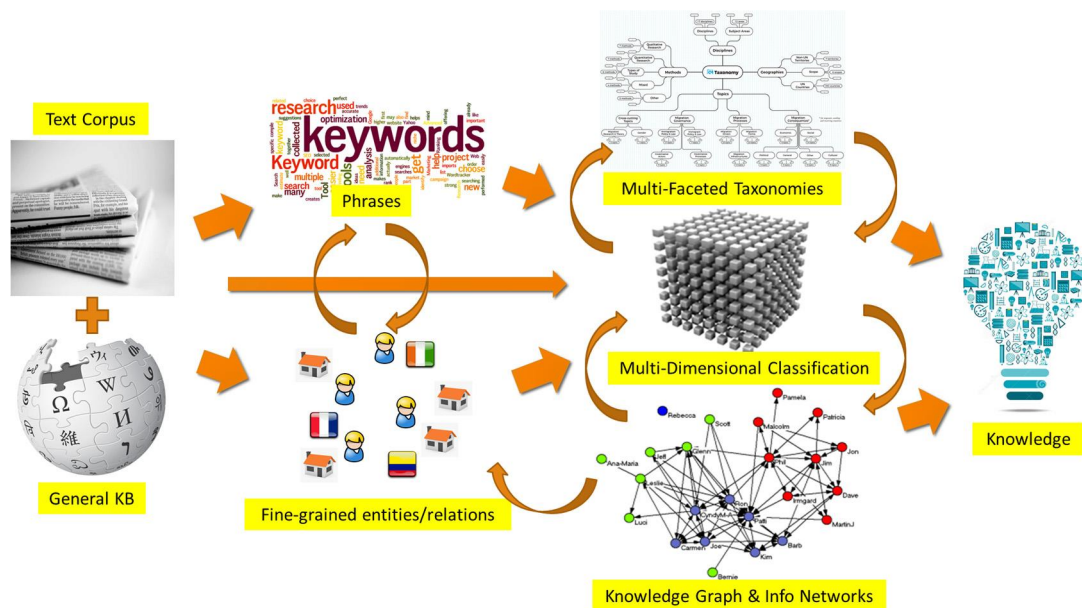
---

## **12.3 Data mining methodologies and systems**

### **12.3.1 Structuring unstructured data for knowledge mining: a data-driven approach**

With massive unstructured data stored or streaming in dynamically, an important methodology for turning data to knowledge is to systematically transform unstructured text-rich data into organized, relatively structured data so that knowledge can be extracted effectively based on a user’s requests. Among different approaches on turning unstructured data to structured knowledge, we promote a distantly supervised, data-driven approach, as outlined in Fig. 12.6. The essence of this approach is to make good use of available knowledge-bases such as Wikipedia, domain-specific dictionaries, and pre-trained language models computed from massive corpora, explore the power of distant-supervision or human-guided weak supervision, and conduct information extraction, taxonomy construction, document classification, knowledge graph, or information network construction and enrichment. Such information-rich structure will assist knowledge discovery in massive text. Although many of these functional components are still under active research and development, we overview some essential progress or major ideas in the following discussion.

**Taxonomy construction and refinement using massive text corpora.** Taxonomy organizes important concepts into semantically rich structures and may play an essential role at organizing massive unstruc-



**FIGURE 12.6**

Structuring text to knowledge: A data-driven approach.

Structured text data into relatively organized structures. There are a lot of largely human-curated taxonomies available in different domains, such as science, engineering, bio-medical, and business. However, a suitable taxonomy that fits a particular application should be multifaceted and corpus- and application-dependent. Thus it is often desirable to (i) generate a corpus- or application-dependent new taxonomy, based on available corpus and application demands; (ii) refine an existing taxonomy, which could be initially constructed/provided by domain experts but is outdated or unfit to a particular application; or (iii) expand a human-provided, incomplete taxonomy skeleton, based on the corpus and expected applications. In all these cases, human-extensive annotation could be costly and inconsistent, and it is often desirable to develop a weakly supervised or distantly supervised approach to do this automatically or semiautomatically (e.g., interacting with human experts).

In recent years, weakly or distantly supervised methods have been developed for taxonomy generation and expansion. For example, a Rank-Ensemble method has been developed in set expansion to automatically expand from a small set of user-provided seeds, representing a set of items of similar semantics (e.g., a set of US states) and generate additional entities in the same set (e.g., the remaining US states). Such set expansion methods can be further enhanced by parallel expansion of the negative sets (to guard each other and avoid semantic drifting) or by exploring the power of pretrained language models. Further, taxonomy can be generated by starting with a small sample taxonomy and conducting both depth and width expansion to form a more complete hierarchy or generated by embedding-based hierarchical clustering. Moreover, a taxonomy can be extended by adding new emerging terms or delet-

ing old, obsolete ones. Appropriate taxonomy modification can be done incrementally by evaluation of the semantic consistency and balance of the updated taxonomy.

**Weakly supervised text classification.** Text could be massive, diverse, and in multiple granularities, and it is costly to rely on human to annotate text data in complex and dynamic environment. Although there are many supervised methods for document classification, it is often realistic and desirable to rely on weakly supervised text classification with a small set of labeled data and a massive set of unlabeled text data. Given a small set of labels, which can be informative category names, or some human-provided keywords or labeled documents or their combinations, the key challenge becomes how to enlarge the seeds based on the massive set of unlabeled data. Several effective weakly supervised text classification methods have been explored: (i) use embedding methods to generate pseudo-documents and then train neural networks based on the generated pseudo-documents and the large set of unlabeled text data (e.g., in WeSTClass); (ii) use category-guided embedding to generate class-distinctive keywords or phrases (which enlarges the seed set effectively) (e.g., in CatE); or (iii) use pretrained language model (e.g., BERT) to generate class-distinctive keywords to improve the quality of weakly supervised classification (e.g., in LOTClass).

In many real-world applications, text classification may need to consider a large number of classes (e.g., potential themes of a research paper) and multiple labels (e.g., a paper can be tagged by a set of tags or themes). Massive human-labeled documents are often too costly to obtain. Fortunately, a large set of classes are often organized into a taxonomy. A taxonomy-based, hierarchical multilabel text classification method can be developed (e.g., TaxoClass) to tag each document with a set of classes from a class hierarchy, as follows. First, class surface names are represented by nodes in a taxonomy, and such a skeleton structure can be used to generate category-distinctive keywords or phrases (e.g., using hierarchical text embedding methods) and be used as supervision signals. Second, a multiclass classification scheme may first identify a few most essential classes for a document as its “core classes,” and then check the parent/ancestor classes of the core classes to generate remaining related tags. Third, it is easier to conduct a top-down search for the right classes since there will be only a small number of the candidate classes that need to be considered at the top-layer, and the search for “core classes” can follow the most likely high-level nodes to walk down the taxonomy as promising paths. Finally, it is beneficial to calculate document-class similarities using a textual entailment model, identify a document’s core classes, utilize confident core classes to train a taxonomy-enhanced classifier, and generalize the classifier via multilabel self-training.

**Fine-grained information extraction via context-aware distant supervision.** Instead of identifying an entity belonging to one of several major types (e.g., person, organization, location, time), it is often more useful to conduct fine-grained entity recognition to extract the concrete role that an entity plays in a particular context (e.g., Trump can be a businessman, president, or ex-president in different contexts). For identification of the context of an entity in a sentence or paragraph, it is good to first use an effective data-driven phrase mining method to generate entity mention candidates and relation phrases, and then conduct distant supervision to find the most appropriate fine-grained types for the entities to be examined. Distant supervision may come from the “type-labeled” entities in Wikipedia, domain-specific dictionaries, or other knowledge bases. One method, ClusType, enforces the principle that relation phrases should be softly clustered when propagating type information between their argument entities and jointly optimizes two tasks, type propagation with relation phrases, and multiview relation phrase clustering to achieve good performance. Embedding can also be explored in type inference and relation phrase clustering. Moreover, one can explore taxonomy-guided supervision and pretrained language

model for fine-grained entity recognition. Its general philosophy is to use a human-selected taxonomy as guidance to generate additional terms/phrases belonging to the corresponding node classes in the taxonomy, based on dictionaries, embedding, and pretrained language models. The taxonomy so enriched with additional phrases can be used for taxonomy-guided text classification and for fine-grained named entity recognition. When an entity may have multiple potential types, the one that fits the local context best, according to the taxonomy information, will be given the highest confidence to the corresponding fine-grained type.

**Knowledge graph/information network construction.** Knowledge graphs and information networks are important structures to help turn unstructured data into structures and knowledge. Knowledge graphs consist of a set of entities, associated with their corresponding attributes and values, and a set of (possibly labeled) edges linking among entities. The graph can be further structured with taxonomy information and with conditions/probabilities associated with edges or attributes/values, indicating under what condition/probability such a relationship holds. A heterogeneous information network may not treat one entity (e.g., an author) as the attribute value of another entity (e.g., a paper) but treat them as heterogeneously typed entities linking together via a labeled edge (e.g., wrote).

It is useful to use distant or weak supervision for construction of knowledge graphs/information networks from massive text. Taxonomy-guided text classification may allocate text to the corresponding nodes or subgraphs. This will help extraction of information related to particular entities and their associated attributes/values with fine-grained entity recognition. With massive text data, it is likely that an entity can be associated with different attributes/values and linked to different other entities in different times or conditions. Without clear distinction of appropriate conditions, it is easy to cause confusion if such different associations are merged into a single “global” knowledge graph. Therefore in many cases, it is often more useful to construct local knowledge graphs from a set of documents corresponding to specific situations and use such local knowledge graphs under similar conditions.

### ***Bibliographic notes***

Lots of research have been contributing to this important research frontier: structuring unstructured data for knowledge mining. Good progress has been made on data-driven, weakly, or distantly supervised approach in recent years (e.g., Wang and Han [WH15]; Liu, Shang, and Han [LSH17]; Ren and Han [RH18]; and Zhang and Han [ZH19]). This includes (i) weakly/distantly supervised or unsupervised phrase mining for extraction of informative entities from massive unstructured text (e.g., AutoPhrase by Shang et al. [SLJ<sup>+</sup>18] and UCPhrase by Gu et al. [GWB<sup>+</sup>21]), (ii) distantly supervised or ontology-guided fine-grained NER or pretrained language model-based NER (e.g., ClusType by Ren et al. [REKW<sup>+</sup>15], ChemNER by Wang et al. [WHS<sup>+</sup>21], and RoSTER by Meng et al. [MZH<sup>+</sup>21]), (iii) embedding-based taxonomy construction and expansion (e.g., TaxoGen by Zhang et al. [ZTC<sup>+</sup>18], HiExpan by Shen et al. [SWL<sup>+</sup>18], SetCoExpan by Huang et al. [HXM<sup>+</sup>20], TaxoExpan by Shen et al. [SSX<sup>+</sup>20]), and (iv) weakly supervised and/or ontology-guided text classification (e.g., WeSTClass by Meng, Shen, Zhang, and Han [MSZH18], LoTClass by Meng et al. [MZH<sup>+</sup>20a], and TaxoClass by Shen et al. [SQM<sup>+</sup>21]).

### **12.3.2 Data augmentation**

A high-performing data mining model often requires a great amount of labeled data samples for training. However, for many domains, due to the data privacy, expensive labor cost, data imbalance, and ever-growing new data, we might be only provided by a limited number of labeled data. For example,

in the medical image processing domain, an important data resource is from magnetic resonance imaging (MRI) whose average cost is more than 2000 dollars in the United States. Besides, many patients, out of concern of their privacy, might be unwilling to provide their MRI images for data analysis. Hence, for a wide range of real-world applications, we have to train a data mining model with scarce training data, which could lead to the overfitting issue. A myriad of solutions are proposed, including those introduced in Chapter 7 (i.e., classification with weak supervision). Here, we briefly introduce another promising technique called **data augmentation**. In general terms, the core idea of data augmentation is to enrich the training data set to help the data mining and machine learning models extract meaningful features for generalization. Roughly speaking, data augmentation methods can be categorized in basic methods and learning-based methods.

The key idea of data augmentation is to utilize the *invariance* property of data samples. Let us look at some examples to illustrate the invariance property of different types of data.

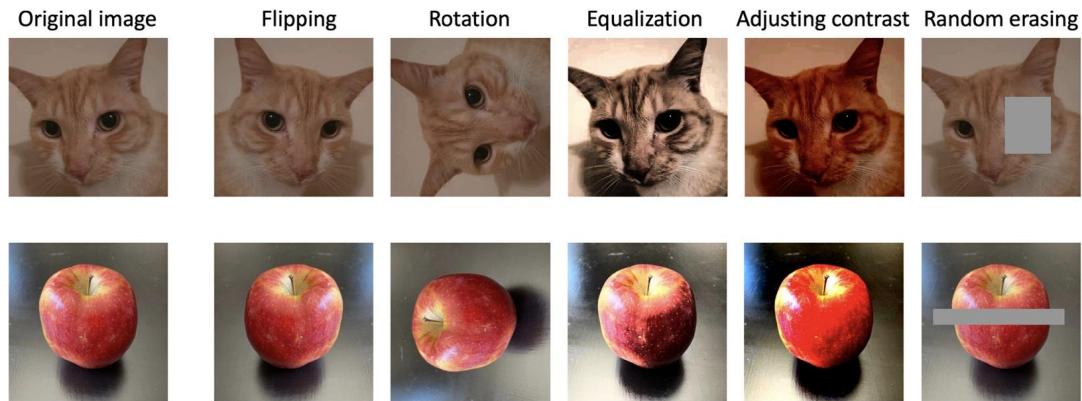
**Example 12.3.** For an image classification task, the human visual system can recognize a cat from a wide variety of angles; for audio data, most speech can be recognized with the partial loss on time domain and frequency domain; and for network data such as social networks, the hidden user profiles for a group of users do not vary significantly by adding a few new relationships with another group of users. □

**Basic methods.** Thanks to the strong generalization ability and robustness of human recognition systems, basic data augmentation methods develop a set of operations that can be verified by the human recognition systems to generate augmented samples. By augmenting training samples, the data mining models are expected to be as robust as the human recognition systems and therefore are capable of extracting features with greater generalizability. For instance, for most cases, a flipped image has the same label as the original image and can be added into the training set with the same label as the original image. Other basic image augmentation methods include adjusting the color space, rotation, erasing part of the images, and many more. We provide several examples for basic image augmentation methods in Fig. 12.7.

As the invariance property exists in various types of data, the basic augmentation methods can boost a wide range of tasks as well. For instance, the audio with randomly erased pieces can serve as the augmented audio recognition data. Based on the knowledge of basic augmentation methods, next, we further introduce three typical learning-based augmentation methods, including augmentation policy learning, generative adversarial nets (GANs)-based methods, and adversarial training.

**Augmentation policy learning.** Most of the basic data augmentation methods are centered around the guidance of human recognition systems, which might make the workflow somehow ad hoc and suboptimal. For instance, to achieve a more comprehensive augmentation, some works combine multiple basic augmentations together as *augmentation policy*. However, not all the policies are equally informative. Intuitively, for example, the samples augmented by “flipping+rotation” might not be as informative as the samples augmented by “rotation+equalization.” To automatically search for the best augmentation policy, existing works such as AutoAugment define the augmentation policy by a sequence of subpolicies, each of which is composed of a specific operation and the corresponding magnitude. For example, an augmentation policy can be divided into two subpolicies; the first subpolicy is to *rotate* the image by  $x$  degrees, and the second subpolicy is to *equalize* the image with magnitude  $y$ . Then, a parameterized policy selection model (e.g., recurrent neural networks) searches the combination of different subpolicies as the candidate policy. Finally, AutoAugment formulates it as a meta-learning problem



**FIGURE 12.7**

Examples of basic augmentations on image data.

whose procedure is as follows: (1) the policy selection model selects a set of augmentation policies to enrich the data set; (2) another target model for a specific task (e.g., convolutional neural networks for image classification) is trained with the augmented data; and (3) it validates the performance of the well-trained target model (by a validation set) and provides feedback to update the policy selection model.

**GANs-based methods.** With the augmentation policy selection model, the augmentation procedure can be implemented automatically and more accurately. However, users of the system still need to assign specific basic augmentation methods to construct the search space, and the effectiveness of such a system might also be limited by the power of basic augmentation methods. In fact, a wide range of transformations can retain the invariance of data labels but nonetheless cannot be represented by the basic augmentation methods effectively. For example, a running cat and a sleeping cat should both be labeled as “a cat” however, we cannot apply any combinations of the basic augmentation methods to transform a running cat image into a sleeping cat image. The development of GANs provides a new solution toward the data augmentation problem. The full details of GANs are outside the scope of this textbook. In a nutshell, the training of GANs can be viewed as a cat-and-mouse game between its two components, including the generator and the discriminator, where the generator tries to mimic the distribution of real data samples, and the discriminator is optimized to effectively tell the generated samples and the real samples apart. Hence, for data scarcity scenarios, especially data imbalance scenarios, GANs-based data augmentation method can enrich the classes with limited data samples by learning from the classes with abundant samples. To be specific, by training GANs with the classes of abundant samples (e.g., running dogs and sleeping dogs), the generator learns to fool the discriminator by importing versatile posture. Then if the user change the input of generator from dog images to cat images, the generator is expected to generate both running cats and sleeping cats. Valuable transformations such as changing the postures of animals in images are hard to be covered in the previously-introduced methods. In contrast, GANs-based methods can handle it effectively. It is worth mentioning that this line of methods is closely related with the “imagination” (or “hallucination”) concept in the few-shot learning scenario. Interested readers can refer to the references in the bibliographic notes.

**Adversarial training.** Another related work of learning-based data augmentation is adversarial training whose intuition shares several common grounds with the GANs-based methods. The motivation of adversarial training lies in the fragility of great quantities of learning models. Both empirical and theoretical results illustrate that unnoticeable perturbations toward the original data samples can often change the output of data mining models dramatically. However, every story has two sides and by mixing the trickily perturbed samples into the training samples, the trained models have been proven to be more robust compared with the models trained with original training samples.

To make a comparison, the basic augmentation methods and the augmentation policy learning methods require predefined basic operations (e.g., flipping, rotation for image data) to guide the augmentation; augmentation policy learning, GANs-based methods, and adversarial training resort to parameterized components that are often learned from abundant data. For augmentation policy learning methods, they might suffer from the efficiency problem since every updating of the policy selection model requires the feedback from a retrained target model on a set of augmented samples. As for GANs-based augmentation methods, in principle, the augmentation can be viewed as an instantiation of *transfer learning* by which the models learn the manifold from classes with plenty of data samples to construct the manifold of classes with limited number of data samples. However, in many tasks (e.g., medical imaging processing), it is hard to find a class with plenty of data samples and how to design effective transfer learning mechanism from other data-rich domains is the key challenge. For adversarial training, although it has intrinsic connection with data augmentation methods, there are some subtle differences: (1) samples “augmented” by adversarial training are nearly the same as the original samples for the human recognition system since they are perturbed “unnoticeably”; and (2) the goal of adversarial training is to improve the robustness of models (against adversarial attack), whereas the goal of data augmentation is to improve the generalizability of models.

### ***Bibliographic notes***

AutoAugment developed by Cubuk et al. [CZM<sup>+</sup>18] is the pioneering work that automates the augmentation procedure but requires a great amount of computing resources. For the family of GANs-based methods, DAGAN proposed by Antoniou et al. [ASE17] is the first work to incorporate GANs model into the data augmentation task and has inspired a great number of GANs-based works in various applications such as machine fault diagnosis by Shao, Wang, and Yan [SWY19] and medical image analysis by Shin et al. [STR<sup>+</sup>18]. The “hallucination” strategy developed by Wang et al. [WGHH18] for the few-shot learning scenario is closely related with the GANs-based methods. Shorten and Khoshgof-taar [SK19] provided a comprehensive survey about the augmentation for image data.

For other data types such as audio data, Park et al. [PCZ<sup>+</sup>19] developed SpecAugment to obtain strong performance improvement by augmentation from frequency and time domains. The augmentation of graph-structured data is still underexplored. One of the GANs-based solutions, GraphSGAN developed by Ding et al. [DTZ18], showed that data augmentation can benefit semisupervised learning on graph data effectively. You et al. [YCS<sup>+</sup>20] augmented the graph data by predefined basic operations to improve the performance of graph representation learning.

### **12.3.3 From correlation to causality**

Data mining models have obtained dramatic achievements in many applications, ranging from stock market prediction to recommended system. However, most of the existing data mining methods are

essentially correlation analysis and much less has been done on causality analysis. In general terms, correlation analysis is to study the statistical associations between different observed variables, whereas causal analysis focuses on the causality relation between them. We use the following example to illustrate the difference between these two kinds of analyses.

**Example 12.4.** Suppose the temperature is low in winter. Both the electric bill and food expense are very high. In this scenario, we have observed three variables, including low temperature, high electric bill, and high food expense. There exists a correlation between the high electric bill and the high food expense. However, it is not the high electric bill that has caused the high food expense and vice versa. The causality might instead lie in the low temperature for both the food expense and the electric bill. □

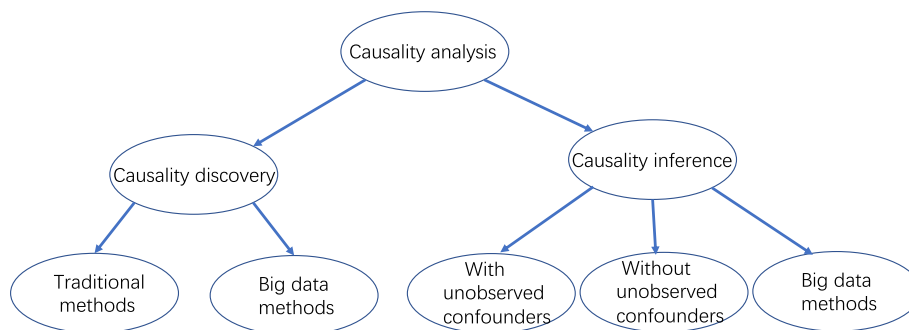
Compared with correlation analysis, the challenges for causality analysis could be explained by its main tasks, including causality inference and causality discovery.

- *Causality discovery* is qualitative: analysis. For example, if we want to change the value of some specific variable(s), which variables should we manipulate?
- *Causality inference* is quantitative: analysis. For instance, if we modify some variable's value, what kind of quantitative changes will happen to some specific variable(s)?

For causality discovery, it is more difficult than correlation analysis and may need some prior expert knowledge; as for causality inference, it needs well-designed complex experiments like A/B test, where other variables are kept same.

Causality analysis can be applied to different aspects of data mining models. For example, with a good causality discovery, the interpretability of black-box deep learning models will be enhanced. In some intelligent systems (e.g., university admission system), causality analysis can help avoid unfair results related to some sensitive variables (e.g., gender). In addition, causality analysis could improve the robustness of models.

The taxonomy of causality analysis is shown in Fig. 12.8. We divide existing methods according to their tasks. For causality discovery, traditional methods include constraint-based algorithms, score-based algorithms, and functional causal models. The *Peter-Clark (PC) Algorithm* tests the conditional



**FIGURE 12.8**

The taxonomy of causality analysis. Adapted from Guo et al. [GCL<sup>+</sup>20].

independence while generating possible causal graphs. Score-based algorithms evaluate the score of generated causal graph. In functional causal models, variables are formulated as output of functions with direct cause and noise as input. To tackle the challenge posed by big data, it has been shown that high-dimensional situation could be solved by adding restrictions on search space. For causality inference task, with the assumption that all confounders are observed, there exist three representative categories of methods, including regression adjustment, propensity score methods, and covariate balancing methods. Regression adjustment is based on counterfactual analysis to estimate the average treatment effect (ATE) score. Propensity score method has been designed to match each instance into a set, and ATE is estimated within each set. Instead of weighting instances with propensity scores, Entropy Balancing (EB) has been used to learn instance weight, which is a classical method in covariate balancing category. Usually, some unobserved confounders exist in real applications, which means back-door path in causal model can not be blocked by conditions. To handle these cases, instrumental variable (IV) and front-door criterion are proposed. In particular, the utilization of IV allows us to analyze the causal effect in two stages. The effect of relevant IV on treatment is analyzed in the first stage, followed by analyzing treatment's effect on outcome in the second stage. In front-door criterion, mediating variables are set to block paths from the treatment to the outcome, which makes it easier to conduct causality inference with unobserved confounders. With the support of big data, some advanced learning methods emerge for causality inference. Neural networks and ensemble models are applied to enhance causal model's representation power.

Current trends in causality analysis are twofold. In the era of big data, researchers aim to replace prior knowledge in traditional methods with data-driven strategy. Deep learning methods have become popular, which could be integrated in causal models to further improve the performance. Some problems are still open for causality analysis. For example, when the treatment is complex or varies with time, the causality analysis becomes much more difficult. Many applications of causality analysis such as black-box interpretation, model robustness, and fairness still wait to be explored.

### ***Bibliographic notes***

Spirtes et al. [SGSH00] first generate a skeleton casual graph and decide the direction of each edge later. Chickering et al. [Chi02] set additive Gaussian noise in the structural equations and calculate the score function of a causal graph based on the structural equations. Shimizu et al. [SHH<sup>+</sup>06] transform detecting causal relations into a lower triangle matrix estimation task. Nandy et al. [NHM<sup>+</sup>18] verify the theoretical consistency from a low-dimensional situation to a high-dimensional situation.

For causality inference task, Hirano et al. [HIR03] use inverse probability to construct the weight of different instances. Louizos et al. [LSM<sup>+</sup>17] adopt a variational autoencoder to learn latent representation of instances. Hill et al. [Hil11] utilize Bayesian Additive Trees to obtain the conditional average treatment effect (CATE). Shalit et al. [SJS17] focus on the individual causal effect and give a generalization bound with the help of neural networks. In Wager et al. [WA18], heterogeneous treatment effect is analyzed by a nonparametric random forest, which is also a representative ensemble model.

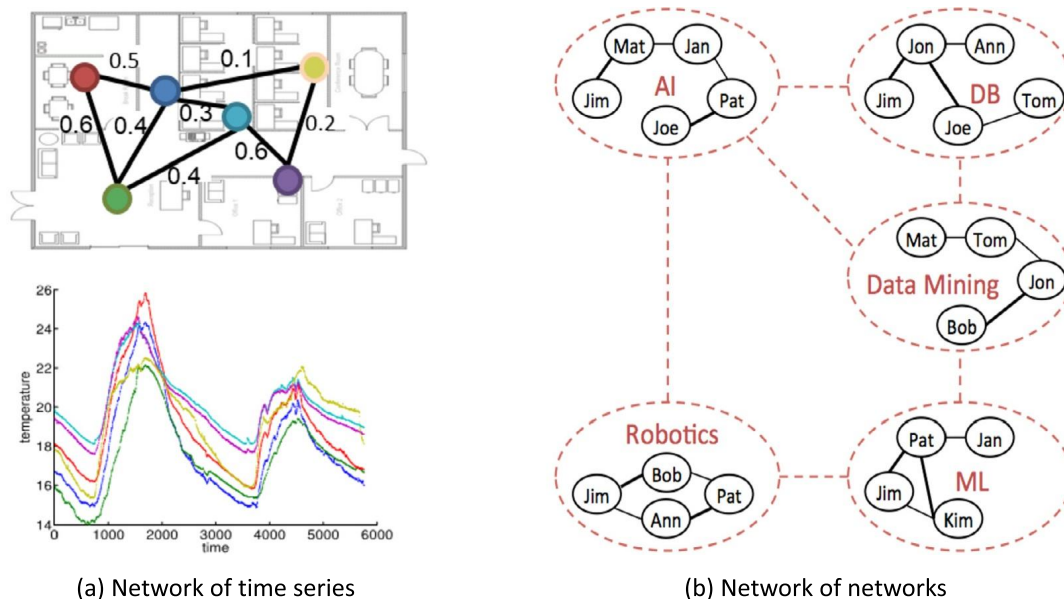
### **12.3.4 Network as a context**

Networks (i.e., graphs) not only appear in many high-impact application domains but also have become an indispensable ingredient in a variety of data mining and machine learning problems. Specifically, networks have not only become a ubiquitous data type (see Section 12.1.3 for some representative

works on mining network and graph data) but also provided a powerful *context* that links different types of data from different sources with different data mining algorithms. We refer to this phenomenon as *Network-of-X*, where different *Xs* (i.e., entities, data sets or data mining models) are interconnected with each other by an underlying *contextual network*. In other words, each node of the underlying contextual network is associated with or mapped to an *X* representing an entity, a data set or a data mining model. In particular, when each *X* represents an entity (e.g., a user, a web page, a device, etc.), Network-of-*X* is simply equivalent to the typical network or graph data we have seen in Section 12.1.3. When each *X* represents an entire data set or a data mining model, we can leverage Network-of-*X* to enrich the modeling power and boost the mining performance in various real-world applications. Let us look at some examples, including network of time series, network of networks, and network of regression models.

**Network of time series.** A typical example of network of time series is the complex surveillance system as follows.

**Example 12.5.** A complex surveillance system to detect invasions is often integrated with the devices that monitor multiple types of signals, including ratio frequency and temperature. These devices are connected by a contextual sensor network, and generate specific coevolving time-series data to monitor abnormal activities. Fig. 12.9(a) shows an illustrative example. Nodes represent different devices that are connected by edges indicating the similarities or correlations among devices. Each device monitors



**FIGURE 12.9**

Illustrative examples of network-of-*X*. (a) An example of a sensor network with nodes as time series of temperature (adapted from Cai et al. [CTF<sup>+</sup>15]). (b) An example of a research domain network with nodes as domain-specific collaboration networks.

a specific type of signals (i.e., time series). Here, the time series of temperature monitored by all devices are shown. □

In this example, each device (e.g., a sensor) provides signals (e.g., a time series) monitoring a specific aspect of activity, which alone might fall short in detecting abnormal activities. The underlying contextual network is the device-to-device network (i.e., a sensor network) whose edges capture the correlations between different devices. By mining multiple intercorrelated signals (i.e., multiple time series) together with the underlying sensor network, it could help detect the abnormal activities more precisely. The key idea for mining network of time series data is to (1) model each time series by either traditional signal processing approaches (e.g., Kalman filter) or deep neural networks (e.g., long short-term memory) and (2) leverage the contextual network to regularize different time-series models.

**Network of networks.** When each  $X$  (i.e., node) itself of the contextual network represents another domain-specific network, this corresponds to the network of networks model. Let us look at an example.

**Example 12.6.** Authors collaborate with each other to publish papers in different research domains (e.g., data mining, database, machine learning, etc.), which leads to a set of domain-specific collaboration networks shown in Fig. 12.9(b). These collaboration networks are connected by a research domain network whose edges could measure the similarities or correlations among different domains. Note that some authors may exist in more than one domain if they have multiple research interests. □

The main advantages of the network of networks model over other complex network models are mainly twofold. First, by connecting domain-specific networks with a contextual network, the model explicitly encodes the hierarchical structure such that the multiresolution characteristic of such complex data sets can be exploited for the mining tasks. Second, the contextual network provides additional regularization for the mining tasks by admitting a *cross-network consistency principle* upon the common nodes shared by different domain-specific networks (e.g., the same author who publishes papers in multiple domains).

**Network of data mining models.** When each  $X$  itself is a data mining model, we can use Network-of- $X$  to link multiple, potentially intercorrelated data mining models together. An example of team performance prediction (see Section 12.2.4 for a general introduction of data mining application in team science) is illustrated in Example 12.7. With the help of the underlying contextual network, different performance prediction models can “borrow” data from each other and thus mutually boost each other’s prediction performance. In this setting, the contextual network may work as a graph-based regularization term on the performance prediction model parameters.

**Example 12.7.** One crucial ingredient of the success of an organization is the coordination and collaboration among different task-specific teams (e.g., research teams, production teams, human resource teams, etc., in an IT company). The collaboration among teams can be modeled as a contextual network. With a regression model on each team to predict its performance, these regression models are connected by the contextual collaboration network. □

Network-of- $X$  model is still an underexplored area with many possible future directions. First, the contextual network construction is of key importance in the overall performance and a poorly constructed contextual network may even hurt the performance. For example, the contextual network is usually constructed by computing the domain similarities as edges of the contextual networks. However, it remains unclear what the “optimal” similarity measures are for a specific task. Second, in terms

of  $X$ , it has many other options so that the model can be applied to more applications, such as knowledge graphs from different domains or database, heterogeneous networks, and representation learning models. The third future direction is to integrate other data mining problems with network-of- $X$ , including adversarial learning, fairness, and explainable learning.

### ***Bibliographic notes***

For a network of time series, Cai, Tong, Fan, and Ji propose a dynamic contextual matrix factorization method to learn common latent factors behind the time-series and the contextual network structure [CTFJ15]. Li, Yu, Shahabi, and Liu propose a sequence-to-sequence architecture with diffusion convolutional gated recurrent units to capture both the spatial information behind the sensor network and the temporal information of the time series [LYSL17]. Similarly, other spatial-temporal forecasting methods can be also considered as the prediction methods on network of time series (Yu, Yin, and Zhu [YYZ17] and Geng et al. [GLW<sup>+</sup>19]). These models focus on the single-mode time series. To model more complex systems where each temporal snapshot of the coevolving time series is a multimode tensor (e.g., describing temperature, wind speed, etc.), tensor decomposition can be adopted to preserve both contextual constraints and temporal smoothness of multimode time series (Cai et al. [CTF<sup>+</sup>15]). In addition, Jing, Tong, and Zhu [JTZ21] propose a deep learning model that captures both the explicit relations by tensor graph convolution networks and the implicit relations of temporal dynamics by tensor recurrent neural networks.

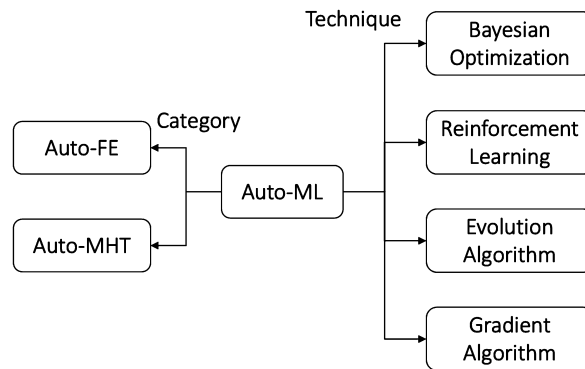
For network of networks, Ni, Tong, Fan, and Zhang [NTFZ14] propose to apply the cross-network consistency principle that the ranking scores of the common nodes shared across multiple domain-specific networks should be similar if these domains themselves are closely related. Likewise, Ni, Tong, Fan, and Zhang [NTFZ15] also propose a clustering method by assuming the cluster assignments of the same node in two highly similar domain-specific networks should be similar to each other. In addition, for the task of graph classification, by constructing each graph instance as a node of the contextual network, the classifiers at both the graph instance (i.e., domain-specific network) level and the contextual network level can be learned in an alternative manner (Li et al. [LRC<sup>+</sup>19]).

Lastly, for network of data mining models, Li and Tong [LT15] propose a joint predictive model that connects individual predictive models with the main network and encourages the parameters of the closely related predictive models to be consistent.

### **12.3.5 Auto-ML: methods and systems**

Data mining and machine learning models have been widely applied and implemented to tackle with a variety of applications in the fields of computer vision, natural language processing, and many more. However, there are numerous and diverse settings (e.g., tasks and data sets) in the real world, and thus building and training appropriate mining models for each task could be time-consuming. Additionally, domain experts (e.g., biology and business) might not be always familiar with the detailed implementation of mining models, not to mention building and tuning appropriate mining models for the specific domain applications.

**Example 12.8.** Suppose a stock trader would like to forecast the stock price of some companies of interest, but he is not familiar with data mining or machine learning models for modeling time series and has no idea about how to build and train an accurate prediction model. In this scenario, the trader could leverage the Auto-ML platform to find the optimal model and its hyperparameters.  $\square$

**FIGURE 12.10**

Taxonomy of Auto-ML. FE denotes feature engineering. MHT denotes model and hyperparameter tuning.

To tackle with these problems, Automated Machine Learning (Auto-ML) has been proposed, and it has garnered plenty of research attention in recent years. Auto-ML can be classified into two major categories, including (1) Automated Feature Engineering (Auto-FE), which automatically detects the most representative and informative features, and (2) Automated Model and Hyperparameter Tuning (Auto-MHT), which automatically builds machine learning models and tunes the hyperparameters. The most commonly used techniques for Auto-ML include Reinforcement Learning (RL), Evolutionary Algorithms (EA), Bayesian Optimization (BO), and Gradient Approaches (GA). The taxonomy of Auto-ML is presented in Fig. 12.10.

In the category of Auto-FE, an RL-based method constructs a transformation graph consisting of all of combination of the transformation operations and features. The ultimate goal is to learn the optimal path in the transformation graph, which will lead to the best performance. An EA-based approach constructs a tree to represent the transformation of the features. In the category of Auto-MHT, BO is one of the most widely used approaches, which leverages probabilistic models (e.g., Gaussian process) to find the optimal settings for hyperparameters. RL is another widely adopted approach for tuning hyperparameters and designing machine learning pipelines. Some methods view the hyperparameter tuning problem as a multiarm bandit problem, where the hyperparameters of a learning model are viewed as arms. Policy gradient has been adopted to train agents to construct the optimal neural architectures. The EA-based approaches mainly focus on finding the optimal machine learning model architectures or pipeline. A tree-structured machine learning pipeline has been formulated, where a leaf node represents the input data and an intermediate node represents a particular machine learning model. Auto-ML can also be formulated as a case of the tournament selection, which repeatedly compares two randomly selected architectures and then chooses the one with better performance into the next population. Methods based on GA usually encode neural architectures into a continuous space. Some methods relax the categorical choices over the operations to a softmax of all the possible operations during training, whereas others map the input network architecture into a hidden representation via an encoder and then generate a new network architecture via a decoder. Interested readers can refer to bibliographic notes for other related works.



There are several key challenges for Auto-ML. The first challenge is the lack of authoritative benchmarks for Auto-FE and Auto-MHT. For Auto-FE, there is no standard protocol for the training setting and the transform operations for features by the date of writing this textbook. For Auto-MHT, especially deep learning models, the existing benchmark named NAS-Bench-101 only considers cell-based search space, whereas there is still no benchmark for the whole-network space. The second challenge is the efficiency of Auto-ML. Many Auto-MHT methods for deep learning models are time and resource consuming. Besides, some models also require prolonged time at the evaluation stage. The third challenge is how to incorporate human knowledge or experiences into the Auto-ML to find suitable subspace of the entire search space. The fourth challenge is the interpretability of Auto-ML. It will be useful if the users of the Auto-ML framework could understand why the features and the specific architectures are chosen. The fifth challenge is the scope of applications. Most of the existing Auto-ML frameworks focus on the classification and regression tasks. Auto-ML frameworks for more complex tasks such as image captioning and recommender systems are still under explored.

### ***Bibliographic notes***

Auto-FE manipulates input data features to improve the model's performance. Khurana et al. [KST18] and Chen et al. [CLL<sup>+</sup>19] use RL to find the optimal sequence of data transformations. Tran et al. [TXZ16] and Viegas et al. [VRG<sup>+</sup>18] generate tree-structured data transformations by EA. There are also some other approaches for Auto-FE. Kanter et al. [KV15] and Katz et al. [KSS16] employ truncated SVD and random forest to select features.

Auto-MHT aims to obtain an optimal model and/or its hyperparameters for the given data sets and tasks. For traditional machine learning methods, Snoek et al. [SLA12] and Hutter et al. [HHLB11] propose BO-based approaches relying on the Gaussian process and random forest, respectively. Li et al. [LJD<sup>+</sup>17] use RL to tune hyperparameters. Olson et al. [OBUM16] and Chen et al. [CWM<sup>+</sup>18] propose a tree-based EA and a layer-based EA, respectively. For deep learning methods, Jin et al. [JSH19] introduces Auto-Keras for a neural architecture search based on BO. Zoph et al. [ZL16,ZVSL18] use policy gradient and proximal policy optimization of RL to search for the optimal neural networks. Real et al. [RMS<sup>+</sup>17,RAHL19] propose EA-based methods that search for the optimal neural architectures via variants of tournament selection process (Goldberg and Deb [GD91]). GA-based approaches encode the architectures into continuous vector representations. Liu et al. [LSY18] and Luo et al. [LTQ<sup>+</sup>18], respectively, use mixed operations and encoder-decoder to learn continuous representations for neural architectures.

---

## **12.4 Data mining, people, and society**

### **12.4.1 Privacy-preserving data mining**

Data mining harvests knowledge from data. At the same time, people may also raise concerns about privacy protection when their data is analyzed. As a concrete example of privacy intrusion by mining personal data, in 2012, a major retail company accurately identified that a teen girl was pregnant and sent her ads for baby products using a pattern that indicates a woman is likely pregnant if 25 specific products were purchased together. The dad only discovered the pregnancy after receiving the ads. How to protect privacy in data mining becomes a more and more serious and extensively worrying concerns.

Privacy leakage during data analytics may bring dramatic risks to data owners, data users, data mining service providers, and our society in general. For example, privacy leakage may lead to significant risks for business. With big data and powerful data mining techniques, individuals may be re-identified, and anonymity may be broken. For example, a user may provide reviews on products under the agreement of being kept anonymous. However, though accurate mining of rich customer data, the user may be re-identified, and thus the user's privacy is intruded. Data breaches or misusing data in analytics may cause lawsuits and consequent financial liabilities. At the ethical level, just because that some knowledge can be discovered and some events can be predicted, should that knowledge be used and the prediction be acted on?

As re-identifying an object, such as a user, from data is one important type of intrusions, a large group of methods have been developed to anonymize data and protect privacy. For example,  $k$ -anonymity (Samarati and Sweeney [SS98]) is one of the early and fundamental techniques. Consider Table 12.1, which contains a set of personal records. The column ID is for reference only and is not released. We want to publish the data to researchers so that they can investigate the distribution of disabled people. Thus the column Disability is to be published as it is. Attributes Address and Age in the table are identifying attributes. Suppose we want to protect privacy of any people with disability. Publishing the original table immediately leaks the privacy of records R3 and R5 to R8, since using the address and age information the person with disability can be identified. The  $k$ -anonymity idea generalizes the data in the identifying attributes such that each record published is identical to at least  $k - 1$  other records on the identifying attributes. For example, Table 12.2 shows such a four-anonymization, where each tuple is identical to another three tuples on the identifying attributes, and thus one cannot be re-identified from the published data with probability higher than  $\frac{1}{k}$ , that is,  $\frac{1}{4}$  in this example. When we produce a  $k$ -anonymity, we try to make minimal changes so that the data utility can be retained as much as possible. For example, in the second group containing records R5 to R8 in Table 12.2, the street name, Franklin Avenue, is retained, and only the street numbers are generalized.

$k$ -anonymity is simple and can protect privacy to some extent. However, it is vulnerable to many attacks, particularly when attackers are equipped with some background knowledge. For example, the group of records R5 to R8 have the same value on the sensitive attribute Disability. Thus even though the identifying attributes are  $k$ -anonymized, the sensitive values of these group of four records are leaked. This is known as the homogeneity attack (Machanavajjhala, Gehrke, Kifer, and Venkatasubramaniam [MGKV06]). As another example, if an attacker may learn the background knowledge that people in

**Table 12.1** A data set containing sensitive personal data attribute “disability.”

| ID | Address                                     | Age | Disability |
|----|---------------------------------------------|-----|------------|
| R1 | 12 Front Street, Central Park, MyState      | 32  | No         |
| R2 | 38 Main Street, Central Park, MyState       | 43  | No         |
| R3 | 16 Front Street, Central Park, MyState      | 35  | Yes        |
| R4 | 833 Clinton Drive, Central Park, MyState    | 40  | No         |
| R5 | 1654 Franklin Avenue, Central Park, MyState | 74  | Yes        |
| R6 | 235 Franklin Avenue, Central Park, MyState  | 78  | Yes        |
| R7 | 2323 Franklin Avenue, Central Park, MyState | 72  | Yes        |
| R8 | 392 Franklin Avenue, Central Park, MyState  | 75  | Yes        |

**Table 12.2 A data set containing sensitive personal data attribute “disability.”**

| ID | Address                                | Age     | Disability |
|----|----------------------------------------|---------|------------|
| R1 | Central Park, MyState                  | [30-45] | No         |
| R2 | Central Park, MyState                  | [30-45] | No         |
| R3 | Central Park, MyState                  | [30-45] | Yes        |
| R4 | Central Park, MyState                  | [30-45] | No         |
| R5 | Franklin Avenue, Central Park, MyState | [70-75] | Yes        |
| R6 | Franklin Avenue, Central Park, MyState | [70-75] | Yes        |
| R7 | Franklin Avenue, Central Park, MyState | [70-75] | Yes        |
| R8 | Franklin Avenue, Central Park, MyState | [70-75] | Yes        |

age group [70–75] have a probability of disable at least twice higher than the average, then the attacker can issue a background knowledge attack (Machanavajjhala, Gehrke, Kifer, and Venkatasubramanian [MGKV06]) to increase the chance of obtaining the sensitive information on victims in the age group.

In order to describe the patterns of groups in a database and, at the same time, protect the information about individuals, differential privacy (Dwork, McSherry, Nissim, and Smith [DMNS06]) is developed. The general idea is that we allow users to derive aggregates on groups of records in a data set, as long as for an aggregate derived from a group, whether an individual belongs to the group cannot be determined with a high confidence. Since whether a single individual belongs to a group or not cannot be determined accurately, the privacy of individuals is protected.

Technically, a randomized algorithm  $\mathcal{A}$  is said to be  $\epsilon$ -differentially private if for any two neighboring data sets  $D_1$  and  $D_2$  that differ on only a single element and for all subsets  $S$  of possible outputs of  $\mathcal{A}$ ,  $\Pr[\mathcal{A}(D_1) \in S] \leq e^\epsilon \Pr[\mathcal{A}(D_2) \in S]$ . In other words, for any two neighboring data sets  $D_1$  and  $D_2$ ,  $e^{-\epsilon} \leq \frac{\Pr[\mathcal{A}(D_1) \in S]}{\Pr[\mathcal{A}(D_2) \in S]} \leq e^\epsilon$ . Here,  $\epsilon$  is a parameter taking a small real number. For example, when  $\epsilon = 0.01$ ,  $0.99 \leq \frac{\Pr[\mathcal{A}(D_1) \in S]}{\Pr[\mathcal{A}(D_2) \in S]} \leq 1.01$ . Since the probabilities of the results are produced by  $D_1$  and  $D_2$  are so close, an attacker cannot accurately determine whether the only element that is the difference between  $D_1$  and  $D_2$  participates in the computation or not, and thus the privacy of that element is protected. When  $\epsilon = 0$ , the level of privacy protection is maximized, since the algorithm outputs  $\mathcal{A}(D_1)$  and  $\mathcal{A}(D_2)$  with indistinguishable distributions. In this situation, the output results do not reflect any useful information about the data. Thus parameter  $\epsilon$  balance the tradeoff between privacy and data utility.

How can we add noise to achieve differential privacy? For a function  $f$ , the global sensitivity of  $f$  for all pairs of neighboring data sets  $D_1$  and  $D_2$  is the maximum difference of the function values, that is,  $GS_f = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_1$ , where  $\|\cdot\|_1$  is the  $L_1$  norm. We can make up a randomized algorithm  $\mathcal{A}(D) = f(D) + Z$ , where  $Z \sim \text{Lap}(\frac{GS_f}{\epsilon})$  is random noise following the Laplace distribution with scale  $\frac{GS_f}{\epsilon}$ . The Laplace distribution centered at  $\mu$  with scale  $b$  is the distribution with probability density function  $h(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$ .

The global sensitivity of some functions is easy to calculate, such as sum, count, and max. However, there are many functions whose global sensitivity is hard to compute or even infinite, such as the maximum diameter of  $k$ -means clusters and subgraph counting. There are many relaxations and extensions

of differential privacy to accommodate the need of balancing tradeoff between privacy protection and data utility.

Privacy leakage happens not only between data owners and data mining result consumers, but also may occur between different data owners who collaboratively conduct data mining using jointly their data. For example, multiple companies want to collaboratively select  $k$  locations to build product shipment delivery stations so that their customers can be served with fast deliveries. This can be solved by conducting  $k$ -means clustering on the customer address data. However, those companies do not want to share the customer address information, since customer address information may be considered private information for customers and commercial secrets for each company. How can those companies jointly conduct data mining using their data but do not disclose the sensitive address information?

One general framework to tackle the challenge is to conduct federated learning and federated analytics. Federated learning and federated analytics apply data mining methods to analyze data stored locally on data owners' devices and sites. It works by running local computation over each data owner's data, and makes only the aggregated results available to the central server or the other users. Detailed data from individual data owners is never transmitted or reflected.

For example, to conduct  $k$ -means over data from  $n$  owners in a federated manner, instead of consolidating the data from those owners into a single data set, the central server randomly selects the initial  $k$  centers and sends to the owners. Each owner locally computes the data objects assigned to each center, and the updated local center for each cluster. Only the  $k$  updated local center and, for each updated center, the corresponding number of objects assigned and the within-cluster variation (Eq. (9.1)) are reported to the central server. Any specific data records are kept within the data owner and never shared with either the central server or other data owners. Then the central server collects the local information from the owners and updates the global  $k$  centers. Multiple iterations can be conducted until the global  $k$  centers become stable.

Federated learning and federated analytics provide a promising way to achieve data mining and data analytics collaboration. Various models can be built in federated way, such as classifiers, clustering, and data distributions. Federation may happen in horizontal or vertical ways. Horizontally, the data owned by each data owner follows the same schema. Vertically, different data owners have the data on only some attributes of a set of objects. In the above example, centralized federated learning is assumed, where there is a central server to orchestrate different steps of the mining process and coordinate the participating data owners. Alternatively, in decentralized federated learning, the participating data owners coordinate themselves. Recently, heterogeneous federated learning is developed to accommodate heterogeneous users with very different computation and communication capabilities.

Privacy preservation in data mining goes far beyond just technical. It calls for societal and legal efforts. For example, significant efforts on legislations have been committed to protect privacy on various aspects. For example, the European Union (EU) and the United States established Directive 95/46/EC of the European Parliament and the Council of 1995 and Health Insurance Portability and Accountability Act of the US Department of Health and Human Services (HHS) of 1996 (HIPAA) as the anonymization and de-identification-based legislation. General Data Protection Regulation (EU GDPR) is a regulation in EU law on data and data protection, that gives individuals control on personal data, and simplifies the regulatory environment for international business by unifying the regulation within the EU.

## 12.4.2 Human-algorithm interaction

So far in this textbook, we have been mainly focusing on designing effective and scalable algorithms for a data mining model. “*But, how do such algorithms interact with humans (i.e., system administrators, end-users)?*” In general terms, the *human-algorithm interaction* aims to optimize the collaboration between human intelligence and data mining algorithms and consequently enhances the system performance and user experience. Let us elaborate this from the following perspectives, including (1) utilizing the human intelligence to solve tasks that are difficult for algorithms (i.e., *crowdsourcing*); (2) leveraging human intervention to further improve the data mining algorithms (*human-in-the-loop*); (3) improving humans’ query strategy by algorithms (i.e., *machine-in-the-loop*); and (4) fostering the effective teamwork between humans and algorithms (i.e., *human-machine-teaming*).

**Crowdsourcing.** The objective of crowdsourcing is to harness the human intelligence to resolve the problems that are difficult to be solely solved by machine learning and data mining algorithms, such as recognizing a specific type of plant in a picture, rating the products on a website, or writing a summary for a short paragraph. The major applications for crowdsourcing are summarized as follows. First, an important application of crowdsourcing is generating data samples with high-quality labels, including binary or categorical labels, sentence translation, and image annotations. To improve the quality of the crowdsourced labels, some studies present each data sample to multiple workers and obtain the final label by summarizing the workers’ responses. Another research direction of label generation is to evaluate the worker’s quality by considering other workers’ response on the same instances (i.e., peer prediction). Second, crowdsourcing has been widely applied to evaluate the learning models and debug faulty components in a learning system. For model evaluation, crowdsourcing is leveraged to determine whether the output from the unsupervised topic model is meaningful (i.e., to what extent the extracted key words from an article are helpful for users to understand the article). Additionally, crowdsourcing is capable of evaluating the interpretability of model predictions in some critical domains such as medical diagnosis where researchers are interested in developing human-interpretable algorithms. In terms of debugging the AI system, existing works utilize human power to identify the weakest component in the AI system that consists of multiple discrete parts for a complex task. Third, human intelligence has been explored in the AI systems that are dependent on human judgment or domain knowledge. Such hybrid AI systems often achieve superior performance compared to individual humans or learning algorithms in the tasks of clustering, planning and scheduling, events prediction, and forecasting. Beyond machine learning and data mining research, crowdsourcing platforms have been extensively exploited in psychology and social science researches to conduct experiments on social behaviors, which consequently facilitates the development of interdisciplinary research and helps gain a profound comprehension of the interaction between human and learning algorithms (e.g., whether people trust the prediction results and how they react to the recommended items in online shopping).

**Human-in-the-loop.** In general, human-in-the-loop (HITL) aims to effectively incorporate human intelligence in the development of learning algorithms to create advanced AI systems. Human interventions in an AI system can be in various formats. In the training phase, supervised learning tasks (e.g., image classification, speech recognition) often rely on human effort to obtain high-quality data samples with annotations (i.e., labels) and to design algorithms that are applicable to the specific scenario. Recent trends in deep neural networks have led the automated AI systems to be increasingly powerful and complex. Nonetheless, the intrinsic black-box property of deep learning approaches makes it highly challenging to interpret the prediction results for end users. Therefore human intelligence with sufficient

background knowledge is essential to enhance the model's capability to explain predictions, particularly in societally critical domains of applications such as healthcare and justice systems. Additionally, human intelligence can also function as the evaluator for algorithms that aim to generate explanations for the results by assessing the quality of explanations, which is also one important role of human in the system (i.e., improving the quality of explanations). In medical science, experts' experience and prior knowledge are critical for annotating medical data. For example, in medical image acquisition, certain expertise is required to label the informative area in an image for accurate computer-aided diagnosis. Future cyber-physical systems (CPSs) also rely on a closer connection with human intent to develop powerful and complex systems with a large number of intelligent devices (e.g., smart phones, sensors, computational resources.).

**Machine-in-the-loop.** Compared to human-in-the-loop systems, machine-in-the-loop concentrates more on leveraging learning algorithms to support human workers to accomplish certain tasks. Generally speaking, the goal of machine-in-the-loop is to explore the best approaches to integrate learning algorithms into human decision making. For instance, existing work investigates the possibility of incorporating algorithms to generate ideas for creative writing. Another interesting application is that artificial intelligence techniques are leveraged to help users achieve fast and creative drawing. Given the fact that labeled data are often expensive and requires laborious human effort and sometimes the number of labeled data is limited because of privacy constraint, numerous works in active learning (AL) focus on achieving comparable performance while maximally reducing the number of labeled data for training.

**Human-machine teaming.** The recent advancement of team science in the context of networks (see Section 12.2.4 for an introduction), together with the breakneck development of data mining and machine learning algorithms, cultivates a unique form of teams that involves both human agents and learning algorithms (i.e., human-machine-team, or HMT for short). Humans and algorithms have mutually complementary skills. Therefore the well-coordinated teamwork between human agents and machine agents will be likely to achieve superior performance than human-only or machine-only teams. The key challenges of human-machine teaming include the following: (1) effectively modeling the coordination, communication, and collaborations in a human-machine team given its nature of heterogeneity, hierarchy, and dynamics; (2) accurately predicting the team performance in both static and dynamic scenarios; and (3) refining the human-machine team toward high performance in terms of improving learning algorithms and updating human-machine interactions. Fig. 12.11 provides an illustration of human-machine teaming, where Fig. 12.11(a) is an overview of HMT that represents the key components and the interactions inside an HMT network. Fig. 12.11(b) presents more details for a cyber-defense scenario involving both human and machine members, where the security agents (i.e., machines) aim to identify suspicious cyber activities and the cyber analyst (i.e., humans) can provide feedback (e.g., annotation) for machine agents and resolve difficult cases. Effective cyber defense is achieved by inter/intrateam information sharing among the human and machine members in this system.

### ***Bibliographic notes***

Crowdsourcing has been widely explored in a variety of tasks. For example, Raykar et al. [RYZ<sup>+</sup>10] propose to leverage crowdsourcing to generate image annotation in order to train a computer vision model. Miao et al. study the problem of adversarial activities against crowdsourcing system [MLS<sup>+</sup>18]. Crowdsourcing techniques are also utilized to improve the reliability of medical diagnosis, which is

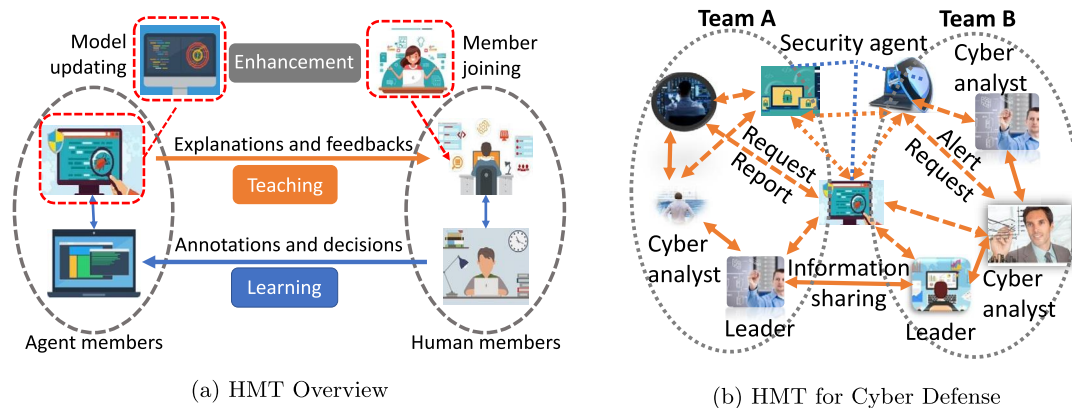


FIGURE 12.11

Human-machine teaming.

studied by Li et al. [LDL<sup>+</sup>17]. To resolve the noisy issue of data labels, Khetan and Oh [KO16] introduce an adaptive scheme to balance the tradeoff between model performance and budget. To infer the true labels from the large pool of noisy labels, Zhou and He [ZH16] first leverage tensor augmentation and completion to improve the quality of obtained labels. For heterogeneous data, Zhou and He [ZH17] propose a learning framework that can effectively utilize the structural information from data heterogeneity to improve the model generality and quality. The quality of workers is a key factor in determining the label accuracy. To address this challenge, Zhou, Ying, and He [ZYH19] propose an optimization framework to model the task and the worker dual heterogeneity by studying the cross-network behaviors.

By incorporating humans in the loop in building hybrid intelligent systems, clustering algorithms can be facilitated based on shared knowledge, which is investigated by Heikinheimo and Ukkonen [HU13]. Human intelligence is also leveraged to effectively select the high-quality training set to construct a machine learning model (i.e., machine teaching). For instance, Zhou, Nelakurthi, and He [ZNH18] propose a novel teaching framework to supervise crowd to label. Under the scenario that the quality of some labels are imperfect, Zhou et al. propose an adaptive approach to improve the labeling through sequential interactions between machine teacher and workers. Human can also support the explainability of machine learning algorithms; Lai, Carton, and Tan [LCT20] aim to assist human decision making through high-quality explanations. Mothilal, Sharma, and Tan [MST20] propose a framework for generating and evaluating counterfactual explanations to aid users understand the learning algorithms.

### 12.4.3 Mining beyond maximizing accuracy: fairness, interpretability, and robustness

The ever-growing amount of data and the rapid development of data mining techniques have enabled ubiquitous automated decision making in various application domains. For example, social media applications such as Facebook and Instagram can automatically recommend multimedia contents and

friends based on users' tastes and social networks. Companies such as LinkedIn and XING adopt data mining-based systems for ranking potential candidates during recruiting and hiring.<sup>4</sup> Courts in United States adopt the COMPAS algorithm for recidivism prediction, which is reported to achieve higher accuracy compared with trained humans.<sup>5</sup> For such applications and many more, in addition to accuracy (e.g., precision and recall of recommendation as well as ranking, and AUC of recidivism prediction), there are a number of other important metrics we need to consider. In this section, we look at three of them: *fairness*, *interpretability*, and *robustness*.

**Algorithmic fairness.** As promising as the data mining techniques might be, there might exist potential, often unintentional, bias when used inappropriately.

In order to solve the fairness issues in the data mining techniques, the first question is how to define fairness in a formal way. From a data mining perspective, the fairness definition can be categorized as group fairness, individual fairness, and counterfactual fairness. First, group fairness includes three subcategories, namely demographic parity (i.e., statistical parity), equalized odds, and predictive rate parity. Specifically, the demographic parity denotes that the probability of prediction results should be independent of the sensitive attributes (e.g., gender, race, etc.). The equalized odds denote that for different sensitive attributes, the conditional probability of the same prediction results given the same labels should be equal. The predictive rate parity means that for different sensitive attributes, the conditional probability of the labels given the prediction results should be equal. Second, the idea of individual fairness is to preserve individual similarities, such that similar individuals should be treated similarly. Third, the counterfactual fairness aims to correct predictions of a label variable that are unfairly altered by an individual's sensitive attribute. It offers a way to check the potential impact of replacing the sensitive attribute.

Generally speaking, there are three types of strategies for fair learning algorithms, including preprocessing, optimization at training, and postprocessing. The key idea of preprocessing methods is to learn a representation of individual features so that the impact of sensitive attributes are removed. The learned representation is in turn used in the downstream data mining tasks, so that the individual fairness or the demographic parity could be enforced. The key idea of optimization at training is to use additional regularizers or constraints in the original objective function for the corresponding data mining tasks. The optimization at training methods could often obtain a good tradeoff between mitigating the bias and retaining the original mining accuracy. Given a classifier which uses real-valued predictive score (e.g., FICO scores for predicting loan default), the key idea of postprocessing is to find a proper threshold using the original score function for each group in order to achieve certain types of fairness constraints (e.g., equalized odds). Neither preprocessing nor postprocessing requires changes to the original data mining model (e.g., a classifier). A summary of these three categories of learning algorithms can be found in Table 12.3.

Algorithmic fairness is an active research area with many possible future directions. To name a few, besides inspecting data mining models, one could scrutinize data and how the bias was introduced in the first place during the data generation process. Besides static one-shot problems that are the main focus of the current research, it would be interesting to examine the long-term effects of feedback loops and human interventions on the potential bias of the data mining systems. For the fair learning model

---

<sup>4</sup> <https://emerj.com/ai-sector-overviews/machine-learning-for-recruiting-and-hiring/>.

<sup>5</sup> <https://news.berkeley.edu/2020/02/14/algorithms-are-better-than-people-in-predicting-recidivism-study-says/>.



| Fair learning methods | Preprocessing                                                                                                                          | Optimization at training                                                                                                                                      | Postprocessing                                                                                                  |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Key ideas             | Learning feature representation to mitigate the impact of sensitive attributes                                                         | Using additional constraints in the objective of original mining tasks                                                                                        | Modifying thresholds of learned models in order to satisfy fairness constraints                                 |
| Pros                  | (1) Can be used for any downstream task<br>(2) No need to modify classifier<br>(3) No need to access sensitive attributes at test time | (1) Relatively good performance<br>(2) Flexible on tradeoff between accuracy and fairness measures<br>(3) No need to access sensitive attributes at test time | (1) Can be applied after any classifiers<br>(2) Relatively good performance<br>(3) No need to modify classifier |
| Cons                  | (1) Only capable of optimizing statistical parity and individual fairness<br>(2) Relatively lower performance                          | (1) Task specific solver<br>(2) Need to modify classifier (may not be feasible)                                                                               | (1) Need to access the sensitive attribute at test time<br>(2) No explicit accuracy and fairness tradeoff       |

itself, both the theoretical and experimental tradeoff between utilities and individual or group fairness is worth further studying.

**Interpretability.** Most of the existing data mining and machine learning methods are “black boxes” and thus are hard for the end users (who are often not data mining experts) to understand the mining process or the results. The lack of interpretability and transparency in a data mining model and system will in turn cause trust, safety, and contestability issues.

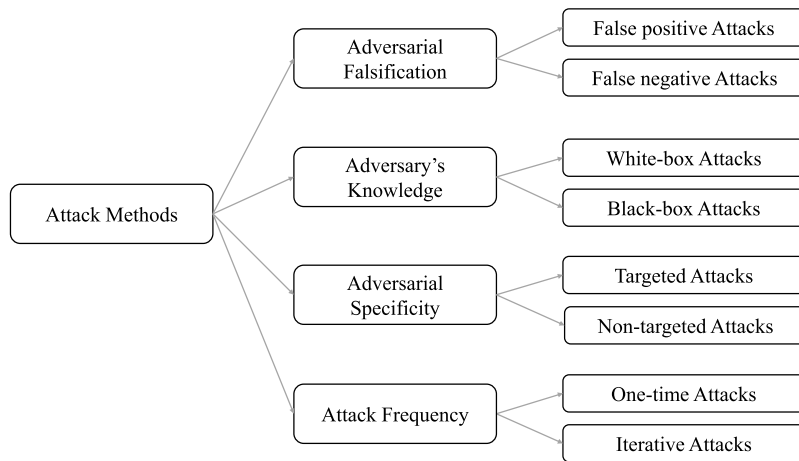
**Example 12.9.** The AI-aided diagnosis of Alzheimer’s disease<sup>6</sup> via patients’ word usage or brain MRI image has received extensive interest in data mining research. However, if the model can not provide human (doctor) interpretable predictions, the results will face trust and reliability issues. In another application, the lack of contestability has already led to significant criticism of proprietary recidivism predictors such as COMPAS, because it fails to let people appeal these decisions.<sup>7</sup> □

In Chapter 7, we have learned some basic techniques to interpret classification. Beyond that, many interpretable data mining methods have been developed. To name a few, some works use summary statistics for each feature as interpretation; some methods visualize the feature summary statistics as intuitive interpretation; some methods aim to explore self-interpretable components of the models for interpretation, such as the weights in linear models or the learned tree structure of decision trees; and other methods focus on identifying key data points (both existing and newly created samples) to make a model interpretable.

**Robustness.** Another important aspect is the robustness of data mining models. Generally speaking, a robust data mining model should satisfy the condition that the test results are consistent with training results, or the test results stay stable with unintentionally inserted noises and intentional adversarial attacks.

<sup>6</sup> <https://www.scientificamerican.com/article/ai-assesses-alzheimers-risk-by-analyzing-word-usage/>.

<sup>7</sup> <https://www.theatlantic.com/technology/archive/2018/01/equivalent-compas-algorithm/550646/>.

**FIGURE 12.12**

The taxonomy of adversarial attacks.

**Example 12.10.** One of the most well-known examples of robustness is the image classification by Convolutional Neural Networks (CNNs). Researchers have found that slight modifications that are imperceptible to the human eyes (e.g., a few additional darker pixels inserted in an image) may cause a CNN model to produce drastically different classification results.  $\square$

The taxonomy of representative adversarial attacks is shown in Fig. 12.12, based on four criteria (i.e., adversarial falsification, adversary's knowledge, adversarial specificity, and attack frequency). Common defend strategies can be divided as *reactive* methods and *proactive* methods. Reactive methods detect adversarial examples after data mining models have been built, whereas proactive methods try to make mining models more robust before attackers generate adversarial examples. Representative reactive approaches include adversarial detection, input reconstruction, and network verification; representative proactive approaches include network distillation, adversarial (re)training, and classifier robustifying. Besides adversarial attacks and defense, other robustness studies focus on the stability of the model performance. For example, an interesting method for studying the model robustness is through a game. Considering a classifier, the goal of designing a robust classifier through the game is to minimize the loss while choosing a distribution of test data to maximize the expected loss given the knowledge of this classifier.

### **Bibliographic notes**

Calmon et al. [CWV<sup>+</sup>17] develop a convex optimization method for learning a data transformation in order to mitigate discrimination, limit distortion in individual data samples, and preserve utility. Gordaliza et al. [GDBFL19] try to detect when a binary classification rule lacks fairness and to fight against the potential discrimination by two fairness definitions (i.e., disparate impact and balanced error rate). Zemel et al. [ZWS<sup>+</sup>13] develop an optimization method for maintaining group fairness and finding a good representation for obfuscating the data. Lum et al. build a statistical framework

for removing the sensitive information in the features. Calders, Kamiran, and Pechenizkiy [CKP09] develop an accurate model for which the predictions are independent from a given binary (sensitive) attribute for ensuring individual fairness. Kang et al. [KHMT20] define individual fairness for graph mining and develop debiased approaches for tackling fairness problem on graph mining tasks.

Molnar comprehensively summarizes the fundamental interpretable machine learning approaches [Mol20]. Samek et al. [SWM17] propose two methods to explain predictions of deep learning models. One method computes the sensitivity of the prediction with respect to changes in the input, and the other decomposes the decision in terms of the input variables. Lundberg et al. [LNV<sup>+</sup>18] explore an explainable machine learning approach for the prevention of hypoxaemia during surgery, which draws extensive attention in both machine learning and bioinformatics domains. Holzinger [Hol18] comprehensively elaborates various explainable machine learning and points out several future directions.

Ren et al. [RZQL20] survey the recent advances in adversarial attacks and defense approaches comprehensively. Papernot et al. use network distillation to defend deep neural networks against adversarial examples [PMW<sup>+</sup>16]. Goodfellow et al. [GSS14] and Huang et al. [HXSS15] propose to include adversarial examples in the training stage and generate adversarial examples in every step of training to inject them into the training set. Zugner et al. [ZAG18] introduce the adversarial attack problem on graphs, which focuses on the training stage of Graph Convolutional Networks. Bagnell et al. develop a wrapper framework around a broad class of supervised learning algorithms to guarantee the robust behavior under changes in the input distribution [Bag05].

#### 12.4.4 Data mining for social good

In recent years, data mining techniques have been applied to various settings with significant societal impacts, under an umbrella term “data mining for social good.” Let us look at an example of the lead water pipe detection project.

**Example 12.11.** Due to the corrosion of lead water pipes, Flint’s drinking water in Michigan was contaminated seriously, with significant consequences to public health. Since most of the records of water pipe services were incomplete or even lost, city officials were uncertain about the locations of lead water pipes. The ActiveRemediation project, led by a group of researchers at the University of Michigan, adopts various machine learning and data mining techniques to guide the pipe replacement procedure. For instance, it combines XGBoost (an ensemble technique introduced in Chapter 6) with a hierarchical Bayesian spatial model to estimate the probability that a pipe contains hazardous materials. Furthermore, active learning technique (introduced in Chapter 7) is used to identify homes for inspection and to guide the data collection process. Since its first deployment in 2016, the ActiveRemediation project has achieved an impressive accuracy of 70%, and it has successfully replaced 6228 pipes made of hazardous metals through 2017. □

Numerous efforts have been made by leveraging data mining techniques to make positive societal impacts on a multitude of applications, ranging from education, public health, combating information manipulation, social care and urban planning, and public safety, to transportation (see Table 12.4 for a summary). For example, various data mining models (such as Random Forests and Adaboost) have been used to identify students who need intervention to graduate high school on time; a deep learning approach called EpiDeep has been developed to predict the future trend of the epidemic, with improved prediction accuracy and interpretability; the SVM classifier with RBF kernel function has been

**Table 12.4 Applications and related data mining techniques. Adapted from Shi, Wang, and Fang [SWF20].**

| Applications                       | Key Data Mining Techniques                                                                 |
|------------------------------------|--------------------------------------------------------------------------------------------|
| Education                          | Random Forests<br>Logistic Regressions                                                     |
| Public Health                      | Long Short-Term Memory (LSTM)<br>Autoencoder<br>Deep Clustering                            |
| Combating Information Manipulation | SVM with RBF Kernel Function<br>Markov Random Field (MRF)<br>Graph Attention Network (GAT) |
| Social Care and Urban Planning     | Gradient Boosting Decision Trees<br>AdaBoost<br>Transfer Learning                          |
| Public Safety                      | Generalized Linear Model (GLM)<br>Naive Bayes (NB)                                         |
| Transportation                     | Demand and Supplier Modeling<br>Multiagent Reinforcement Learning                          |

shown to be effective to detect rumors on social media; a multimodal transfer learning method called FLORAL has been developed that is capable of transferring semantically related dictionaries between different cities and enriching feature representations of a target city with knowledge from the source city; generalized linear models including logistic regression have been used to forecast civil protests from multiple data sources such as tweets, news, and blogs; and multiagent reinforcement learning framework has been developed to model the complicated and high-dimensional dynamics between demands and supplies on ride-sharing platforms.

The key challenges of data mining for social good come from the following aspects. First (data scarcity), it is usually difficult to collect large-scale data from real scenarios, which could impair the performance of supervised data mining methods. To tackle this challenge, researchers have used unsupervised learning or transfer learning in some current efforts. Second (evaluation), since the primary goal of data mining for social good is to address real-world problems with significant societal consequence, standard evaluation metrics alone might be insufficient. It is important to develop application domain-specific evaluation metrics. Third (human-in-the-loop), domain experts such as social workers and doctors have many critical knowledge and experiences that should be leveraged to further improve the data mining methods. Hence, how to seamlessly incorporate such human knowledge into the data mining process and build effective interactions between humans and algorithms is another important research challenge. Fourth (sustainable deployment), the ultimate goal of data mining for social good is to develop models that could be deployed in real-world sustainably. However, only a few of the current projects have completely achieved this goal. The reasons include the willingness of collaborating partners and the status of funding.

### ***Bibliographic notes***

Abernethy et al. [ACF<sup>+</sup>18] develop ActiveRemediation to detect pipes made of hazardous metal. They combine XGBoost with a hierarchical Bayesian spatial model to estimate the probability that a pipe contains hazardous materials and use Importance Weighted Active Learning to select homes for inspection. Lakkaraju et al. [LAS<sup>+</sup>15] develop a method to identify students who need intervention to graduate high school on time. They present a comprehensive comparison of various data mining models such as Random Forests and Adaboost on data collected from real scenarios. Baytas et al. [BXZ<sup>+</sup>17] develop Time-Aware LSTM (T-LSTM), which is able to handle irregular time intervals in patient records. Combining T-LSTM with the autoencoder, they introduce an unsupervised approach that can group patients only by their historical records. Adhikari, Xu, Ramakrishnan, and Prakash [AXRP19] design a novel deep learning approach called EpiDeep that learns embeddings from the historical data to predict the future trend of the epidemic. With the learned embeddings, EpiDeep is capable of finding the closest historical seasons to the current season that improves the performance and interpretability. Fan et al. [YLYY12] employ the SVM classifier with RBF kernel function to detect rumors on a micro-blogging platform. Rayana and Akoglu [RA15] develop a novel approach, SpEagle, to detect spam reviews based on metadata (text, timestamp, rating) and relational data (review network). SpEagle supports both the unsupervised and semisupervised setting, and the performance could be boosted significantly with only limited labeled data. Wei, Zheng, and Yang [WZY16] develop a multimodal transfer learning method called FLORAL to tackle the data insufficiency challenge in urban computing task. FLORAL was evaluated on the air quality prediction problem and exhibits superior performance on data from real scenarios. Avvenuti et al. [ACM<sup>+</sup>14] propose an emergency management system, EARS, to discover meaningful tweets about outbreaking crisis events such as earthquakes. With careful design of feature extraction and detection rules, EARS is capable of detecting earthquake events with a relatively low false positive and alert interested parties in time. Ramakrishnan et al. [RBM<sup>+</sup>14] develop the EMBERS system, which aims at forecasting civil protests from multiple data sources such as tweets, news, blogs, and other sources.

A variety of data mining models such as logistic regression and generalized linear models has been employed to make predictions for data from different sources, and these predictions are fused to generate final results. Lin, Zhao, Xu, and Zhou [LZXZ18] propose a multiagent reinforcement learning framework to model the complicated and high-dimensional dynamics between demands and supplies on ride-sharing platforms. With a proper design of action, reward, and state, the framework obtains efficient coordination between different agents in the dynamically changing environment.

# Mathematical background



## Notation naming convention

Unless otherwise stated, we use bold upper-case letters for matrices (e.g.,  $\mathbf{A}$ ), bold lower-case letters for vectors (e.g.,  $\mathbf{u}$ ), and lower-case letters for scalars (e.g.,  $c$ ). Regarding matrix indexing conventions, we use rules similar to Numpy. We use  $\mathbf{A}[i, j]$  to represent the entry of matrix  $\mathbf{A}$  at the  $i$ th row and the  $j$ th column,  $\mathbf{A}[i, :]$  to represent the  $i$ th row of matrix  $\mathbf{A}$ , and  $\mathbf{A}[:, j]$  to represent the  $j$ th column of matrix  $\mathbf{A}$ . We use superscript  $T$  to represent the transpose of a matrix (i.e.,  $\mathbf{A}^T$  is the transpose of matrix  $\mathbf{A}$ ) and the superscript plus sign to represent the pseudo-inverse of matrix (i.e.,  $\mathbf{A}^+$  is the pseudo-inverse of matrix  $\mathbf{A}$ ).

## A.1 Probability and statistics

### A.1.1 PDF of typical distributions

For a continuous random variable, its probability density function (PDF) is a function whose value at any given sample is the relative likelihood that the random variable would equal that sample. Any PDF  $f(x)$  must satisfy two conditions: (1)  $\forall x, f(x) \geq 0$  and (2)  $\int_{-\infty}^{+\infty} f(x)dx = 1$ .

Some PDFs of typical distributions are listed as follows:

- **Uniform Distribution:**  $X \sim U(a, b)$ , where  $a < b$ . The mean value of this distribution is  $(a + b)/2$  and the variance is  $(b - a)^2/12$ . The PDF of a uniform distribution is as follows:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a < x < b \\ 0, & \text{others.} \end{cases}$$

- **Exponential Distribution:**  $X \sim E(\lambda)$ , where  $\lambda > 0$ . The mean value of this distribution is  $1/\lambda$ . The variance of this distribution is  $1/\lambda^2$ . The PDF of an exponential distribution is as follows:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & \text{others.} \end{cases}$$

- **Normal Distribution:**  $X \sim N(\mu, \sigma)$ , where  $-\infty < \mu < \infty$  and  $\sigma > 0$ . The mean value of this distribution is  $\mu$ . The variance of this distribution is  $\sigma^2$ . The PDF of a normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

- **Gamma Distribution:**  $X \sim Ga(\alpha, \beta)$ , where  $\alpha > 0$  and  $\beta > 0$  are parameters. The mean value of this distribution is  $\alpha/\beta$ . The variance of this distribution is  $\alpha/\beta^2$ . The PDF of a gamma distribution is  $f(x, \beta, \alpha) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$ ,  $x > 0$ , where  $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$  is the Gamma function.
- **Beta Distribution:**  $X \sim B(a, b)$ , where  $0 < x < 1$ ,  $a > 0$  and  $b > 0$ . The mean value of this distribution is  $a/(a+b)$ . The variance of this distribution is  $ab/((a+b)^2(a+b+1))$ . The PDF of a beta distribution is  $f(x, a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}$ ,  $a > 0$ ,  $b > 0$ , where  $\Gamma(\cdot)$  is the Gamma function.

### A.1.2 MLE and MAP

Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) are two statistical methods for estimating parameters. We have the observed data points  $\mathbf{X} = \{\mathbf{x}_i\}$ , which satisfies the *i.i.d.* (independent and identically distributed) condition. The target for both MLE and MAP is to find the best parameter  $\theta$ .

In MLE, we assume that there exists a true fixed parameter  $\theta$ . The process to obtain  $\theta$  is as follows:

$$\begin{aligned}\theta_{MLE} &= \arg \max_{\theta} P(\mathbf{X}|\theta) \\ &= \arg \max_{\theta} \prod_i P(\mathbf{x}_i|\theta).\end{aligned}\tag{A.1}$$

Usually, we use the logarithm form for the above equation as follows:

$$\begin{aligned}\theta_{MLE} &= \arg \max_{\theta} \log P(\mathbf{X}|\theta) \\ &= \arg \max_{\theta} \log \prod_i P(\mathbf{x}_i|\theta) \\ &= \arg \max_{\theta} \sum_i \log P(\mathbf{x}_i|\theta).\end{aligned}\tag{A.2}$$

For MAP, it regards  $\theta$  as a random variable. By Bayes' rule, we have

$$\begin{aligned}P(\theta|\mathbf{X}) &= \frac{P(\mathbf{X}|\theta)P(\theta)}{P(\mathbf{X})} \\ &\propto P(\mathbf{X}|\theta)P(\theta).\end{aligned}\tag{A.3}$$

Therefore

$$\begin{aligned}\theta_{MAP} &= \arg \max_{\theta} \log P(\mathbf{X}|\theta)P(\theta) \\ &= \arg \max_{\theta} \log \prod_i P(\mathbf{x}_i|\theta)P(\theta) \\ &= \arg \max_{\theta} \sum_i \log P(\mathbf{x}_i|\theta) + \log P(\theta).\end{aligned}\tag{A.4}$$

We can see that the difference between MLE and MAP is the final term  $\log P(\theta)$ , which represents the prior distribution of the parameter  $\theta$ .

### A.1.3 Significance test

Significance test is a procedure for assessing the truthfulness of a claim about the observed data. The claim is also called the *null hypothesis* ( $H_0$ ).  $H_0$  is usually about *no difference* for a specific value. Its opposite hypothesis is referred to as the *alternative hypothesis* ( $H_a$ ). If  $H_a$  states the parameter is larger or smaller than the value in  $H_0$ , it is defined as one-side alternative hypothesis. If  $H_a$  just claims that the value is different from that in  $H_0$ , it is two-side alternative hypothesis. Significance level  $\alpha$  is the probability that we make a mistake when  $H_0$  is true but the significance test result suggests that we should reject  $H_0$ . In most cases, we select  $\alpha$  from  $\{0.001, 0.005, 0.01, 0.05\}$ . Before the significance test is conducted, the value of  $\alpha$  should be determined.  $p$ -value is used as the statistic evidence of the observed data, which is the probability that  $H_0$  is true. Therefore if  $p$ -value  $< \alpha$ , we should reject  $H_0$ . Otherwise, we should accept  $H_0$ .

The procedure of significance test can be summarized as follows:

- State null hypothesis  $H_0$  and alternative hypothesis  $H_a$ .
- Decide significance level  $\alpha$  and select a suitable testing method according to the observed data.
- Find the  $p$ -value by using a table or statistical software.
- Compare  $p$ -value with  $\alpha$  and decide whether  $H_0$  should be rejected or accepted.

Here, we introduce two most commonly used testing methods ( $z$ -test and  $t$ -test) in detail.

#### $z$ -test

$z$ -test is used when the sample size is large or the variances are known. The data are assumed to have a normal distribution.

- *One-sample  $z$ -test.* It is used to compare the sample mean  $\bar{x}$  and the population mean  $\mu$ .  $z$ -score =  $\frac{\bar{x} - \mu}{\sigma / \sqrt{n}}$ , where  $\sigma$  is the known population standard deviation and  $n$  is the sample size. After obtaining the  $z$ -score, we can find the corresponding  $p$ -value with a  $p$ -value table or a statistical software. If the  $p$ -value is less than the selected significance level  $\alpha$ , we reject  $H_0$ . Otherwise, we accept  $H_0$ .
- *Two-sample  $z$ -test.* It is used to compare the sample means of two groups of samples.  $z$ -score =  $\frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$ , where  $\bar{x}_1, \bar{x}_2$  are sample means,  $\mu_1, \mu_2$  are population means,  $\sigma_1, \sigma_2$  are population standard deviations, and  $n_1, n_2$  are the sample sizes. The remaining procedure is the same as one sample  $z$ -test.

#### $t$ -test

$t$ -test is used when the sample size is small (e.g.,  $n < 30$ ) and the variances are unknown. The data are also assumed to have a normal distribution.

- *One-sample  $t$ -test.* It is used to compare the sample mean  $\bar{x}$  and the population mean  $\mu$ .  $t = \frac{\bar{x} - \mu}{s / \sqrt{n}}$ , where  $s$  is the sample standard deviation and  $n$  is the sample size. After obtaining  $t$ , we can find the corresponding  $p$ -value with a  $p$ -value table or statistical software. If the  $p$ -value is less than the selected significance level  $\alpha$ , we reject the  $H_0$ . Otherwise, we accept  $H_0$ .
- *Two-sample  $t$ -test.* It is used to compare the sample means of two groups of samples.  $t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$ , where  $\bar{x}_1, \bar{x}_2$  are sample means,  $\mu_1, \mu_2$  are population means,  $s_1, s_2$  are sample



standard deviations, and  $n_1, n_2$  are the sample sizes. The remaining procedure is the same as one sample  $t$ -test.

### A.1.4 Density estimation

The goal of Density Estimation (DE) is to estimate an underlying probability distribution (e.g., probability density function) of a random variable using a set of observed data. To be specific, given a set of  $n$  observations,  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ , that are sampled from an unknown distribution  $P$ , DE aims to recover the probability density function generating the data. Density estimation serves as a fundamental and essential technique in a variety of tasks, such as regression, classification, and clustering.

Existing approaches for density estimation can be characterized into the following two categories, including (1) nonparametric, where no assumption on the probability distributions is specified, and (2) parametric, where a particular form of probability density function is given and DE aims to estimate the corresponding parameters.

For nonparametric density estimation, we do not confine the form of probability distribution,  $P$ , with specified distribution parameters. We introduce four nonparametric methods for density estimation:

- **Histogram.** Histogram is probably the simplest and most adopted probability density estimator. For convenience, we assume that  $\mathbf{x}_i \in [0, 1]$  where  $i \in \{1, \dots, n\}$  and  $P(\mathbf{x}_i) = 0$  for  $\mathbf{x}_i \notin [0, 1]$ . Histogram first partitions the region  $[0, 1]$  into  $M$  bins that are defined as intervals of width  $h = \frac{1}{M}$ . Therefore given a data point  $\mathbf{x}_i$  and its associated bin  $B$ , the probability density can be estimated as

$$\hat{P}(\mathbf{x}_i) = \frac{\text{\#data points in } B}{n} \frac{1}{h}. \quad (\text{A.5})$$

The intuition behind histogram is to assign the same probability density for data points that are close to each other (i.e., in the same bin).

- **Naive Estimator.** The true probability density  $P(\mathbf{x}_i)$  can be also defined as

$$P(\mathbf{x}_i) = \lim_{h \rightarrow 0} \frac{1}{2h} P\{\mathbf{x}_i - h < \mathbf{x} < \mathbf{x}_i + h\}, \quad (\text{A.6})$$

which can be estimated as  $\hat{P}(\mathbf{x}_i) = \frac{\text{\#data points in } (\mathbf{x}_i - h, \mathbf{x}_i + h)}{2nh}$ .  $\hat{P}(\mathbf{x}_i)$  can be further represented as

$$\hat{P}(\mathbf{x}_i) = \frac{1}{nh} \sum_{t=1}^n g\left(\frac{\mathbf{x}_i - \mathbf{x}_t}{h}\right), \quad (\text{A.7})$$

where  $g(x) = \frac{1}{2}$  for  $x \in (-1, 1)$ , and 0 otherwise. Compared to histogram, the naive estimator has  $2h$ -width bins that are centered at the given data points.

- **Kernel Estimator.** By replacing the term  $g(\cdot)$  in naive estimator with a nonnegative and symmetric function (called kernel function)  $K(z)$  that satisfies

$$\int_{-\infty}^{\infty} K(z) dz = 1, \quad (\text{A.8})$$

we obtain the kernel density estimator as follows:

$$\hat{P}(z) = \frac{1}{nh} \sum_{t=1}^n K\left(\frac{z - \mathbf{x}_t}{h}\right), \quad (\text{A.9})$$

where  $h$  is called bandwidth and is a smoothing parameter in kernel density estimator. Intuitively, the kernel density estimator smooths the input data  $X_i$  into small density bumps where the bump shape is determined by the kernel function and  $h$  decides the bump width. The final estimation is obtained by aggregating all the corresponding values of input data. A common choice of the kernel function is Gaussian distribution that is (1) nonnegative and (2) symmetric.

- **Nearest Neighbor.** The nearest neighbor method aims to capture the local information of input data based on distances. Having defined the distance using  $L_2$  norm between data points,  $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ , we denote  $d_k(y)$  as the distance between  $y$  and its  $k$ th nearest neighbor. The nearest neighbor density estimator is as follows:

$$\hat{P}(y) = \frac{k-1}{2nd_k(y)}. \quad (\text{A.10})$$

Let  $d_{i,k}$ ,  $K(\cdot)$  be the distance between  $\mathbf{x}_i$  and its  $k$ th nearest data point in  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \setminus \{\mathbf{x}_i\}$  and kernel function, respectively. We further obtain a variant of kernel estimator as

$$\hat{P}(y) = \frac{1}{n} \sum_{i=1}^n \frac{1}{hd_{i,k}} K\left(\frac{y - \mathbf{x}_i}{hd_{i,k}}\right), \quad (\text{A.11})$$

where the bandwidth is determined by the distance  $d_{i,k}$ . We can observe that for region with sparse data points (i.e., large  $d_{i,k}$ ), the kernel function will be flatter.

For parametric density estimation, we assume the form of the probability density function is known (e.g., Gaussian) and we aim to estimate the associated parameters of the distribution (e.g., mean, variance). In general, given a set of  $n$  observed data points, i.e.,  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  and a specified form of distribution parameterized by  $\theta$ , i.e.,  $P(\mathbf{x}; \theta)$ , the goal is to compute  $\hat{\theta}$  such that  $P(\mathbf{x}; \hat{\theta})$  best fits the observation. According to the input data, parametric density estimation can be (1) supervised where labeled data samples are available, (2) unsupervised where label information is not available (e.g., clustering), and (3) semisupervised where only a portion of observed data points have labels. We can use either maximum likelihood estimation (MLE) or maximum a posteriori (MAP) to estimate the corresponding parameters. MLE and MAP are introduced in the previous section.

### A.1.5 Bias-variance tradeoff

In statistics, the bias is the difference between the estimator's expected prediction and the true values that the estimator aims to predict. High bias represents that the estimator cannot accurately capture the correlation between input features and the true output, which leads to the problem of underfitting. Variance, on the other hand, represents the variability of the estimator in terms of prediction. An estimator with high variance is very sensitive to small perturbations of input features, which causes the overfitting problem. In general, we can reduce the bias by increasing the model complexity (e.g., adding

more learnable parameters); however, higher model complexity might cause the overfitting issue of the model (i.e., high variance), which results in a tension in terms of simultaneously minimizing both bias and variance. We leverage bias-variance decomposition to mathematically analyze the generalization error of a learning algorithm as follows.

Given a set of i.i.d. samples,  $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$  that are drawn from  $P(X, Y)$ , we denote the learned model from  $D$  as  $f_D(\cdot)$ . The generalization error (i.e., expected test error) in the squared loss is computed as

$$\mathbb{E}_{(\mathbf{x}, y) \sim P}[(f_D(\mathbf{x}) - y)^2] = \int_{\mathbf{x}, y} (f_D(\mathbf{x}) - y)^2 Pr(\mathbf{x}, y) d\mathbf{x} dy. \quad (\text{A.12})$$

We can think of  $f_D(\cdot)$  as a random variable of learning model from data set  $D$ . Then, we can represent the expected learning model as  $\bar{f}(\cdot)$ . The expected test error can be further decomposed as

$$\begin{aligned} \mathbb{E}_{\mathbf{x}, y, D}[(f_D(\mathbf{x}) - y)^2] &= \mathbb{E}_{\mathbf{x}, y, D}[(f_D(\mathbf{x}) - \bar{f}(\mathbf{x}) + \bar{f}(\mathbf{x}) - y)^2] \\ &= \mathbb{E}_{\mathbf{x}, D}[(f_D(\mathbf{x}) - \bar{f}(\mathbf{x}))^2] + \mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - y)^2]. \end{aligned} \quad (\text{A.13})$$

Eq. (A.13) holds because  $\mathbb{E}_{\mathbf{x}, y, D}[(f_D(\mathbf{x}) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)] = 0$ . We find that the first term  $\mathbb{E}_{\mathbf{x}, D}[(f_D(\mathbf{x}) - \bar{f}(\mathbf{x}))^2]$  on the right-hand side is the variance, and similarly, we decompose the second term  $\mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - y)^2]$  as

$$\begin{aligned} \mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - y)^2] &= \mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - \bar{y} + \bar{y} - y)^2] \\ &= \mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - \bar{y})^2] + \mathbb{E}_{\mathbf{x}, y}[(\bar{y} - y)^2]. \end{aligned} \quad (\text{A.14})$$

where the second step is due to the fact that  $\mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - \bar{y})(\bar{y} - y)] = 0$ . The first term  $\mathbb{E}_{\mathbf{x}, y}[(\bar{f}(\mathbf{x}) - \bar{y})^2]$  represents squared bias, and the second term  $\mathbb{E}_{\mathbf{x}, y}[(\bar{y} - y)^2]$  is the data noise that cannot be removed.

From the above decomposition, we observe that the expected test error (i.e., generalization error) is composed of the bias, variance, and data noise.

## A.1.6 Cross-validation and Jackknife

### *Cross-validation*

Data used for training data mining models are usually split into a training set, a validation set, and a test set. The model with the best performance on the validation set is finally evaluated on the test set. However, this split method is not suitable for applications with scarce data, because we want to train the model with as much data as possible. Cross-validation can be used to address this challenge. For  $k$ -fold cross validation, the original data set is split into  $k$  chunks. In each time, the model is trained with  $k - 1$  chunks (i.e., folds) and validated on the remaining chunk. This procedure is repeated  $k$  times and the final performance score (e.g., the classification accuracy) is the average of the performance scores from  $k$  runs. If we let  $k = n$ , where  $n$  is the total number of samples in the data set, this technique is called leave-one-out cross-validation.

### Jackknife

Given a data set  $\mathcal{D}_n \triangleq \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ , where  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $y_i \in \mathcal{Y}$  and samples are i.i.d. We would like to fit a prediction model  $f(\mathbf{x}; \theta) : \mathbb{R}^d \rightarrow \mathcal{Y}$  on the data set where  $\theta$  represents the model parameters. Suppose that we have a new test point  $(\mathbf{x}_{n+1}, y_{n+1})$ , the goal of Jackknife is to construct a confidence interval  $C_{n,\alpha}(\mathbf{x}_{n+1})$  such that the target value  $y_{n+1}$  is covered with a probability of at least  $1 - \alpha$ :

$$\mathbb{P}\{y_{n+1} \in C_{n,\alpha}(\mathbf{x}_{n+1})\} \geq 1 - \alpha. \quad (\text{A.15})$$

Let us define some notation first. Suppose  $\mathcal{R} = \{r_1, \dots, r_n\}$ ,  $\widehat{Q}_{n,\alpha}^+$  is defined as the  $(1 - \alpha)$  quantile of the empirical distribution of the set  $\mathcal{R}$ :

$$\widehat{Q}_{n,\alpha}^+(\mathcal{R}) \triangleq \text{the } \lceil (1 - \alpha)(n + 1) \rceil\text{-th smallest value in } \mathcal{R}. \quad (\text{A.16})$$

Similarly,  $\widehat{Q}_{n,\alpha}^-$  denotes the  $\alpha$  quantile of the empirical distribution:

$$\widehat{Q}_{n,\alpha}^-(\mathcal{R}) \triangleq \text{the } \lceil \alpha(n + 1) \rceil\text{-th smallest value in } \mathcal{R}. \quad (\text{A.17})$$

A straightforward solution to construct the confidence interval is to use the  $(1 - \alpha)$  quantile of the residuals  $|y_i - f(\mathbf{x}_i; \hat{\theta})|$  on the training data set. However, due to the overfitting problem, residuals on the training data set are typically smaller than the residual on the test point. Therefore the coverage probability of the interval is likely to be smaller than the target probability  $1 - \alpha$ . To address the overfitting problem, Jackknife computes the confidence interval with the leave-one-out residuals. For each  $i = 1, \dots, n$ , the prediction model  $f(\mathbf{x}; \hat{\theta}_i)$  is trained on the data set without  $i$ th sample  $\mathcal{D}_n \setminus \{(\mathbf{x}_i, y_i)\}$ , and the leave-one-out residual is computed as  $r_i = |y_i - f(\mathbf{x}_i; \hat{\theta}_i)|$ . Then,  $f(\mathbf{x}; \hat{\theta})$  is trained with the full training data, and the Jackknife interval is

$$C_{n,\alpha}(\mathbf{x}_{n+1}) = f(\mathbf{x}_{n+1}; \hat{\theta}) \pm \widehat{Q}_{n,\alpha}^+(\mathcal{R}), \quad (\text{A.18})$$

where  $\mathcal{R}$  is the set of leave-one-out residuals  $\mathcal{R} = \{r_1, \dots, r_n\}$ .

---

## A.2 Numerical optimization

For clarity, we use superscripts with parentheses (e.g.,  $(t)$ ,  $(t + 1)$ ) to distinguish parameters from different time steps (i.e., iterations).

### A.2.1 Gradient descent

Batch gradient descent

Given a loss function  $L(\boldsymbol{\theta}^{(t)})$  and the corresponding gradient information  $\nabla L(\boldsymbol{\theta}^{(t)})$  with respect to the model parameters  $\boldsymbol{\theta}^{(t)}$ . A simple yet often effective method to obtain the (local) minima of the loss function is *gradient descent* (GD), which is also known as *steepest descent*,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.19})$$

where  $\eta > 0$  is *learning rate* representing the step size of update. Note that if the loss function is evaluated on the whole data set, that is,

$$L(\boldsymbol{\theta}^{(t)}) = \sum_i L_i(\boldsymbol{\theta}^{(t)}), \quad (\text{A.20})$$

where  $i$  is the index of a data point, then the method is called *batch gradient descent*.

### Stochastic gradient descent

In order to (1) reduce the computation cost in each iteration and (2) render randomness into the optimization to escape from the local minima, *stochastic gradient descent (SGD)* works as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla L_i(\boldsymbol{\theta}^{(t)}), \quad (\text{A.21})$$

where the loss function is only evaluated on a single data point  $i$  that is randomly sampled from the whole data set. To balance the effectiveness and efficiency, an alternative method named *mini batch gradient descent* randomly samples a *mini batch* (e.g., composed by 32 data points) in each iteration and then evaluates the loss function and its gradient information on the sampled mini batch to guide the optimization process.

## A.2.2 Variants of gradient descent

### Momentum

*Momentum* prevents the oscillation of update by reusing the update vector from the previous step (i.e.,  $\Delta \boldsymbol{\theta}^{(t-1)}$ ) as follows:

$$\begin{aligned} \Delta \boldsymbol{\theta}^{(t)} &= \gamma \Delta \boldsymbol{\theta}^{(t-1)} + \eta \nabla L(\boldsymbol{\theta}^{(t)}) \\ \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \Delta \boldsymbol{\theta}^{(t)}. \end{aligned} \quad (\text{A.22})$$

Note that based on the selection of data points to calculate the loss and the corresponding gradient, momentum (and almost all the methods introduced later) works in either the batch, mini-batch, or stochastic fashions. The mini-batch style is the most common in practice.

### Adagrad

*Adagrad* is designed to adaptively tune the learning rate based on the gradients from previous steps. Its update formula is as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\mathbf{G}^{(t)} + \epsilon}} \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.23})$$

where  $\epsilon$  is a smooth hyperparameter to prevent divide-by-zero error, and  $\mathbf{G}^{(t)}$  is a diagonal matrix whose element  $\mathbf{G}^{(t)}[i, i]$  represents the sum of squared gradients with respect to the  $i$ th variable up to time step  $t$ . Hence if a variable receives a large gradient in the history, it will get a small learning rate at the current time step. Note that  $\frac{\eta}{\sqrt{\mathbf{G}^{(t)} + \epsilon}}$  is a matrix in general, and there is a matrix-vector multiplication between  $\frac{\eta}{\sqrt{\mathbf{G}^{(t)} + \epsilon}}$  and  $\nabla L(\boldsymbol{\theta}^{(t)})$ .

### Adadelta

*Adadelta* extends the idea of Adagrad by modifying the monotonically increasing sum of squared gradients into an exponentially decaying average. To be specific, the expected sum of squared gradients  $E(\mathbf{g}^2)$  is computed as follows:

$$E((\mathbf{g}^{(t)})^2) = \rho E((\mathbf{g}^{(t-1)})^2) + (1 - \rho)(\nabla L(\boldsymbol{\theta}^{(t)}))^2, \quad (\text{A.24})$$

where  $\rho$  is a decay hyperparameter and  $(\nabla L(\boldsymbol{\theta}^{(t)}))^2[i] = (\nabla L(\boldsymbol{\theta}^{(t)})[i])^2$ . By defining root mean square of gradients as  $\text{RMS}(\mathbf{g}^{(t)}) = \sqrt{E((\mathbf{g}^{(t)})^2) + \epsilon}$ , the update of the parameters can be represented as follows:

$$\Delta \boldsymbol{\theta}^{(t)} = \frac{\eta}{\text{RMS}(\mathbf{g}^{(t)})} \nabla L(\boldsymbol{\theta}^{(t)}). \quad (\text{A.25})$$

In addition, in order to match the unit between  $\Delta \boldsymbol{\theta}^{(t)}$  and  $\boldsymbol{\theta}^{(t)}$ , which has been neglected by SGD, Momentum and other optimizers, Adadelta calibrates the  $\Delta \boldsymbol{\theta}^{(t)}$  as

$$\Delta \boldsymbol{\theta}^{(t)} = \frac{\text{RMS}(\Delta \boldsymbol{\theta}^{(t-1)})}{\text{RMS}(\mathbf{g}^{(t)})} \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.26})$$

where  $\text{RMS}(\Delta \boldsymbol{\theta}^{(t)}) = \sqrt{E((\Delta \boldsymbol{\theta}^{(t)})^2) + \epsilon}$  and

$$E((\Delta \boldsymbol{\theta}^{(t)})^2) = \rho E((\Delta \boldsymbol{\theta}^{(t-1)})^2) + (1 - \rho)(\Delta \boldsymbol{\theta}^{(t)})^2. \quad (\text{A.27})$$

Finally, Adadelta updates parameters as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \Delta \boldsymbol{\theta}^{(t)}. \quad (\text{A.28})$$

### RMSprop

*RMSprop* shares the similar idea as Adadelta, although they are developed independently. RMSprop updates parameters as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{E((\mathbf{g}^{(t)})^2) + \epsilon}} \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.29})$$

where

$$E((\mathbf{g}^{(t)})^2) = \rho E((\mathbf{g}^{(t-1)})^2) + (1 - \rho)(\nabla L(\boldsymbol{\theta}^{(t)}))^2. \quad (\text{A.30})$$

### Adam

*Adaptive Moment Estimation (Adam)* is another method designed to adaptively adjust the learning rate by estimating two moments. The first and second moments of gradients are calculated as follows:

$$\begin{aligned} \mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \nabla L(\boldsymbol{\theta}^{(t)}) \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) (\nabla L(\boldsymbol{\theta}^{(t)}))^2. \end{aligned} \quad (\text{A.31})$$

With calibration of both two moments as follows:

$$\begin{aligned}\hat{\mathbf{m}}^{(t)} &= \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t} \\ \hat{\mathbf{v}}^{(t)} &= \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t},\end{aligned}\tag{A.32}$$

where  $\beta_1^t$  and  $\beta_2^t$  refer to the  $t$  power of  $\beta_1$  and  $\beta_2$ , Adam updates parameters as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(t)} + \epsilon}} \hat{\mathbf{m}}^{(t)}.\tag{A.33}$$

### AdaMax

*AdaMax* is a variant of Adam based on the infinity norm. Moment vectors are computed as follows:

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \nabla L(\boldsymbol{\theta}^{(t)}) \\ \mathbf{v}^{(t)} &= \max(\beta_2 \mathbf{v}^{(t-1)}, |\nabla L(\boldsymbol{\theta}^{(t)})|).\end{aligned}\tag{A.34}$$

Finally *AdaMax* updates parameters as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{(1 - \beta_1^t) \mathbf{v}^{(t)}} \mathbf{m}^{(t)},\tag{A.35}$$

where  $\beta_1^t$  refers to the  $t$  power of  $\beta_1$ .

## A.2.3 Newton's method

### Newton's method

Given a function  $f(\theta^{(t)})$  with respect to a single variable  $\theta$ , *Newton's method* aims to find the root of the function  $f(\theta^{(t)}) = 0$  by following approximation:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}.\tag{A.36}$$

Geometrically, the updated  $\theta^{(t+1)}$  is the intersection of the x-axis and the tangent of the function  $f$  at the old  $\theta^{(t)}$ .

Given a loss function  $L(\boldsymbol{\theta}^{(t)})$  with respect to parameters  $\boldsymbol{\theta}^{(t)}$ , both global and local minima satisfy  $\nabla L(\boldsymbol{\theta}^{(t)}) = 0$ . Hence by extending Eq. (A.36) into multivariable scenario and replacing  $f(\theta^{(t)})$  with  $\nabla L(\boldsymbol{\theta}^{(t)})$ , the Newton's method takes the form as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \mathbf{H}^{-1} \nabla L(\boldsymbol{\theta}^{(t)}),\tag{A.37}$$

where  $\mathbf{H}$  is the Hessian matrix of  $L$  with respect to the entries of  $\boldsymbol{\theta}^{(t)}$ . Newton's method works well when the problem is convex; that is, the Hessian matrix is positive definite.

### Quasi-Newton method

Notice that searching the extrema for an optimization problem in Eq. (A.37) requires the availability of the Hessian matrix  $\mathbf{H}$ , which may be violated in some scenarios, and the inverse matrix of Hessian is expensive to compute. In response, a family of approximations of the Hessian matrix are proposed and they are named as *quasi-Newton methods*.

The general formula of quasi-Newton methods is as follows:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - (\mathbf{B}^{(t)})^{-1} \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.38})$$

where  $\mathbf{B}^{(t)}$  is the approximation of the Hessian matrix  $\mathbf{H}$ . A good approximation of Hessian matrix (i.e.,  $\mathbf{B}$ ) should satisfy the following condition:

$$\mathbf{B}^{(t+1)}(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) = \nabla L(\boldsymbol{\theta}^{(t+1)}) - \nabla L(\boldsymbol{\theta}^{(t)}), \quad (\text{A.39})$$

which is known as the *secant equation*. Before we move forward, let us first consider a 1-D scenario from Eq. (A.39) that

$$f''(\theta^{(t+1)}) = \frac{f'(\theta^{(t+1)}) - f'(\theta^{(t)})}{\theta^{(t+1)} - \theta^{(t)}}. \quad (\text{A.40})$$

Hence we can easily rewrite the 1-D version of Eq. (A.38) based on Eq. (A.40) as follows:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \frac{\theta^{(t)} - \theta^{(t-1)}}{f'(\theta^{(t)}) - f'(\theta^{(t-1)})} f'(\theta^{(t)}), \quad (\text{A.41})$$

which is known as the *secant method*, a member of the quasi-Newton methods. In fact, various quasi-Newton methods generalize the secant method into a multidimension scenario. Here, we introduce one of the most commonly used methods. It is called the *Broyden–Fletcher–Goldfarb–Shanno (BFGS) method*. For other quasi-Newton methods, please refer to the bibliographic notes.

Despite of the success of the secant method in one-dimensional scenario, grafting it into multidimensional scenario is not trivial since it is underdetermined to obtain  $\mathbf{B}$  from Eq. (A.39). Since the matrix inverse operation is time-consuming, we directly study the inverse matrix  $\mathbf{B}^{-1}$  as follows:

$$\begin{aligned} \min_{(\mathbf{B}^{(t+1)})^{-1}} \quad & \|(\mathbf{B}^{(t+1)})^{-1} - (\mathbf{B}^{(t)})^{-1}\|_F \\ \text{s.t.} \quad & ((\mathbf{B}^{(t+1)})^{-1})^T = (\mathbf{B}^{(t+1)})^{-1} \\ & \Delta\boldsymbol{\theta}^{(t)} = (\mathbf{B}^{(t+1)})^{-1} \mathbf{y}^{(t)}, \end{aligned} \quad (\text{A.42})$$

where  $\Delta\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}$ ,  $\mathbf{y}^{(t)} = \nabla L(\boldsymbol{\theta}^{(t+1)}) - \nabla L(\boldsymbol{\theta}^{(t)})$ , and  $\|\cdot\|_F$  denotes the Frobenius norm. The first constraint of Eq. (A.42) requires the inverse matrix of  $\mathbf{B}$  to be symmetric, and the second constraint is the secant equation (i.e., Eq. (A.39)). In order to keep symmetric and ensure the closeness between  $(\mathbf{B}^{(t+1)})^{-1}$  and  $(\mathbf{B}^{(t)})^{-1}$ , the BFGS method incrementally updates  $\mathbf{B}$  by two rank-one matrices as follows:

$$\mathbf{B}^{(t+1)} = \mathbf{B}^{(t)} + \mathbf{U}^{(t)} + \mathbf{V}^{(t)} = \mathbf{B}^{(t)} + \alpha \mathbf{u}\mathbf{u}^T + \beta \mathbf{v}\mathbf{v}^T, \quad (\text{A.43})$$



where  $\mathbf{u}$  and  $\mathbf{v}$  are two linearly independent vectors and  $\alpha$  and  $\beta$  are two constants. Recall the secant equation from Eq. (A.39) we have

$$\mathbf{B}^{(t)} \Delta \boldsymbol{\theta}^{(t)} + \alpha \mathbf{u} \mathbf{u}^T \Delta \boldsymbol{\theta}^{(t)} + \beta \mathbf{v} \mathbf{v}^T \Delta \boldsymbol{\theta}^{(t)} = \mathbf{y}^{(t)}. \quad (\text{A.44})$$

By choosing  $\mathbf{u} = \mathbf{y}^{(t)}$  and  $\mathbf{v} = \mathbf{B}^{(t)} \Delta \boldsymbol{\theta}^{(t)}$  we obtain that  $\alpha = \frac{1}{(\mathbf{y}^{(t)})^T \Delta \boldsymbol{\theta}^{(t)}}$  and  $\beta = \frac{-1}{(\Delta \boldsymbol{\theta}^{(t)})^T \mathbf{B}^{(t)} \Delta \boldsymbol{\theta}^{(t)}}$ . Hence Eq. (A.43) can be rewritten as follows:

$$\mathbf{B}^{(t+1)} = \mathbf{B}^{(t)} + \frac{\mathbf{y}^{(t)} (\mathbf{y}^{(t)})^T}{(\mathbf{y}^{(t)})^T \Delta \boldsymbol{\theta}^{(t)}} - \frac{\mathbf{B}^{(t)} \Delta \boldsymbol{\theta}^{(t)} (\Delta \boldsymbol{\theta}^{(t)})^T \mathbf{B}^{(t)}}{(\Delta \boldsymbol{\theta}^{(t)})^T \mathbf{B}^{(t)} \Delta \boldsymbol{\theta}^{(t)}}. \quad (\text{A.45})$$

To incrementally update  $\mathbf{B}^{-1}$  to prevent multiple matrix inverse operations, based on the *Sherman–Morrison–Woodbury formula*, we have

$$(\mathbf{B}^{(t+1)})^{-1} = \left( \mathbf{I} - \frac{\Delta \boldsymbol{\theta}^{(t)} (\mathbf{y}^{(t)})^T}{(\mathbf{y}^{(t)})^T \Delta \boldsymbol{\theta}^{(t)}} \right) (\mathbf{B}^{(t)})^{-1} \left( \mathbf{I} - \frac{\mathbf{y}^{(t)} (\Delta \boldsymbol{\theta}^{(t)})^T}{(\mathbf{y}^{(t)})^T \Delta \boldsymbol{\theta}^{(t)}} \right) + \frac{\Delta \boldsymbol{\theta}^{(t)} (\Delta \boldsymbol{\theta}^{(t)})^T}{(\mathbf{y}^{(t)})^T \Delta \boldsymbol{\theta}^{(t)}}. \quad (\text{A.46})$$

## A.2.4 Coordinate descent

*Coordinate descent* is a technique that solves the problem by breaking up the whole into parts. Specifically, given a loss function  $L(\boldsymbol{\theta})$  with respect to the parameters  $\boldsymbol{\theta}$ , in each iteration, we minimize the loss function with respect to the first parameter  $\boldsymbol{\theta}[1]$ , and then minimize it with respect to the second parameter  $\boldsymbol{\theta}[2]$ , and repeat this process to all the parameters in a cycling fashion. A more general practice extended from coordinate descent is named as *block coordinate descent*, which minimizes the loss function with respect to a subset of parameters.

A typical example where coordinate descent works well is sparse coding whose loss function is as follows:

$$L(\boldsymbol{\Theta}, \boldsymbol{\Phi}) = \|\mathbf{X} - \boldsymbol{\Theta} \boldsymbol{\Phi}\|_F^2 + \lambda \sum_i \|\boldsymbol{\Theta}[i]\|_1, \quad (\text{A.47})$$

where the training data  $\mathbf{X}$  is aimed to be reconstructed by the coefficient matrix  $\boldsymbol{\Theta}$  and a set of overcomplete basis vectors (i.e., the columns of  $\boldsymbol{\Phi}$ ). The sparse coding problem requires the coefficients  $\boldsymbol{\Theta}$  to have few nonzero entries or have few entries far from zero by the second term of Eq. (A.47). The problem itself is not convex with respect to  $\boldsymbol{\Theta}$  and  $\boldsymbol{\Phi}$ . However, if we fix either one of them and minimize the loss function with respect to the other one, the problem is convex. Hence through block coordinate descent method, it is promising to solve this problem more efficiently by updating two parameters alternatively.

## A.2.5 Quadratic programming

If the objective function of an optimization problem is quadratic and the constraints are affine, this problem is named as a *quadratic program (QP)*. Generally, objective functions of the quadratic programs

fall into the following form:

$$\begin{aligned}
 \min \quad & a + \mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} \\
 \text{s.t.} \quad & \mathbf{P} \mathbf{x} \leq \mathbf{d} \\
 & \mathbf{Q} \mathbf{x} = \mathbf{e}.
 \end{aligned} \tag{A.48}$$

Without loss of generality, we can assume that matrix  $\mathbf{C}$  is symmetric.

A typical example is *constrained regression analysis*

$$\begin{aligned}
 \min_{\mathbf{x}} \quad & \|\mathbf{A} \mathbf{x} - \mathbf{b}\|_2^2 \\
 \text{s.t.} \quad & \mathbf{l}[i] \leq \mathbf{x}[i] \leq \mathbf{u}[i], \forall i,
 \end{aligned} \tag{A.49}$$

which is equivalent to

$$\begin{aligned}
 \min_{\mathbf{x}} \quad & \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - 2 \mathbf{b}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{b} \\
 \text{s.t.} \quad & \mathbf{l}[i] \leq \mathbf{x}[i] \leq \mathbf{u}[i], \forall i.
 \end{aligned} \tag{A.50}$$

There are versatile solutions toward various QPs. In this section, we provide a solution toward the *equality-constrained QPs (EQPs)* and solutions for the general QPs can be found in the bibliographic notes. EQPs are QPs where only equality constraints exist, and their objective functions fall into the following form:

$$\begin{aligned}
 \min \quad & \mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} \\
 \text{s.t.} \quad & \mathbf{Q} \mathbf{x} = \mathbf{e},
 \end{aligned} \tag{A.51}$$

where the constant term is dropped. If  $\mathbf{x}^*$  is a solution of an EQP, a necessary condition is to have a  $\boldsymbol{\lambda}^*$  vector satisfying the following *KKT system*:

$$\begin{bmatrix} \mathbf{C} & \mathbf{Q}^T \\ \mathbf{Q} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \mathbf{b} + \mathbf{C} \mathbf{x} \\ \mathbf{Q} \mathbf{x} - \mathbf{e} \end{bmatrix}, \tag{A.52}$$

where  $\mathbf{x}$  is an estimation of the solution and  $\mathbf{x}^* = \mathbf{x} + \mathbf{p}$ . If  $\mathbf{Q}$  is positive definite, based on Eq. (A.52) we can obtain  $\boldsymbol{\lambda}^*$  from the following equation:

$$(\mathbf{Q} \mathbf{C}^{-1} \mathbf{Q}^T) \boldsymbol{\lambda}^* = \mathbf{Q} \mathbf{C}^{-1} \mathbf{b} + \mathbf{e}. \tag{A.53}$$

Hence  $\mathbf{p}$  can be obtained from the following equation:

$$\mathbf{C} \mathbf{p} = \mathbf{Q}^T \boldsymbol{\lambda}^* - \mathbf{b} - \mathbf{C} \mathbf{x}, \tag{A.54}$$

by which we can obtain the final solution  $\mathbf{x}^*$ .

## A.3 Matrix and linear algebra

### A.3.1 Linear system $\mathbf{Ax} = \mathbf{b}$

In this section, we discuss three types of linear systems  $\mathbf{Ax} = \mathbf{b}$ : (1)  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is a standard square matrix, (2)  $\mathbf{A} \in \mathbb{R}^{n \times m}$ ,  $n > m$ ,  $\mathbf{A}$  has more rows than columns (overdetermined), and (3)  $\mathbf{A} \in \mathbb{R}^{n \times m}$ ,  $n < m$ ,  $\mathbf{A}$  has less rows than columns (underdetermined).

#### *Standard square system*

If  $\mathbf{A}$  is invertible, we could directly solve it with the matrix inverse  $\mathbf{A}^{-1}$ :

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}. \quad (\text{A.55})$$

Iterative methods provide an indirect way to solve the linear system. Widely used iterative methods include Jacobi method, Richardson method, Krylov subspace methods, and so on. The key idea of iterative methods is to choose suitable  $\mathbf{S}$  and  $\mathbf{d}$  and set up an iteration in the form of

$$\mathbf{x}^{(k+1)} = \mathbf{S}\mathbf{x}^{(k)} + \mathbf{d}. \quad (\text{A.56})$$

In each iteration, the residual error  $\|\mathbf{x}^{(k)} - \mathbf{x}^*\|$  is reduced where  $\mathbf{x}^*$  is the solution to  $\mathbf{Ax} = \mathbf{b}$ . After several iterations,  $\mathbf{x}^{(k)}$  converges to  $\mathbf{x}^*$ .

A preconditioning method aims to find a simpler matrix  $\mathbf{P}$ , which is close to  $\mathbf{A}$ , that is,  $(\mathbf{A} - \mathbf{P})$  has low rank or small norm. Then, we solve  $\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}$  instead of  $\mathbf{Ax} = \mathbf{b}$  because working on  $\mathbf{P}^{-1}\mathbf{A}$  is usually faster. Some typical choices of  $\mathbf{P}$  include the diagonal matrix with the main diagonal of  $\mathbf{A}$ , triangular matrix copying the corresponding part of  $\mathbf{A}$ , and so on.

#### *Overdetermined system*

Suppose  $\mathbf{A}$  has independent columns  $\text{rank}(\mathbf{A}) = m$ , we have

$$\mathbf{Ax} = \mathbf{b} \iff \mathbf{A}^\top \mathbf{Ax} = \mathbf{A}^\top \mathbf{b} \iff \mathbf{x} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{b}, \quad (\text{A.57})$$

which gives the solution minimizing  $\|\mathbf{Ax} - \mathbf{b}\|^2$ .

When  $\mathbf{A}^\top \mathbf{A}$  is not invertible, the least square solution is  $\mathbf{x} = \mathbf{A}^+ \mathbf{b}$  where  $\mathbf{A}^+$  is the pseudo-inverse of  $\mathbf{A}$ . Specifically, we first perform singular value decomposition (SVD) on  $\mathbf{A}$  and obtain  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ , then we have  $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top$ .

#### *Underdetermined system*

For underdetermined system, there are many least square solutions, and it turns out that  $\mathbf{A}^+ \mathbf{b}$  is the least square solution with the smallest norm  $\|\mathbf{x}\|^2$ . Similar to the overdetermined system, when  $\mathbf{A}$  has independent rows  $\text{rank}(\mathbf{A}) = n$ , we have

$$\mathbf{Ax} = \mathbf{b} \iff \mathbf{x} = \mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^{-1} \mathbf{b}. \quad (\text{A.58})$$

In some cases, we want to regularize the linear system with an  $l_2$  penalty term. Thus our optimization goal becomes

$$\text{Minimize } \|\mathbf{Ax} - \mathbf{b}\|^2 + \delta^2 \|\mathbf{x}\|_2^2, \quad (\text{A.59})$$

where  $\delta^2$  is the weight of the penalty term. It is equivalent to solve  $(\mathbf{A}^\top \mathbf{A} + \delta^2 \mathbf{I}) \mathbf{x} = \mathbf{A}^\top \mathbf{b}$ , and this approach is called ridge regression.

### A.3.2 Norms of vectors and matrices

#### *Norms of vectors*

The norm of a vector is defined as a function mapping from vector space  $V$  to a real value:  $\|\cdot\| : V \rightarrow \mathbb{R}$ . Given a vector  $\mathbf{v}$ , it measures the length of the vector  $\|\mathbf{v}\|$ . For each  $\mathbf{v}, \mathbf{w} \in V$  and  $c \in \mathbb{R}$ , these three properties always hold:

- Positive definite:  $\|\mathbf{v}\| \geq 0$  and  $\|\mathbf{v}\| = 0 \iff \mathbf{v} = \mathbf{0}$ .
- Absolutely homogeneous:  $\|c\mathbf{v}\| = |c|\|\mathbf{v}\|$ , where  $|c|$  is the absolute value of  $c$ .
- Triangle inequality:  $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$ .

Here, we give the definitions of some widely used vector norms:

- $\ell^1$  norm (Manhattan Norm):  $\|\mathbf{v}\|_1 = \sum_i |\mathbf{v}[i]|$ .
- $\ell^2$  norm (Euclidean Norm):  $\|\mathbf{v}\|_2 = \sqrt{\sum_i \mathbf{v}[i]^2} = \sqrt{\mathbf{v}^\top \mathbf{v}}$ .
- $\ell^\infty$  norm:  $\|\mathbf{v}\|_\infty = \max_i |\mathbf{v}[i]|$ .

#### *Norms of matrices*

Induced matrix norm could be induced by the vector norm with:

$$\|\mathbf{A}\| = \max_{\mathbf{v} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{v}\|}{\|\mathbf{v}\|}. \quad (\text{A.60})$$

In addition to the properties that the vector norm satisfies, induced matrix norm has the following properties, that is, for all  $\mathbf{A}, \mathbf{B}$  and  $\mathbf{v}$ ,  $\|\mathbf{A}\mathbf{v}\| \leq \|\mathbf{A}\|\|\mathbf{v}\|$  and  $\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$ .

Here, we introduce how to compute some typical matrix norms:

- $\ell^1$  norm is the largest  $\ell^1$  norm of the columns of  $\mathbf{A}$ :  $\|\mathbf{A}\|_1 = \max_j \sum_i |\mathbf{A}[i, j]|$ .
- $\ell^2$  norm is the largest singular value of  $\mathbf{A}$ :  $\|\mathbf{A}\|_2 = \sqrt{\text{maxeig}(\mathbf{A}^\top \mathbf{A})}$ , where  $\text{maxeig}(\cdot)$  denotes the maximum eigenvalue.
- $\ell^\infty$  norm is the largest  $\ell^1$  norm of the rows of  $\mathbf{A}$ :  $\|\mathbf{A}\|_\infty = \max_i \sum_j |\mathbf{A}[i, j]|$ .
- Frobenius norm comes from the singular values  $\sigma_i$  ( $i = 1, \dots, r$ ) of  $\mathbf{A}$ , that is,  $\|\mathbf{A}\|_F^2 = \sigma_1^2 + \dots + \sigma_r^2$ , where  $r$  is the rank of matrix  $\mathbf{A}$ . It could also be derived with:  $\|\mathbf{A}\|_F^2 = \sum_{ij} |\mathbf{A}[i, j]|^2$ .

### A.3.3 Matrix decompositions

#### *Eigenvalues and eigendecomposition*

Given a square matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , if  $\lambda \in \mathbb{R}$  and  $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$  satisfy that  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ , we call  $\lambda$  an eigenvalue of  $\mathbf{A}$  and  $\mathbf{x}$  is the corresponding eigenvector.<sup>1</sup> From  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ , we have  $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$ . Since  $\mathbf{x}$  is not

<sup>1</sup> We often make eigenvectors have unit length.

a zero vector, we have that  $\mathbf{A} - \lambda\mathbf{I}$  is not invertible, and its determinant must be zero. To compute all the eigenvalues of  $\mathbf{A}$ , we need to solve the roots of equation  $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ . Since the equation has  $n$  degrees for  $\lambda$ , there are  $n$  eigenvalues in total, with possible duplicates.

There are some useful relationships between eigenvalues and matrix's traces and determinants:

- The sum of eigenvalues equals to the trace:  $\text{tr}(\mathbf{A}) = \sum_{i=1}^n \lambda_i$ .
- The product of eigenvalues equals to the determinant:  $\det(\mathbf{A}) = \prod_{i=1}^n \lambda_i$ .

When matrix  $\mathbf{A}$  has  $n$  independent eigenvectors, it could be decomposed into

$$\mathbf{A} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^{-1}, \quad (\text{A.61})$$

where the columns of  $\mathbf{X}$  are the  $n$  eigenvectors of  $\mathbf{A}$ , and  $\mathbf{\Lambda}$  is a diagonal matrix whose diagonal entries are the  $n$  eigenvalues of  $\mathbf{A}$ . This decomposition is named as eigendecomposition. Notice the symmetric matrix always has  $n$  independent eigenvectors and could always be eigendecomposed. From  $\mathbf{A} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^{-1}$ , we have  $\mathbf{A}^k = \mathbf{X}\mathbf{\Lambda}^k\mathbf{X}^{-1}$ , which means the eigenvalues of  $\mathbf{A}^k$  are  $\lambda_1^k, \dots, \lambda_n^k$ . Besides, for each  $\mathbf{v} \in \mathbb{R}^n$ , it could be written as  $\mathbf{v} = c_1\mathbf{x}_1 + \dots + c_n\mathbf{x}_n$ . This gives an easier way to compute  $\mathbf{A}^k\mathbf{v}$ :

$$\mathbf{A}^k\mathbf{v} = c_1\lambda_1^k\mathbf{x}_1 + \dots + c_n\lambda_n^k\mathbf{x}_n. \quad (\text{A.62})$$

### **Singular value decomposition (SVD)**

Given a matrix  $\mathbf{A}^{m \times n}$  with rank  $r$ , the SVD of  $\mathbf{A}$  is written as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \quad (\text{A.63})$$

where  $\mathbf{U} \in \mathbb{R}^{m \times n}$  is the orthogonal matrix containing orthonormal eigenvectors of  $\mathbf{A}\mathbf{A}^T$ ,  $\mathbf{V} \in \mathbb{R}^{n \times n}$  is the orthogonal matrix containing orthonormal eigenvectors of  $\mathbf{A}^T\mathbf{A}$ ,  $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$  is a matrix that satisfies  $\Sigma_{ii} = \sigma_i, i = 1, \dots, r$ , and the other entries are all zeros. Here, we call  $\sigma_1, \dots, \sigma_r$  singular values. The singular values are all positive and in the descending order. Notice that different from eigendecomposition, SVD is applicable for any rectangle matrix.

To compute SVD, we first perform eigendecomposition on  $\mathbf{A}^T\mathbf{A}$  and obtain the orthonormal eigenvectors  $\mathbf{v}_1, \dots, \mathbf{v}_r$ . From the form of SVD, we know that  $\mathbf{A}\mathbf{v}_k = \mathbf{u}_k\sigma_k, k = 1, \dots, r$ . Hence  $\mathbf{u}_k = \frac{\mathbf{A}\mathbf{v}_k}{\sigma_k}, k = 1, \dots, r$ . For the last  $n - r$  columns of  $\mathbf{V}$ , we have  $\mathbf{A}\mathbf{v}_k = \mathbf{0}, k = r + 1, \dots, n$ . Therefore  $\mathbf{v}_{r+1}, \dots, \mathbf{v}_n$  are the orthonormal basis from the nullspace of  $\mathbf{A}$ . Similarly,  $\mathbf{u}_{r+1}, \dots, \mathbf{u}_m$  are the orthonormal basis from the nullspace of  $\mathbf{A}^T$ .

With SVD,  $\mathbf{A}$  could be written as the sum of rank one matrices:

$$\mathbf{A} = \sigma_1\mathbf{u}_1\mathbf{v}_1^T + \dots + \sigma_r\mathbf{u}_r\mathbf{v}_r^T. \quad (\text{A.64})$$

This provides important factors for matrix approximation. Specifically, singular values  $\sigma_1, \sigma_2, \dots, \sigma_r$  are ordered according to its importance. The first term  $\sigma_1\mathbf{u}_1\mathbf{v}_1^T$  is the closest rank one matrix to  $\mathbf{A}$ , which means  $\|\mathbf{A} - \mathbf{B}\|$  achieves the minimum value in terms of both  $l_2$  norm and Frobenius norm only if rank one matrix  $\mathbf{B} = \sigma_1\mathbf{u}_1\mathbf{v}_1^T$ . Furthermore,  $\mathbf{A}_k = \sigma_1\mathbf{u}_1\mathbf{v}_1^T + \dots + \sigma_k\mathbf{u}_k\mathbf{v}_k^T$  is the closest rank  $k$  approximation to  $\mathbf{A}$ .

### A.3.4 Subspace

*Subspace* of a *vector space* is an important concept in linear algebra. In this section, we first introduce the concept of the *vector space* and then introduce the concept of the *subspace*. We also present several typical examples of subspaces.

#### Vector space

Formally, a vector space  $VS$  is a set of vectors, such that

1. the addition of any two vectors  $\forall \mathbf{x}, \mathbf{y} \in VS$  satisfies  $\mathbf{x} + \mathbf{y} \in VS$ ;
2. the multiplication of any vector  $\forall \mathbf{x} \in VS$  and any real number  $\forall \lambda \in \mathbb{R}$  satisfies  $\lambda \cdot \mathbf{x} \in VS$ .

#### Subspace

The definition of the *subspace* is similar to the vector space.

Formally, a subspace  $SS$  of a vector space  $VS$  satisfies the following:

1. it is a nonempty subset of  $VS$ ;
2. the addition of any two vectors  $\forall \mathbf{x}, \mathbf{y} \in SS$  satisfies  $\mathbf{x} + \mathbf{y} \in SS$ ;
3. the multiplication of any vector  $\forall \mathbf{x} \in SS$  and any real number  $\forall \lambda \in \mathbb{R}$  satisfies  $\lambda \cdot \mathbf{x} \in SS$ .

According to the above definition of the subspace, it is obvious that there are two *trivial subspaces* of a vector space  $VS$ : *zero vector space*  $Z$ , which only contains the zero vector (i.e., the origin), and *the vector space itself*  $VS$ , which contains all the vectors within  $VS$ .

In the  $\mathbb{R}^n$  space, in addition to the two trivial spaces: the set of  $\mathbf{0}$  and  $\mathbb{R}^n$ , any hyperplane containing  $\mathbf{0}$  is also a subspace of  $\mathbb{R}^n$ . However, a hyperplane that does not contain  $\mathbf{0}$  is not a subspace of  $\mathbb{R}^n$ . For example, in the  $\mathbb{R}^2$  space, any line passing through  $\mathbf{0}$  is a subspace of  $\mathbb{R}^2$ , but a line that does not pass through  $\mathbf{0}$  is not a subspace of  $\mathbb{R}^2$ .

The concept of subspace is closely related to the concept of *span*. Formally, given a set of vectors  $S$ , then its span  $span(S)$  is a set of vectors that contains all of the linear combinations of vectors in  $S$ :

$$span(S) = \left\{ \sum_{i=1}^k \lambda_i \mathbf{x}_i \mid k \in \mathbb{N}, \mathbf{x}_i \in S, \lambda_i \in \mathbb{R} \right\}. \quad (\text{A.65})$$

A subspace can be constructed from a set of vectors. Given a set of vectors  $S = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ , where  $\forall \mathbf{x}_i \in VS$  and  $i \in [1, \dots, m]$ , then  $span(S)$  is a subspace of  $VS$ . On the other hand, for any subspace  $SS$ , we could always find a set of vectors  $S$ , such that its span is  $SS$ :  $span(S) = SS$ .

Given a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , the span of its rows is a subspace of  $\mathbb{R}^m$ , and the span of its columns is a subspace of  $\mathbb{R}^n$ . These two subspaces are referred to as the *row space* and *column space*, respectively.

Given a linear equation  $\mathbf{Ax} = \mathbf{0}$ , where  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{x} \in \mathbb{R}^n$ , the solution of this equation is called *null space* and denoted as  $null(\mathbf{A})$ , which is a subspace of  $\mathbb{R}^n$ :

$$null(\mathbf{A}) = \{\mathbf{x} \mid \mathbf{Ax} = \mathbf{0}\}. \quad (\text{A.66})$$

### A.3.5 Orthogonality

The concept of *orthogonality* in linear algebra is a generalization of *perpendicularity* in elementary geometry, which denotes the relation between two lines where they meet at 90 degrees. Orthogonality is an important concept for studying the space and subspace.

#### Orthogonal vectors

Two vectors  $\mathbf{x} \in \mathbb{R}^n$  and  $\mathbf{y} \in \mathbb{R}^n$  are said to be *orthogonal* if their *inner product* is zero:

$$\mathbf{x}^T \mathbf{y} = \sum_{i=1}^n \mathbf{x}[i] \cdot \mathbf{y}[i] = 0. \quad (\text{A.67})$$

If  $\mathbf{x}^T \mathbf{y} = 0$ , then the angle between  $\mathbf{x}$  and  $\mathbf{y}$  is  $90^\circ$ ; if  $\mathbf{x}^T \mathbf{y} < 0$ , then the angle is greater than  $90^\circ$ ; if  $\mathbf{x}^T \mathbf{y} > 0$ , then the angle is less than  $90^\circ$ .

A useful fact of the orthogonality is that if nonzero vectors  $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$  are mutually orthogonal, then they are *linearly independent*.

If we further let  $\mathbf{x}$  and  $\mathbf{y}$  to be unit vectors:  $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$ , then they are *orthonormal* vectors.

When studying geometry or linear function within a subspace or a space, it is always convenient to find a set of *basis vectors* of the subspace or space to make the calculation simple. For example, any vector  $\mathbf{x} \in \mathbb{R}^2$ , can be easily represented by a linear combination of the *coordinate vectors*  $\mathbf{e}_1 = [1, 0]^T$  and  $\mathbf{e}_2 = [0, 1]^T$ , which represent the  $x$ -axis and  $y$ -axis:  $\mathbf{x} = \mathbf{x}[1]\mathbf{e}_1 + \mathbf{x}[2]\mathbf{e}_2$ . In addition to the natural orthonormal basis (i.e., coordinate vectors), there are also other popular orthonormal basis, such as  $\mathbf{v}_1 = [\cos \theta, \sin \theta]^T$  and  $\mathbf{v}_2 = [-\sin \theta, \cos \theta]^T$ .

#### Orthogonal subspaces

Given two subspaces  $SS_1$  and  $SS_2$ , these two subspaces are orthogonal, if any vector  $\mathbf{x}$  in  $SS_1$  and any vector  $\mathbf{y}$  in  $SS_2$  are orthogonal:  $\mathbf{x}^T \mathbf{y} = 0, \forall \mathbf{x} \in SS_1, \forall \mathbf{y} \in SS_2$ .

In the  $\mathbb{R}^2$  space, all the lines passing through the zero vector  $\mathbf{0}$  are its subspaces. If the angle between the two lines is  $90^\circ$ , then they are orthogonal. Besides, the zero subspace  $\{\mathbf{0}\}$  is orthogonal to all the subspaces of  $\mathbb{R}^2$ .

There are two closely related subspaces of  $\mathbf{A}\mathbf{x} = 0$ : the row space of  $\mathbf{A}$  and the null space of  $\mathbf{A}$ . These two subspaces are orthogonal to each other. This is because the inner product of any row of  $\mathbf{A}$  and any vector  $\mathbf{x}^*$  satisfying  $\mathbf{A}\mathbf{x}^* = 0$  is zero:  $\mathbf{A}[i, :]\mathbf{x} = 0$ .

*Orthogonal complement* is an important concept related to the orthogonal subspace. Formally, given a subspace  $SS$  of the vector space  $VS$ , the space of all vectors which is orthogonal to  $SS$  is called the *orthogonal complement* of  $SS$ , which is denoted as  $SS^\perp$ .

According to this terminology, it is obvious that the null space of  $\mathbf{A}$  is the orthogonal complement for the row space of  $\mathbf{A}$ .

An interesting property of the subspace  $SS$  of  $VS$  and its orthogonal complement  $SS^\perp$  is that the dimension of  $VS$  equals to the dimension of  $SS$  adds the dimension of  $SS^\perp$ .

## A.4 Concepts and tools from signal processing

### A.4.1 Entropy

In information theory, the *entropy* of a random variable (r.v.) measures the amount of uncertainty of this random variable. Let  $X$  be a discrete random variable with the probability mass function (PMF)  $p_X(x) = \Pr(X = x, x \in \mathcal{X})$ . The entropy  $H(X)$  of the discrete random variable  $X$  is summarized as follows:

$$H(X) = - \sum_x p_X(x) \log p_X(x) = \mathbb{E}_{x \sim p_X}[-\log p_X(x)]. \quad (\text{A.68})$$

In the case of continuous random variable, the amount of uncertainty of such random variable is called *differential entropy*. Let  $Y$  be a continuous random variable with PDF  $p_Y(y)$ , the differential entropy  $H(Y)$  of the continuous random variable  $Y$  is summarized as follows:

$$H(Y) = - \int_y p_Y(y) \log p_Y(y) dx = \mathbb{E}_{y \sim p_Y}[-\log p_Y(y)]. \quad (\text{A.69})$$

#### Remarks.

- We assume  $0 \log 0 = 0$ . The rationale behind this assumption is due to  $\lim_{x \rightarrow 0} x \log x = 0$ .
- If the base of logarithm is 2, the entropy is measured in bits. If the base of logarithm is  $e$ , the entropy is measured in nats. Unless otherwise specified, the base of logarithm is 2 through this section.

#### Other commonly used entropy measures.

- *Joint Entropy*. Given a joint random variable  $(X, Y)$  of two random variables  $X$  and  $Y$  with PMF  $p_{XY}(x, y)$ , the *joint entropy*  $H(X, Y)$  of the joint random variable  $(X, Y)$  is summarized as follows:

$$H(X, Y) = - \sum_{x,y} p_{XY}(x, y) \log p_{XY}(x, y) = \mathbb{E}_{(x,y) \sim p_{XY}}[-\log p_{XY}(x, y)]. \quad (\text{A.70})$$

- *Conditional Entropy*. Given two random variables  $X$  and  $Y$ , the *conditional entropy*  $H(X|Y)$  of the random variable  $X$  given the random variable  $Y$  is summarized as follows:

$$H(X|Y) = - \sum_{x,y} p_{XY}(x, y) \log \frac{p_{XY}(x, y)}{p_Y(y)} = \mathbb{E}_{(x,y) \sim p_{XY}}[-\log \frac{p_{XY}(x, y)}{p_Y(y)}], \quad (\text{A.71})$$

where  $p_Y(y)$  is the marginal PMF of random variable  $Y$ .

#### Properties of entropy.

- Entropy is always nonnegative, that is,  $H(X) \geq 0, \forall X$ .
- Entropy follows the chain rule, that is,  $H(X, Y) = H(X) + H(Y|X)$ .
- Entropy  $H(X)$  is a concave function with respect to PMF  $p_X(x)$ .
- Entropy achieves the maximum value when the distribution is uniform, that is,  $H(X) \leq \log n$  and  $H(X) = \log n$  if  $p_X(x) = \frac{1}{n}$ .
- If  $X$  and  $Y$  are two independent random variables,



- its joint entropy is  $H(X + Y) = H(X) + H(Y)$ .
- its conditional entropy is  $H(X|Y) = H(X)$ .
- Entropy will not increase with the information on another random variable, that is,  $H(X|Y) \leq H(X)$ .
- (Fano's inequality) The intuition of Fano's inequality is that the probability of making a wrong estimate  $\tilde{X}$  of a random variable  $X$  when using another random variable  $Y$  depends on how certain we are about  $X$  given  $Y$ . Given a Markov chain  $X \rightarrow Y \rightarrow \tilde{X}$ , where  $\tilde{X}$  is an estimation of  $X$  by applying some function on  $Y$  (i.e.,  $f(Y) = \tilde{X}$ ), let  $e$  represent the occurrence of  $X \neq \tilde{X}$  and  $P(e) = P(X \neq \tilde{X})$ , then we have

$$H(X|Y) \leq H(X|\tilde{X}) \leq H(e) + P(e) \log |\mathcal{X}|, \quad (\text{A.72})$$

where  $\mathcal{X}$  is the support set that includes all possible values of  $X$ , and  $|\mathcal{X}|$  is the cardinality of  $X$ .

#### A.4.2 Kullback-Leibler divergence (KL-divergence)

In information theory, the *Kullback-Leibler divergence* (KL-divergence) or *relative entropy* is a measure of the difference between two probability distributions. Given a random variable  $X$ , a true distribution  $p_X$ <sup>2</sup> and a target distribution  $q_X$ , the KL-divergence  $D_{KL}(p_X||q_X)$  of distributions  $p_X$  and  $q_X$  is summarized as follows:

$$\begin{aligned} D_{KL}(p_X||q_X) &= \mathbb{E}_{x \sim p_X} \left[ \log \frac{p_X(x)}{q_X(x)} \right] \\ &= \begin{cases} \sum_x p_X(x) \log \frac{p_X(x)}{q_X(x)}, & X \text{ is discrete r.v.} \\ \int_{x \sim p_X} p_X(x) \log \frac{p_X(x)}{q_X(x)} dx, & X \text{ is continuous r.v.} \end{cases} \end{aligned} \quad (\text{A.73})$$

**Remarks.** We assume that the following conventions hold in order to validate the definition of KL-divergence.

- $0 \log \frac{0}{0} = 0$
- $0 \log \frac{0}{q_X(x)} = 0$
- $p_X(x) \log \frac{p_X(x)}{0} = \infty$

With that being said, if for some value in the true distribution  $x \sim p_X(x)$  with  $p_X(x) > 0$  and  $q_X(x) = 0$ , we have  $D_{KL}(p_X||q_X) = \infty$ .

#### Properties of KL-divergence.

- KL-divergence is always nonnegative, that is,  $D_{KL}(p||q) \geq 0, \forall p, q$ .
- KL-divergence is not symmetric, that is,  $D_{KL}(p||q) \neq D_{KL}(q||p)$ .
- KL-divergence is not a metric because it is not symmetric and does not follow triangle inequality.
- KL-divergence is 0 if and only if  $p_X = q_X$  for a random variable  $X$ , that is,  $p_X = q_X \Leftrightarrow D_{KL}(p||q) = 0$ .

<sup>2</sup> Here, we violate our general naming conventions and use italic lowercase letters to represent a probability distribution.

- KL-divergence follows chain rule, that is,  $D_{KL}(p_{XY}||q_{XY}) = D_{KL}(p_X||q_X) + D_{KL}(p_{Y|X}||q_{Y|X})$ .
- KL-divergence is a convex function with respect to PMFs  $p_X$  and  $q_X$ .

### A.4.3 Mutual information

In information theory, the *mutual information* of two random variables measures the amount of information we gain on one random variable with the prior knowledge on observing the other random variable. Given two random variables  $X$  and  $Y$ , the mutual information  $I(X; Y)$  of random variables  $X$  and  $Y$  is summarized as follows:

$$\begin{aligned}
 I(X; Y) &= \mathbb{E}_{(x,y) \sim p_{XY}} \left[ \log \frac{p_{XY}(x, y)}{p_X(x)p_Y(y)} \right] \\
 &= \begin{cases} \sum_{(x,y)} p_{XY}(x, y) \log \frac{p_{XY}(x,y)}{p_X(x)p_Y(y)}, & X, Y \text{ are discrete r.v.} \\ \int_{(x,y) \sim p_{XY}} p_{XY}(x, y) \log \frac{p_{XY}(x,y)}{p_X(x)p_Y(y)} dx dy, & X, Y \text{ are continuous r.v.} \end{cases} \quad (\text{A.74})
 \end{aligned}$$

where  $p_{XY}$  is the joint distribution of the joint random variable  $(X, Y)$ ,  $p_X$  is the marginal distribution of random variable  $X$ , and  $p_Y$  is the marginal distribution of random variable  $Y$ .

**Relationship between mutual information and KL-divergence.** In terms of KL-divergence, mutual information  $I(X; Y)$  of two random variables  $X$  and  $Y$  is equivalent to the KL-divergence between the joint distribution  $p_{XY}$  and the product of their marginal distributions  $p_X \otimes p_Y$ . The relationship between mutual information and KL-divergence is summarized as follows:

$$I(X; Y) = D_{KL}(p_{XY}||p_X \otimes p_Y). \quad (\text{A.75})$$

**Relationship between mutual information and entropy.** In terms of entropy, mutual information of two random variables measures how much uncertainty we lose in one random variable by observing the other random variable. The relationship between mutual information and entropy is summarized as follows:

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X, Y), \quad (\text{A.76})$$

where the last equality holds by applying the chain rule of entropy.

**Conditional mutual information.** Given three random variables  $X$ ,  $Y$ , and  $Z$ , the conditional mutual information of the random variables  $X$  and  $Y$  given the random variable  $Z$  is summarized as follows:

$$I(X; Y|Z) = \mathbb{E}_{(x,y,z) \sim p_{XYZ}} \left[ \log \frac{p_{X,Y|Z}(x, y|z)}{p_{X|Z}(x|z)p_{Y|Z}(y, z)} \right] = H(X|Z) - H(X|Y, Z). \quad (\text{A.77})$$

#### Properties of mutual information.

- Mutual information is nonnegative, that is,  $I(X; Y) \geq 0, \forall X, Y$ .
- Mutual information is symmetric, that is,  $I(X; Y) = I(Y; X)$ .
- Mutual information of a random variable with itself is the entropy of that random variable, that is,  $I(X; X) = H(X)$ .

- Mutual information of two independent random variables is 0, that is,  $I(X; Y) = 0$  if  $X$  is independent to  $Y$ .
- Mutual information follows the chain rule, that is,  $I(X; Y, Z) = I(X; Y) + I(X; Z|Y)$ , where  $I(X; Y, Z)$  is the mutual information of  $X$  and joint random variable  $(Y, Z)$ .
- Data processing inequality. Suppose the random variable  $Y$  is obtained by applying some function  $f$  on the random variable  $X$  (i.e.,  $Y = f(X)$ ) and the random variable  $Z$  is obtained by processing the random variable  $Y$  (e.g., applying some function  $g$  on the random variable  $Y$  to have  $Z = g(Y)$ ). Data processing inequality guarantees that the mutual information between  $X$  and  $Z$  can never be larger than the mutual information between  $X$  and  $Y$ . Mathematically, given a Markov chain  $X \rightarrow Y \rightarrow Z$  of three random variables, we have

$$I(X; Z) \leq I(X; Y), \quad (\text{A.78})$$

where the equality holds if and only if  $I(X; Y|Z) = 0$ .

- The *variation of information* of two random variables, which is a metric, can be inferred by their mutual information and joint entropy, that is,  $d(X, Y) = H(X, Y) - I(X; Y)$ .

#### A.4.4 Discrete Fourier transform (DFT) and fast Fourier transform (FFT)

Unless otherwise specified, we use sequence to represent discrete-time sequence in this section and use  $N$  to represent the length of a finite discrete-time sequence or the length of a period of a periodic discrete-time sequence.

**Finite sequence.** A *finite sequence*  $x[n]$  of length  $N$  is the sequence of the following form<sup>3</sup>:

$$x[n] = \begin{cases} \text{some number,} & 0 \leq n \leq N - 1 \\ 0, & \text{otherwise.} \end{cases} \quad (\text{A.79})$$

**Relationship between finite sequence and periodic sequence.** It is intuitive that any finite sequence  $x[n]$  is associated with a corresponding periodic sequence  $\tilde{x}[n]$ , that is,

$$\tilde{x}[n] = \sum_{t=-\infty}^{+\infty} x[n - tN] = x[(n \bmod N)], \quad (\text{A.80})$$

where mod. is the modulo operation.

Similarly, a finite sequence  $x[n]$  can always be extracted from its corresponding periodic sequence  $\tilde{x}[n]$ , that is,

$$x[n] = \begin{cases} \tilde{x}[n], & n \in \{0, \dots, N - 1\} \\ 0, & \text{otherwise.} \end{cases} \quad (\text{A.81})$$

<sup>3</sup> Here, we violate the general naming conventions. We use (1) italic lowercase letters to represent a sequence of signals and (2)  $[i]$  to represent the  $i$ th position in this sequence.

**Discrete Fourier series (DFS) of a periodic sequence.** Given a periodic sequence  $\tilde{x}[n]$  with period  $N$ , the analysis and synthesis equations of the DFS  $\tilde{X}[k]$  ( $k \in \{0, \dots, N-1\}$ ) of the periodic sequence  $\tilde{x}[n]$  is summarized as follows<sup>4</sup>:

$$\begin{aligned} \text{Analysis equation: } \tilde{X}[k] &= \sum_{n=0}^{N-1} e^{-i(2\pi/N)kn} \tilde{x}[n] \\ \text{Synthesis equation: } \tilde{x}[n] &= \sum_{k=0}^{N-1} e^{i(2\pi/N)kn} \tilde{X}[k]. \end{aligned} \quad (\text{A.82})$$

**DFT of a finite sequence.** Since a finite sequence  $x[n]$  is one period of a periodic sequence  $\tilde{x}[n]$ , in order to maintain the duality between time and frequency domains, the DFT  $X[k]$  of the finite sequence  $x[n]$  corresponds to one period of DFS  $\tilde{X}[k]$  of its corresponding periodic sequence  $\tilde{x}[n]$ . As a result, the analysis and synthesis equations of the DFT  $X[k]$  of a finite sequence  $x[n]$  is summarized as follows:

$$\begin{aligned} \text{Analysis equation: } X[k] &= \sum_{n=0}^{N-1} e^{-i(2\pi/N)kn} x[n] \\ \text{Synthesis equation: } x[n] &= \sum_{k=0}^{N-1} e^{i(2\pi/N)kn} X[k]. \end{aligned} \quad (\text{A.83})$$

It is worth mentioning that the matrix  $\mathbf{W}$  with  $\mathbf{W}[i, j] = e^{-i(2\pi/N)ij}$  is called the *DFT matrix*.

### Properties of DFT.

- Linearity. Given two finite sequences  $x_1[n]$ ,  $x_2[n]$  and their associated DFT  $X_1[k]$ ,  $X_2[k]$ , if  $x_3[n] = ax_1[n] + bx_2[n]$ , the DFT of  $x_3[n]$  is

$$X_3[k] = aX_1[k] + bX_2[k]. \quad (\text{A.84})$$

- Given a finite sequence  $x[n]$  and its DFT  $X[k]$ , if we have a circular shift of length  $m$  in the time window and get the new finite sequence  $x_1[n] = x[((n-m) \bmod N)]$ , the DFT of the new finite sequence  $x_1[k]$  is

$$X_1[k] = e^{-i(2\pi/N)km} X[k]. \quad (\text{A.85})$$

- The vector  $\mathbf{u}_k = [e^{-i(2\pi/N)k0}, e^{-i(2\pi/N)k1}, \dots, e^{-i(2\pi/N)k(N-1)}]$  forms an orthogonal basis over the set of  $N$ -dimensional complex vectors, i.e.,  $\mathbf{u}_k^T \mathbf{u}_k^* = N$ .
- Parseval's theorem. Given two sequences  $x_1[n]$  and  $x_2[n]$  and their associated DFT  $X_1[k]$  and  $X_2[k]$ , we have

$$\sum_{n=0}^{N-1} x_1[n] x_2^*[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_1[k] X_2^*[k], \quad (\text{A.86})$$

<sup>4</sup> Here, we violate the general naming conventions and use italic uppercase letters to represent a sequence in frequency domain.

where  $x_2^*[n]$  is the conjugate transpose of the sequence  $x_2[n]$  and  $X_2^*[k]$  is the conjugate transpose of  $X_2[k]$ .

- Duality. Given a sequence  $x[n]$  and its DFT  $X[k]$ , the DFT of  $X[k]$  is  $Nx[(-k \bmod N)]$
- For a finite sequence  $x[n]$  and its DFT  $X[k]$ , if all values are real numbers, we have

$$X[k] = X^*[(-k \bmod N)], \quad (\text{A.87})$$

which is called *even symmetry*. Here,  $X^*[k]$  is the conjugate transpose of  $X[k]$ ; if all values are imaginary numbers, we have

$$X[k] = -X^*[(-k \bmod N)], \quad (\text{A.88})$$

which is called *odd symmetry*.

- Given three finite sequences  $x_1[n]$ ,  $x_2[n]$ , and  $x_3[n]$  and their corresponding DFT  $X_1[k]$ ,  $X_2[k]$ , and  $X_3[k]$ , if  $X_3[k] = X_1[k]X_2[k]$ , we have

$$x_3[n] = \sum_{m=0}^{N-1} x_2[m]x_1[((n-m) \bmod N)]. \quad (\text{A.89})$$

**FFT.** FFT is a set of algorithms that efficiently compute the DFT of a finite sequence, based on theories ranging from arithmetic to number theory. FFT is widely applied in many real world scenarios like mathematics, engineering and music. The key principle of FFT is to decompose the computation of the DFT of a length- $N$  sequence into computing several smaller subsequences. Some representative FFT algorithms include the prime-factor algorithm (PFA), the Cooley-Tuck algorithm, the Goertzel's algorithm, and the Winograd's algorithm. For details of FFT algorithms, please refer to the bibliographic notes.

---

## A.5 Bibliographic notes

Probability density function (PDF) and significance test are introduced in many statistic books (Larsen and Marx [LM05] and Casella and Berger [CB21]). For MLE and MAP, they are covered in many machine learning books (Bishop [Bis06b] and James, Witten, Hastie, and Tibshirani [JWHT]). Cross-validation is described by Bishop [Bis06b], and Jackknife is introduced by Barber, Candes, Ramdas, and Tibshirani [BCRT21]. GD, SGD, variants of GD, coordinate descent, and Newton's method are introduced in many primary and advanced books such as Goodfellow, Bengio, and Courville [GBC16] and Boyd, Boyd, and Vandenberghe [BBV04]. Dennis and Moré provide a detailed survey of Quasi-Newton methods [DM77]. For quadratic programming (QP), it is systematically introduced by Boyd, Boyd, and Vandenberghe [BBV04]. The solutions for QP are versatile such as active set methods (Murty and Yu [MY88]), Interior-point methods (Wright [Wri97]), and many more. For basic concepts, theories, and applications of linear algebra, they are comprehensively introduced by Strang [Str93, Str19] and Leon, Bica, and Hohn [LBH06].

Density estimation, which aims to approximate the underlying probability density function, has been an essential topic in statistics textbooks (Silverman [Sil18] and Devroye and Lugosi [DL12]). Density estimation using kernel tricks is extensively investigated in related research works (Botev, Grotowski,

and Kroese [BGK10] and Sheather and Jones [SJ91]). In addition to shallow methods, neural networks-based density estimation approaches are well explored in a variety of applications (Likas [Lik01] and Magdon-Ismail and Atiya [MIA99]). For instance, Magdon-Ismail and Atiya leveraged neural networks for density estimation [MIA99]. Density estimation is adopted in a variety of high-impact applications, including wind power forecasting (He and Li [HL18]) and predicting data distribution for large-scale sensor network (Nakamura and Hasegawa [NH16]). The bias-variance trade-off represents a competing scenario where it is impossible to simultaneously reduce learning model's bias and variance. Many efforts have been devoted to understanding the bias-variance trade-off (Belkin, Hsu, Ma, and Mandal [BHMM19]). For example, Yang et al. re-investigated the bias-variance problem in neural networks [YYY+20]. Li et al. incorporated the bias-variance trade-off to improve the performance of face recognition [LSG11].

Many information theory-related books (e.g., Cover [Cov99]) cover the fundamental concepts of entropy, Kullback-Leibler (KL)-divergence and mutual information. Oppenheim [Opp99] systematically studied discrete Fourier transform (DFT), and introduced several fast Fourier transform (FFT) algorithms for efficient computation of DFT. The prime-factor algorithm (PFA) was firstly proposed by Good [Goo58]. Cooley and Tuck proposed the divide-and-conquer based Cooley-Tuck algorithm [CT65], which is later found to be a re-invention of the algorithm proposed by Gauss [Gau66] to interpolate trajectories of two asteroids. Goertzel proposed Goertzel's algorithm [G+58] by solving a digital filtering problem with the Goertzel filter. Winograd proposed the Winograd's algorithm [Win78] for efficient DFT computation by drawing ideas from cyclic convolution computation.

This page intentionally left blank

# Bibliography

- [AAD<sup>+</sup>96] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, S. Sarawagi, On the computation of multidimensional aggregates, in: Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96), Bombay, India, Sept. 1996, pp. 506–521.
- [AAL<sup>+</sup>15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh, Vqa: visual question answering, in: Proc. 2015 Int. Conf. Computer Vision (ICCV'15), Santiago, Chile, Dec. 2015, pp. 2425–2433.
- [AAP01] R. Agarwal, C.C. Aggarwal, V.V.V. Prasad, A tree projection algorithm for generation of frequent itemsets, *Journal of Parallel and Distributed Computing* 61 (2001) 350–371.
- [AB79] B. Abraham, G.E.P. Box, Bayesian analysis of some outlier problems in time series, *Biometrika* 66 (1979) 229–248.
- [AB00] Réka Albert, Albert-László Barabási, Topology of evolving networks: local events and universality, *Physical Review Letters* 85 (24) (2000) 5234.
- [ABA06] M. Agyemang, K. Barker, R. Alhaji, A comprehensive survey of numeric and symbolic outlier mining techniques, *Intelligent Data Analysis* 10 (2006) 521–538.
- [ABKS99] M. Ankerst, M. Breunig, H.-P. Kriegel, J. Sander, OPTICS: ordering points to identify the clustering structure, in: Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99), Philadelphia, PA, USA, June 1999, pp. 49–60.
- [ACF<sup>+</sup>18] Jacob Abernethy, Alex Chojnacki, Arya Farahi, Eric Schwartz, Jared Webb, Activeremediation: the search for lead pipes in flint, Michigan, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 5–14.
- [ACL00] William Aiello, Fan Chung, Linyuan Lu, A random graph model for massive graphs, in: Proc. 2000 ACM Symp. Theory of Computing (SOTC'00), Portland, OR, USA, May. 2000, pp. 171–180.
- [ACM<sup>+</sup>14] Marco Avvenuti, Stefano Cresci, Andrea Marchetti, Carlo Meletti, Maurizio Tesconi, Ears (earthquake alert and report system) a real time decision support system for earthquake crisis management, in: Proc. 2014 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'14), New York, NY, USA, Aug. 2014, pp. 1749–1758.
- [AD91] H. Almuallim, T.G. Dietterich, Learning with many irrelevant features, in: Proc. 1991 Nat. Conf. Artificial Intelligence (AAAI'91), Anaheim, CA, USA, July 1991, pp. 547–552.
- [AEEK99] M. Ankerst, C. Elsen, M. Ester, H.-P. Kriegel, Visual classification: an interactive approach to decision tree construction, in: Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99), San Diego, CA, USA, Aug. 1999, pp. 392–396.
- [AEMT00] K.M. Ahmed, N.M. El-Makky, Y. Taha, A note on “beyond market basket: generalizing association rules to correlations”, *SIGKDD Explorations* 1 (2000) 46–48.
- [AG60] F.J. Anscombe, I. Guttman, Rejection of outliers, *Technometrics* 2 (1960) 123–147.
- [Aga06] D. Agarwal, Detecting anomalies in cross-classified streams: a bayesian approach, *Knowledge and Information Systems* 11 (2006) 29–44.
- [AGAV09] E. Amigó, J. Gonzalo, J. Ariles, F. Verdejo, A comparison of extrinsic clustering evaluation metrics based on formal constraints, *Information Retrieval* 12 (2009).
- [Agg07] Charu C. Aggarwal, *An Introduction to Data Streams*, Springer, 2007.
- [Agg15a] Charu C. Aggarwal, *Data Classification*, Morgan Springer, 2015.
- [Agg15b] Charu C. Aggarwal, *Data Mining: The Textbook*, Springer, 2015.



- [Agg15c] Charu C. Aggarwal, *Outlier analysis*, in: *Data Mining*, Springer, 2015, pp. 237–263.
- [AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, in: *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, Seattle, WA, USA, June 1998, pp. 94–105.
- [AGM04] F.N. Afrati, A. Gionis, H. Mannila, Approximating a collection of frequent sets, in: *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'04)*, Seattle, WA, USA, Aug. 2004, pp. 12–19.
- [AGS97] R. Agrawal, A. Gupta, S. Sarawagi, Modeling multidimensional databases, in: *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, Birmingham, England, April 1997, pp. 232–243.
- [Aha92] D. Aha, Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms, *International Journal of Man-Machine Studies* 36 (1992) 267–287.
- [AHMJP12] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Gerald Penn, Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition, in: *Proc. 2012 Int. Conf. Acoustics, Speech and Signal Processing (ICASSP'12)*, Kyoto, Japan, March 2012, pp. 4277–4280.
- [AHS96] P. Arabie, L.J. Hubert, G. De Soete, *Clustering and Classification*, World Scientific, 1996.
- [AHWY03] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in: *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, Berlin, Germany, Sept. 2003, pp. 81–92.
- [AHWY04] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for projected clustering of high dimensional data streams, in: *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, Toronto, Canada, Aug. 2004, pp. 852–863.
- [AIS93] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)*, Washington, DC, USA, May 1993, pp. 207–216.
- [AK93] T. Anand, G. Kahn, Opportunity explorer: navigating large databases using knowledge discovery templates, in: *Proc. 1993 Workshop Knowledge Discovery in Databases (KDD'93)*, Washington, DC, USA, July 1993, pp. 45–51.
- [AKK18] Gowtham Atluri, Anuj Karpatne, Vipin Kumar, Spatio-temporal data mining: a survey of problems and methods, *ACM Computing Surveys* 51 (4) (August 2018).
- [AL99] Y. Aumann, Y. Lindell, A statistical theory for quantitative association rules, in: *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, USA, Aug. 1999, pp. 261–270.
- [All94] B.P. Allen, Case-based reasoning: business applications, *Communications of the ACM* 37 (1994) 40–42.
- [Alp11] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed., MIT Press, 2011.
- [AMG<sup>+</sup>20] Sina F. Ardabili, Amir Mosavi, Pedram Ghamisi, Filip Ferdinand, Annamaria R. Varkonyi-Koczy, Uwe Reuter, Timon Rabczuk, Peter M. Atkinson, Covid-19 outbreak prediction with machine learning, *Algorithms* 13 (10) (2020) 249.
- [AMS<sup>+</sup>96] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, T. Bollinger, The Quest data mining system, in: *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, Portland, OR, USA, Aug. 1996, pp. 244–249.
- [AP94] A. Aamodt, E. Plazas, Case-based reasoning: foundational issues, methodological variations, and system approaches, *AI Communications* 7 (1994) 39–52.
- [AP05] F. Angiulli, C. Pizzuti, Outlier mining in large high-dimensional data sets, *IEEE Transactions on Knowledge and Data Engineering* 17 (2005) 203–215.
- [APW<sup>+</sup>99] C.C. Aggarwal, C. Procopiuc, J. Wolf, P.S. Yu, J.-S. Park, Fast algorithms for projected clustering, in: *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, Philadelphia, PA, USA, June 1999, pp. 61–72.

- [ARV09] S. Arora, S. Rao, U. Vazirani, Expander flows, geometric embeddings and graph partitioning, *Journal of the ACM* 56 (2009) 5:1–5:37.
- [AS94a] R. Agrawal, R. Srikant, Fast algorithm for mining association rules in large databases, Research Report RJ 9839, IBM Almaden Research Center, San Jose, CA, USA, June 1994.
- [AS94b] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), Santiago, Chile, Sept. 1994, pp. 487–499.
- [AS95] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proc. 1995 Int. Conf. Data Engineering (ICDE'95), Taipei, Taiwan, Mar. 1995, pp. 3–14.
- [AS96] R. Agrawal, J.C. Shafer, Parallel mining of association rules: design, implementation, and experience, *IEEE Transactions on Knowledge and Data Engineering* 8 (1996) 962–969.
- [ASE17] Antreas Antoniou, Amos J. Storkey, Harrison Edwards, Data augmentation generative adversarial networks, *CoRR*, arXiv:1711.04340 [abs], 2017.
- [ASS00] E.L. Allwein, R.E. Shapire, Y. Singer, Reducing multiclass to binary: a unifying approach for margin classifiers, *Journal of Machine Learning Research* 1 (2000) 113–141.
- [ATK15] Leman Akoglu, Hanghang Tong, Danai Koutra, Graph based anomaly detection and description: a survey, *Data Mining and Knowledge Discovery* 29 (3) (2015) 626–688.
- [ATVF12] Leman Akoglu, Hanghang Tong, Jilles Vreeken, Christos Faloutsos, Fast and reliable anomaly detection in categorical data, in: Proc. 2012 Int. Conf. Information and Knowledge Management (CIKM'12), Maui, HI, USA, Oct. 2012, pp. 415–424.
- [AV07] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in: Proc. 2007 ACM-SIAM Symp. Discrete Algorithms (SODA'07), Tokyo, Japan, 2007, pp. 1027–1035.
- [AW10] Charu C. Aggarwal, Haixun Wang, *A Survey of Clustering Algorithms for Graph Data*, Springer US, Boston, MA, USA, 2010, pp. 275–301.
- [AXRP19] Bijaya Adhikari, Xinfeng Xu, Naren Ramakrishnan, B. Aditya Prakash, Epideep: exploiting embeddings for epidemic forecasting, in: Proc. 2019 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'19), Anchorage, AK, USA, Aug. 2019, pp. 577–586.
- [AY99] C.C. Aggarwal, P.S. Yu, A new framework for itemset generation, in: Proc. 1998 ACM Symp. Principles of Database Systems (PODS'98), Seattle, WA, USA, June 1999, pp. 18–24.
- [AY01] C.C. Aggarwal, P.S. Yu, Outlier detection for high dimensional data, in: Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01), Santa Barbara, CA, USA, May 2001, pp. 37–46.
- [AZ12] Charu C. Aggarwal, ChengXiang Zhai (Eds.), *Mining Text Data*, Springer, 2012.
- [BA97] L.A. Breslow, D.W. Aha, Simplifying decision trees: a survey, *Knowledge Engineering Review* 12 (1997) 1–40.
- [BA99] Albert-László Barabási, Réka Albert, Emergence of scaling in random networks, *Science* 286 (5439) (1999) 509–512.
- [Bag05] J. Andrew Bagnell, Robust supervised learning, in: Proc. 2005 AAAI Conf. Artificial Intelligence (AAAI'05), Pittsburgh, PA, USA, 2005, pp. 714–719.
- [Bai13] Eric Bair, Semi-supervised clustering methods, *Wiley Interdisciplinary Reviews: Computational Statistics* 5 (5) (2013) 349–361.
- [Bal12] Pierre Baldi, Autoencoders, unsupervised learning, and deep architectures, in: Proc. 2012 ICML Workshop Unsupervised and Transfer Learning (ICML'12), Bellevue, WA, USA, July 2012, pp. 37–49.
- [BAM18] Tadas Baltrušaitis, Chaitanya Ahuja, Louis-Philippe Morency, Multimodal machine learning: a survey and taxonomy, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2) (2018) 423–443.
- [Bar89] Horace B. Barlow, Unsupervised learning, *Neural Computation* 1 (3) (1989) 295–311.
- [Bay98] R.J. Bayardo, Efficiently mining long patterns from databases, in: Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA, USA, June 1998, pp. 85–93.

- [BB98] A. Bagga, B. Baldwin, Entity-based cross-document coreferencing using the vector space model, in: Proc. 1998 Annual Meeting of the Association for Computational Linguistics and Int. Conf. Computational Linguistics (COLING-ACL'98), Montreal, Canada, Aug. 1998.
- [BBD<sup>+</sup>02] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proc. 2002 ACM Symp. Principles of Database Systems (PODS'02), Madison, WI, USA, June 2002, pp. 1–16.
- [BBM02] Sugato Basu, Arindam Banerjee, Raymond J. Mooney, Semi-supervised clustering by seeding, in: Proc. 2002 Int. Conf. Machine Learning (ICML'02), San Francisco, CA, USA, 2002, pp. 27–34.
- [BBM04] Sugato Basu, Mikhail Bilenko, Raymond J. Mooney, A probabilistic framework for semi-supervised clustering, in: Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'04), New York, NY, USA, Association for Computing Machinery, 2004, pp. 59–68.
- [BBN<sup>+</sup>17] Amin Beheshti, Boualem Benattallah, Reza Nouri, Van Munin Chhieng, HuangTao Xiong, Xu Zhao, Coredb: a data lake service, in: Proc. 2017 Conf. Information and Knowledge Management (CIKM'17), New York, NY, USA, Nov. 2017, pp. 2451–2454.
- [BBV04] Stephen Boyd, Stephen P. Boyd, Lieven Vandenbergh, Convex Optimization, Cambridge University Press, 2004.
- [BC83] R.J. Beckman, R.D. Cook, Outlier...s, *Technometrics* 25 (1983) 119–149.
- [BCB15] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural machine translation by jointly learning to align and translate, in: Proc. 2015 Int. Conf. Learning Representation (ICLR'15), San Diego, CA, USA, May 2015.
- [BCG01] D. Burdick, M. Calimlim, J. Gehrke, MAFIA: a maximal frequent itemset algorithm for transactional databases, in: Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, April 2001, pp. 443–452.
- [BCG10] Bahman Bahmani, Abdur Chowdhury, Ashish Goel, Fast incremental and personalized pagerank, *Proceedings of the VLDB Endowment* 4 (3) (December 2010) 173–184.
- [BCP93] D.E. Brown, V. Corruble, C.L. Pittard, A comparison of decision tree classifiers with backpropagation neural networks for multimodal classification problems, *Pattern Recognition* 26 (1993) 953–961.
- [BCRT21] Rina Foygel Barber, Emmanuel J. Candes, Aaditya Ramdas, Ryan J. Tibshirani, Predictive inference with the jackknife+, *The Annals of Statistics* 49 (1) (2021) 486–507.
- [BCV13] Yoshua Bengio, Aaron C. Courville, Pascal Vincent, Representation learning: a review and new perspectives, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (8) (2013) 1798–1828.
- [BD<sup>+</sup>99] Kristin Bennett, Ayhan Demiriz, et al., Semi-supervised support vector machines, in: Proc. 1999 Conf. Neural Information Processing Systems (NIPS'99), Denver, CO, USA, Dec. 1999, pp. 368–374.
- [BDF<sup>+</sup>97] D. Barbará, W. DuMouchel, C. Faloutsos, P.J. Haas, J.H. Hellerstein, Y. Ioannidis, H.V. Jagadish, T. Johnson, R. Ng, V. Poosala, K.A. Ross, K.C. Servcik, The New Jersey data reduction report, *Buletin of the Technical Committee on Data Engineering* 20 (Dec. 1997) 3–45.
- [BDG96] A. Bruce, D. Donoho, H.-Y. Gao, Wavelet analysis, *IEEE Spectrum* 33 (Oct 1996) 26–35.
- [BDJ<sup>+</sup>05] D. Burdick, P. Deshpande, T.S. Jayram, R. Ramakrishnan, S. Vaithyanathan, OLAP over uncertain and imprecise data, in: Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05), Trondheim, Norway, Aug. 2005, pp. 970–981.
- [Ber06] P. Berkhin, in: *A Survey of Clustering Data Mining Techniques*, Springer Berlin Heidelberg, Berlin, 2006, pp. 25–71.
- [Bez81] J.C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, 1981.
- [BF13] Jimmy Ba, Brendan Frey, Adaptive dropout for training deep neural networks, in: Proc. 2013 Conf. Neural Information Processing Systems (NIPS'13), Lake Tahoe, NV, USA, Dec. 2013, pp. 3084–3092.

- [BFOS84] L. Breiman, J. Friedman, R. Olshen, C. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.
- [BFR98] P. Bradley, U. Fayyad, C. Reina, Scaling clustering algorithms to large databases, in: Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98), New York, NY, USA, Aug. 1998, pp. 9–15.
- [BG05] I. Ben-Gal, Outlier detection, in: O. Maimon, L. Rockach (Eds.), *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, Kluwer Academic, 2005.
- [BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, Enriching word vectors with subword information, *Transactions of the Association for Computational Linguistics* 5 (2016) 135–146.
- [BGK10] Zdravko I. Botev, Joseph F. Grotowski, Dirk P. Kroese, Kernel density estimation via diffusion, *The Annals of Statistics* 38 (5) (2010) 2916–2957.
- [BGKW03] C. Bucila, J. Gehrke, D. Kifer, W. White, DualMiner: a dual-pruning algorithm for itemsets with constraints, *Data Mining and Knowledge Discovery* 7 (2003) 241–272.
- [BGMP03] F. Bonchi, F. Giannotti, A. Mazzanti, D. Pedreschi, ExAnte: anticipated data reduction in constrained pattern mining, in: Proc. 2003 European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD'03), Cavtat-Dubrovnik, Croatia, Sept. 2003, pp. 59–70.
- [BGRS99] K.S. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is “nearest neighbor” meaningful?, in: Proc. 1999 Int. Conf. Database Theory (ICDT'99), Jerusalem, Israel, Jan. 1999, pp. 217–235.
- [BGV92] B. Boser, I. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: Proc. 1992 Conf. Computational Learning Theory (COLT'92), San Mateo, CA, USA, July 1992, pp. 144–152.
- [BH20] Yikun Ban, Jingrui He, Generic outlier detection in multi-armed bandit, in: Proc. 2020 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'20), March 2020, pp. 913–923.
- [BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off, *Proceedings of the National Academy of Sciences* 116 (32) (2019) 15849–15854.
- [Bis95] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
- [Bis06a] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
- [Bis06b] Christopher M. Bishop, *Pattern Recognition*, 2006.
- [BKNS00] M.M. Breunig, H.-P. Kriegel, R. Ng, J. Sander, LOF: identifying density-based local outliers, in: Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, USA, May 2000, pp. 93–104.
- [BLC<sup>+</sup>03] D. Barbará, Y. Li, J. Couto, J.-L. Lin, S. Jajodia, Bootstrapping a data mining intrusion detection system, in: Proc. 2003 ACM Symp. Applied Computing (SAC'03), March 2003.
- [BLCW09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, Jason Weston, Curriculum learning, in: Proc. 2009 Int. Conf. Machine Learning (ICML'09), Montreal, QC, Canada, June 2009, pp. 41–48.
- [BM98a] A. Blum, T. Mitchell, Combining labeled and unlabeled data with co-training, in: Proc. 11th Conf. Computational Learning Theory (COLT'98), Madison, WI, USA, 1998, pp. 92–100.
- [BM98b] Paul S. Bradley, Olvi L. Mangasarian, Feature selection via concave minimization and support vector machines, in: Proc. 1998 Int. Conf. Machine Learning (ICML'98), Madison, WI, USA, July 1998, pp. 82–90.
- [BMAD06] Z.A. Bakar, R. Mohamad, A. Ahmad, M.M. Deris, A comparative study for outlier detection techniques in data mining, in: Proc. 2006 IEEE Conf. Cybernetics and Intelligent Systems, Bangkok, Thailand, 2006, pp. 1–6.

- [BMS97] S. Brin, R. Motwani, C. Silverstein, Beyond market basket: generalizing association rules to correlations, in: Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97), Tucson, AZ, USA, May 1997, pp. 265–276.
- [BMUT97] S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket analysis, in: Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97), Tucson, AZ, USA, May 1997, pp. 255–264.
- [BN92] W.L. Buntine, T. Niblett, A further comparison of splitting rules for decision-tree induction, *Machine Learning* 8 (1992) 75–85.
- [BP92] J.C. Bezdek, S.K. Pal, *Fuzzy Models for Pattern Recognition: Methods That Search for Structures in Data*, IEEE Press, 1992.
- [BPT97] E. Baralis, S. Paraboschi, E. Teniente, Materialized view selection in a multidimensional database, in: Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97), Athens, Greece, Aug. 1997, pp. 98–112.
- [BPW88] E.R. Bareiss, B.W. Porter, C.C. Weir, Protos: an exemplar-based learning apprentice, *International Journal of Man-Machine Studies* 29 (1988) 549–561.
- [BR99] K. Beyer, R. Ramakrishnan, Bottom-up computation of sparse and iceberg cubes, in: Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99), Philadelphia, PA, USA, June 1999, pp. 359–370.
- [Bre96] L. Breiman, Bagging predictors, *Machine Learning* 24 (1996) 123–140.
- [Bre01] L. Breiman, Random forests, *Machine Learning* 45 (2001) 5–32.
- [BRS<sup>+</sup>19] Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, Yejin Choi, Comet: commonsense transformers for automatic knowledge graph construction, *arXiv preprint, arXiv:1906.05317*, 2019.
- [BS97] D. Barbara, M. Sullivan, Quasi-cubes: exploiting approximation in multidimensional databases, *SIGMOD Record* 26 (1997) 12–17.
- [BS03] S.D. Bay, M. Schwabacher, Mining distance-based outliers in near linear time with randomization and a simple pruning rule, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug 2003, pp. 29–38.
- [BSM09] Daniela Brauckhoff, Kave Salamatian, Martin May, Applying pca for traffic anomaly detection: problems and solutions, in: Proc. 2009 Int. Conf. Computer Communications (INFOCOM'09), Rio de Janeiro, Brazil, Apr. 2009, pp. 2866–2870.
- [BSM17] David Berthelot, Thomas Schumm, Luke Metz, Began: boundary equilibrium generative adversarial networks, *arXiv preprint, arXiv:1703.10717*, 2017.
- [BSZG18] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, Stephan Günnemann, Netgan: generating graphs via random walks, in: Proc. 2018 Int. Conf. Machine Learning (ICML'18), Stockholm, Sweden, July 2018, pp. 610–619.
- [BT99] D.P. Ballou, G.K. Tayi, Enhancing data quality in data warehouse environments, *Communications of the ACM* 42 (1999) 73–78.
- [BU95] C.E. Brodley, P.E. Utgoff, Multivariate decision trees, *Machine Learning* 19 (1995) 45–77.
- [BUGD<sup>+</sup>13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, Oksana Yakhnenko, Translating embeddings for modeling multi-relational data, in: Proc. 2013 Conf. Neural Information Processing Systems (NIP'13), Lake Tahoe, NV, USA, Dec. 2013, pp. 1–9.
- [Bun94] W.L. Buntine, Operations for learning with graphical models, *Journal of Artificial Intelligence Research* 2 (1994) 159–225.
- [Bur98] C.J.C. Burges, A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery* 2 (1998) 121–168.
- [BW00] D. Barbará, X. Wu, Using loglinear models to compress datacube, in: Proc. 2000 Int. Conf. Web-Age Information Management (WAIM'2000), Shanghai, China, June 2000, pp. 311–322.

- [BWJ01] D. Barbara, N. Wu, S. Jajodia, Detecting novel network intrusion using bayesian estimators, in: Proc. 2001 SIAM Int. Conf. Data Mining (SDM'01), Chicago, IL, USA, April 2001.
- [BXZ<sup>+</sup>17] Inci M. Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K. Jain, Jiayu Zhou, Patient subtyping via time-aware lstm networks, in: Proc. 2017 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'17), Halifax, NS, Canada, Aug. 2017, pp. 65–74.
- [BZSL13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, Yann LeCun, Spectral networks and locally connected networks on graphs, arXiv preprint, arXiv:1312.6203, 2013.
- [BÉBH15] Laure Berti-Équille, Javier Borge-Holthoefer, Veracity of Data: From Truth Discovery Computation Algorithms to Models of Misinformation Dynamics, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2015.
- [Cat91] J. Catlett, Megainduction: Machine Learning on Very large Databases, Ph.D. Thesis, University of Sydney, 1991.
- [CB21] George Casella, Roger L. Berger, Statistical Inference, Cengage Learning, 2021.
- [CBJD18] Mathilde Caron, Piotr Bojanowski, Armand Joulin, Matthijs Douze, Deep clustering for unsupervised learning of visual features, in: Proc. 2018 European Conf. Computer Vision (ECCV'18), Munich, Germany, Sep. 2018, pp. 132–149.
- [CBK09] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Computing Surveys 41 (2009) 1–58.
- [CBM<sup>+</sup>17] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, Jimeng Sun, Generating multi-label discrete patient records using generative adversarial networks, arXiv preprint, arXiv:1703.06490, 2017.
- [CBŠA<sup>+</sup>20] Zlatan Car, Sandi Baressi Šegota, Nikola Anđelić, Ivan Lorencin, Vedran Mrzljak, Modeling the spread of Covid-19 infection using a multilayer perceptron, in: Computational and Mathematical Methods in Medicine, 2020, 2020.
- [CC00] Y. Cheng, G. Church, Biclustering of expression data, in: Proc. 2000 Int. Conf. Intelligent Systems for Molecular Biology (ISMB'00), La Jolla, CA, USA, Aug. 2000, pp. 93–103.
- [CC19] Raghavendra Chalapathy, Sanjay Chawla, Deep learning for anomaly detection: a survey, arXiv preprint, arXiv:1901.03407, 2019.
- [CCLR05] B.-C. Chen, L. Chen, Y. Lin, R. Ramakrishnan, Prediction cubes, in: Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05), Trondheim, Norway, Aug. 2005, pp. 982–993.
- [CCM03] David Cohn, Rich Caruana, Andrew McCallum, Semi-supervised clustering with user feedback, Technical report, 2003.
- [CCS93] E.F. Codd, S.B. Codd, C.T. Salley, Beyond decision support, Computer World 27 (July 1993).
- [CD97] S. Chaudhuri, U. Dayal, An overview of data warehousing and OLAP technology, SIGMOD Record 26 (1997) 65–74.
- [CDH<sup>+</sup>02] Y. Chen, G. Dong, J. Han, B.W. Wah, J. Wang, Multi-dimensional regression analysis of time-series data streams, in: Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), Hong Kong, China, Aug. 2002, pp. 323–334.
- [CDH<sup>+</sup>06] Y. Chen, G. Dong, J. Han, J. Pei, B.W. Wah, J. Wang, Regression cubes with lossless compression and aggregation, IEEE Transactions on Knowledge and Data Engineering 18 (2006) 1585–1599.
- [CG15] John P. Cunningham, Zoubin Ghahramani, Linear dimensionality reduction: survey, insights, and generalizations, Journal of Machine Learning Research 16 (1) (January 2015) 2859–2900.
- [CG16] Tianqi Chen, Carlos Guestrin, Xgboost: a scalable tree boosting system, in: Proc. 2016 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'16), San Francisco, CA, USA, Aug. 2016, pp. 785–794.
- [CGC94] A. Chaturvedi, P. Green, J. Carroll, K-means, k-medians and k-modes: special cases of partitioning multiway data, in: The Classification Society of North America (CSNA) Meeting Presentation, Houston, TX, USA, 1994.

- [CGC01] A. Chaturvedi, P. Green, J. Carroll, K-modes clustering, *Journal of Classification* 18 (2001) 35–55.
- [CH67] T. Cover, P. Hart, Nearest neighbor pattern classification, *IEEE Transactions on Information Theory* 13 (1967) 21–27.
- [CH92] G. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, *Machine Learning* 9 (1992) 309–347.
- [CH15a] Erik Cambria, Amir Hussain, *Sentic Computing: A Common-Sense-Based Framework for Concept-Level Sentiment Analysis*, 1st edition, Springer Publishing Company, Incorporated, 2015.
- [CH15b] Renato Cordeiro De Amorim, Christian Hennig, Recovering the number of clusters in data sets with noise features using feature rescaling factors, *Information Sciences* 324 (December 2015) 126–145.
- [Cha03] S. Chakrabarti, *Mining the Web: Discovering Knowledge from Hypertext Data*, Morgan Kaufmann, 2003.
- [Chi02] David Maxwell Chickering, Optimal structure identification with greedy search, *Journal of Machine Learning Research* 3 (Nov) (2002) 507–554.
- [CHN<sup>+</sup>96] D.W. Cheung, J. Han, V. Ng, A. Fu, Y. Fu, A fast distributed algorithm for mining association rules, in: *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems (PDIS'96)*, Miami Beach, FL, USA, Dec. 1996, pp. 31–44.
- [CHNW96] D.W. Cheung, J. Han, V. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, in: *Proc. 1996 Int. Conf. Data Engineering (ICDE'96)*, New Orleans, LA, USA, Feb. 1996, pp. 106–114.
- [Chr06] Ronald Christensen, *Log-Linear Models and Logistic Regression*, Springer Science & Business Media, 2006.
- [CHY96] M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from a database perspective, *IEEE Transactions on Knowledge and Data Engineering* 8 (1996) 866–883.
- [CK98] M. Carey, D. Kossman, Reducing the braking distance of an SQL query engine, in: *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, New York, NY, USA, Aug. 1998, pp. 158–169.
- [CKP09] Toon Calders, Faisal Kamiran, Mykola Pechenizkiy, Building classifiers with independency constraints, in: *Proc. 2009 Int. Conf. Data Mining (ICDM'09)*, Miami, FL, USA, Dec. 2009, pp. 13–18.
- [Cle93] W. Cleveland, *Visualizing Data*, Hobart Press, 1993.
- [CLL<sup>+</sup>19] Xiangning Chen, Qingwei Lin, Chuan Luo, Xudong Li, Hongyu Zhang, Yong Xu, Yingnong Dang, Kaixin Sui, Xu Zhang, Bo Qiao, et al., Neural feature search: a neural architecture for automated feature engineering, in: *Proc. 2019 Int. Conf. Data Mining (ICDM'19)*, Beijing, China, Nov. 2019, pp. 71–80.
- [CLLM20] Kevin Clark, Minh-Thang Luong, Quoc V. Le, Christopher D. Manning, ELECTRA: pre-training text encoders as discriminators rather than generators, in: *Proc. 2020 Int. Conf. Learning Representations (ICLR'20)*, Apr. 2020.
- [CIZ06] O. Chapelle, B. Schölkopf, A. Zien, *Semi-Supervised Learning*, MIT Press, 2006.
- [CM94] S.P. Curram, J. Mingers, Neural networks, decision tree induction and discriminant analysis: an empirical comparison, *Journal of the Operational Research Society* 45 (1994) 440–450.
- [CMA94] Jerome T. Connor, R. Douglas Martin, Les E. Atlas, Recurrent neural networks and robust time series prediction, *IEEE Transactions on Neural Networks* 5 (2) (1994) 240–254.
- [CMC05] H. Cao, N. Mamoulis, D.W. Cheung, Mining frequent spatio-temporal sequential patterns, in: *Proc. 2005 Int. Conf. Data Mining (ICDM'05)*, Houston, TX, USA, Nov. 2005, pp. 82–89.
- [CMC18] Raghavendra Chalapathy, Aditya Krishna Menon, Sanjay Chawla, Anomaly detection using one-class neural networks, *arXiv preprint*, arXiv:1802.06360, 2018.
- [CMX18] Jie Chen, Tengfei Ma, Cao Xiao, Fastgcn: fast learning with graph convolutional networks via importance sampling, *arXiv preprint*, arXiv:1801.10247, 2018.

- [CMZS15] Ricardo J.G.B. Campello, Davoud Moulavi, Arthur Zimek, Jörg Sander, Hierarchical density estimates for data clustering, visualization, and outlier detection, *ACM Transactions on Knowledge Discovery from Data* 10 (1) (July 2015).
- [CN89] P. Clark, T. Niblett, The CN2 induction algorithm, *Machine Learning* 3 (1989) 261–283.
- [CNHD11] Xiao Cai, Feiping Nie, Heng Huang, Chris Ding, Multi-class  $l_2$ , 1-norm support vector machine, in: *Proc. 2011 Int. Conf. Data Mining (ICDM'11)*, Vancouver, BC, Canada, Dec. 2011, pp. 91–100.
- [Coh95] W. Cohen, Fast effective rule induction, in: *Proc. 1995 Int. Conf. Machine Learning (ICML'95)*, Tahoe City, CA, USA, July 1995, pp. 115–123.
- [Coo90] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence* 42 (1990) 393–405.
- [Cov99] Thomas M. Cover, *Elements of Information Theory*, John Wiley & Sons, 1999.
- [CPC<sup>+</sup>18] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, Yan Liu, Recurrent neural networks for multivariate time series with missing values, *Scientific Reports* 8 (1) (2018) 1–12.
- [CR95] Y. Chauvin, D. Rumelhart, *Backpropagation: Theory, Architectures, and Applications*, Lawrence Erlbaum, 1995.
- [Cra89] S.L. Crawford, Extensions to the CART algorithm, *International Journal of Man-Machine Studies* 31 (Aug. 1989) 197–217.
- [CRST06] B.-C. Chen, R. Ramakrishnan, J.W. Shavlik, P. Tamma, Bellwether analysis: predicting global aggregates from local regions, in: *Proc. 2006 Int. Conf. Very Large Data Bases (VLDB'06)*, Seoul, Korea, Sept. 2006, pp. 655–666.
- [CS93a] P.K. Chan, S.J. Stolfo, Experiments on multistrategy learning by metalearning, in: *Proc. 1993 Int. Conf. Information and Knowledge Management (CIKM'93)*, Washington, DC, USA, Nov. 1993, pp. 314–323.
- [CS93b] P.K. Chan, S.J. Stolfo, Toward multi-strategy parallel & distributed learning in sequence analysis, in: *Proc. 1993 Int. Conf. Intelligent Systems for Molecular Biology (ISMB'93)*, Bethesda, MD, USA, July 1993, pp. 65–73.
- [CS14] Girish Chandrashekar, Ferat Sahin, A survey on feature selection methods, *Computers & Electrical Engineering* 40 (1) (2014) 16–28.
- [CST00] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge Univ. Press, 2000.
- [CSZ09] Olivier Chapelle, Bernhard Scholkopf, Alexander Zien, Semi-supervised learning (Chapelle, O. et al., Eds.; 2006) [Book reviews], *IEEE Transactions on Neural Networks* 20 (3) (2009) 542.
- [CT65] James W. Cooley, John W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation* 19 (90) (1965) 297–301.
- [CTF<sup>+</sup>15] Yongjie Cai, Hanghang Tong, Wei Fan, Ping Ji, Qing He, Facets: fast comprehensive mining of coevolving high-order time series, in: *Proc. 2015 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'15)*, Sydney, NSW, Australia, Aug. 2015, pp. 79–88.
- [CTFJ15] Yongjie Cai, Hanghang Tong, Wei Fan, Ping Ji, Fast mining of a network of coevolving time series, in: *Proc. 2015 SIAM Int. Conf. Data Mining (SDM'15)*, Vancouver, BC, Canada, May. 2015, pp. 298–306.
- [CTTX05] G. Cong, K.-Lee Tan, A.K.H. Tung, X. Xu, Mining top-k covering rule groups for gene expression data, in: *Proc. 2005 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'05)*, Baltimore, MD, USA, June 2005, pp. 670–681.
- [CTY19] Zhen Chen, Hanghang Tong, Lei Ying, Inferring full diffusion history from partial timestamps, *IEEE Transactions on Knowledge and Data Engineering* 32 (7) (2019) 1378–1392.
- [CUH15] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), *arXiv preprint, arXiv:1511.07289*, 2015.



- [CVMG<sup>+</sup>14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP'14), Doha, Qatar, Oct. 2014, pp. 1724–1734.
- [CWM<sup>+</sup>18] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, Hod Lipson, Autostacker: a compositional evolutionary learning system, in: Proc. 2018 Genetic and Evolutionary Computation Conf. (GECCO'18), Kyoto, Japan, July 2018, pp. 402–409.
- [CWV<sup>+</sup>17] Flavio P. Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, Kush R. Varshney, Optimized pre-processing for discrimination prevention, in: Proc. 2017 Conf. Neural Information Processing Systems (NIP'17), Long Beach, CA, USA, Dec. 2017, pp. 3995–4004.
- [CWZ19] Yu Chen, Lingfei Wu, Mohammed J. Zaki, Bidirectional attentive memory networks for question answering over knowledge bases, arXiv preprint, arXiv:1903.02188, 2019.
- [CYHH07] H. Cheng, X. Yan, J. Han, C.-W. Hsu, Discriminative frequent pattern analysis for effective classification, in: Proc. 2007 Int. Conf. Data Engineering (ICDE'07), Istanbul, Turkey, April 2007, pp. 716–725.
- [CYHY08] H. Cheng, X. Yan, J. Han, P.S. Yu, Direct discriminative pattern mining for effective classification, in: Proc. 2008 Int. Conf. Data Engineering (ICDE'08), Cancun, Mexico, April 2008.
- [CZM<sup>+</sup>18] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, Quoc V. Le, Autoaugment: learning augmentation policies from data, CoRR, arXiv:1805.09501 [abs], 2018.
- [Dar10] A. Darwiche, Bayesian networks, *Communications of the ACM* 53 (2010) 80–90.
- [Das91] B.V. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, 1991.
- [Dau92] I. Daubechies, Ten Lectures on Wavelets, Capital City Press, 1992.
- [DB95] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, *Journal of Artificial Intelligence Research* 2 (1995) 263–286.
- [DBK<sup>+</sup>97] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, V.N. Vapnik, Support vector regression machines, in: M. Mozer, M. Jordan, T. Petsche (Eds.), *Advances in Neural Information Processing Systems*, Vol. 9, MIT Press, 1997, pp. 155–161.
- [DBL<sup>+</sup>15] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith, Transition-based dependency parsing with stack long short-term memory, in: Proc. 2015 Association Computational Linguistics (ACL'15), Beijing, China, July 2015, pp. 334–343.
- [DBV16] Michaël Defferrard, Xavier Bresson, Pierre Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Proc. 2016 Conf. Neural Information Processing Systems (NIPS'16), Barcelona, Spain, Dec. 2016, pp. 3844–3852.
- [DCK18] Nicola De Cao, Thomas Kipf, Molgan: an implicit generative model for small molecular graphs, arXiv preprint, arXiv:1805.11973, 2018.
- [DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Bert: pre-training of deep bidirectional transformers for language understanding, arXiv preprint, arXiv:1810.04805, 2018.
- [DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proc. 2019 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'19), Minneapolis, MN, USA, June 2019, pp. 4171–4186.
- [DCS17] Yuxiao Dong, Nitesh V. Chawla, Ananthram Swami, metapath2vec: scalable representation learning for heterogeneous networks, in: Proc. 2017 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'17), Halifax, NS, Canada, Aug. 2017, pp. 135–144.
- [DDS<sup>+</sup>09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei, Imagenet: a large-scale hierarchical image database, in: Proc. 2009 Conf. Computer Vision and Pattern Recognition (CVPR'09), Miami, FL, USA, Sep. 2009, pp. 248–255.

- [DE84] W.H.E. Day, H. Edelsbrunner, Efficient algorithms for agglomerative hierarchical clustering methods, *Journal of Classification* 1 (1984) 7–24.
- [DGK04] Inderjit S. Dhillon, Yuqiang Guan, Brian Kulis, Kernel k-means: spectral clustering and normalized cuts, in: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04*, New York, NY, USA, Association for Computing Machinery, 2004, pp. 551–556.
- [DH73] W.E. Donath, A.J. Hoffman, Lower bounds for the partitioning of graphs, *IBM Journal of Research and Development* 17 (1973) 420–425.
- [DHL<sup>+</sup>01] G. Dong, J. Han, J. Lam, J. Pei, K. Wang, Mining multi-dimensional constrained gradients in data cubes, in: *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, Rome, Italy, Sept. 2001, pp. 321–330.
- [DHL<sup>+</sup>04] G. Dong, J. Han, J. Lam, J. Pei, K. Wang, W. Zou, Mining constrained gradients in multi-dimensional databases, *IEEE Transactions on Knowledge and Data Engineering* 16 (2004) 922–938.
- [DHS01] R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, 2nd ed., John Wiley & Sons, 2001.
- [DHS11] John Duchi, Elad Hazan, Yoram Singer, Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research* 12 (7) (2011).
- [Die02] Thomas G. Dietterich, Machine learning for sequential data: a review, in: *Joint IAPR Int. Workshops Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, Springer, 2002, pp. 15–30.
- [DJ03] T. Dasu, T. Johnson, *Exploratory Data Mining and Data Cleaning*, John Wiley & Sons, 2003.
- [DJMS02] T. Dasu, T. Johnson, S. Muthukrishnan, V. Shkapenyuk, Mining database structure; or how to build a data quality browser, in: *Proc. 2002 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'02)*, Madison, WI, USA, June 2002, pp. 240–251.
- [DKD<sup>+</sup>18] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, Le Song, Learning steady-states of iterative algorithms over graphs, in: *Proc. 2018 Int. Conf. Machine Learning (ICML'18)*, Stockholm, Sweden, July 2018, pp. 1106–1114.
- [DL97] M. Dash, H. Liu, Feature selection methods for classification, *Intelligent Data Analysis* 1 (1997) 131–156.
- [DL99] G. Dong, J. Li, Efficient mining of emerging patterns: discovering trends and differences, in: *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, USA, Aug. 1999, pp. 43–52.
- [DL12] Luc Devroye, Gábor Lugosi, *Combinatorial Methods in Density Estimation*, Springer Science & Business Media, 2012.
- [DLH19] Mengnan Du, Ninghao Liu, Xia Hu, Techniques for interpretable machine learning, *Communications of the ACM* 63 (1) (2019) 68–77.
- [DLR77] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society, Series B* 39 (1977) 1–38.
- [DLT<sup>+</sup>18] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, Le Song, Adversarial attack on graph structured data, *arXiv preprint, arXiv:1806.02371*, 2018.
- [DLY97] M. Dash, H. Liu, J. Yao, Dimensionality reduction of unsupervised data, in: *Proc. 1997 IEEE Int. Conf. Tools with AI (ICTAI'97)*, IEEE Computer Society, 1997, pp. 532–539.
- [DM77] John E. Dennis Jr, Jorge J. Moré, Quasi-Newton methods, motivation and theory, *SIAM Review* 19 (1) (1977) 46–89.
- [DM02] D. Dasgupta, N.S. Majumdar, Anomaly detection in multidimensional data using negative selection algorithm, in: *Proc. 2002 Congress on Evolutionary Computation (CEC'02)*, Washington DC, USA, 2002, pp. 1039–1044.

- [DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, Adam Smith, Calibrating noise to sensitivity in private data analysis, in: Proc. 2006 Theory of Cryptography Conf. (TCC'06), New York, NY, USA, March 2006, pp. 265–284.
- [DMS00] Sergey N. Dorogovtsev, José Fernando, F. Mendes, Alexander N. Samukhin, Structure of growing networks with preferential linking, *Physical Review Letters* 85 (21) (2000) 4633.
- [DMSR18] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, Sebastian Riedel, Convolutional 2d knowledge graph embeddings, in: Proc. 2018 AAAI Conf. Artificial Intelligence (AAAI'18), New Orleans, LA, USA, 2018, pp. 1811–1818.
- [DNR<sup>+</sup>97] P. Deshpande, J. Naughton, K. Ramasamy, A. Shukla, K. Tufte, Y. Zhao, Cubing algorithms, storage estimation, and storage and processing alternatives for OLAP, *Buletin of the Technical Committee on Data Engineering* 20 (1997) 3–11.
- [Doe16] Carl Doersch, Tutorial on variational autoencoders, arXiv preprint, arXiv:1606.05908, 2016.
- [Dom94] P. Domingos, The RISE system: conquering without separating, in: Proc. 1994 IEEE Int. Conf. Tools with Artificial Intelligence (TAI'94), New Orleans, LA, USA, 1994, pp. 704–707.
- [Dom99] P. Domingos, The role of Occam's razor in knowledge discovery, *Data Mining and Knowledge Discovery* 3 (1999) 409–425.
- [Doz16] Timothy Dozat, Incorporating Nesterov Momentum into Adam, 2016.
- [DP96] P. Domingos, M. Pazzani, Beyond independence: conditions for the optimality of the simple Bayesian classifier, in: Proc. 1996 Int. Conf. Machine Learning (ICML'96), Bari, Italy, July 1996, pp. 105–112.
- [DP97] J. Devore, R. Peck, *Statistics: The Exploration and Analysis of Data*, Duxbury Press, 1997.
- [DR99] D. Donjerkovic, R. Ramakrishnan, Probabilistic optimization of top N queries, in: Proc. 1999 Int. Conf. Very Large Data Bases (VLDB'99), Edinburgh, UK, Sept. 1999, pp. 411–422.
- [DS98] Norman R. Draper, Harry Smith, *Applied Regression Analysis*, Vol. 326, John Wiley & Sons, 1998.
- [DS15] Xin Luna Dong, Divesh Srivastava, *Big Data Integration. Synthesis Lectures on Data Management*, Morgan & Claypool Publishers, 2015.
- [DSH13] George E. Dahl, Tara N. Sainath, Geoffrey E. Hinton, Improving deep neural networks for lvsr using rectified linear units and dropout, in: Proc. 2013 Int. Conf. Acoustics, Speech and Signal Processing (ICASSP'13), Vancouver, BC, Canada, 2013, pp. 8609–8613.
- [DT93] V. Dhar, A. Tuzhilin, Abstract-driven pattern discovery in databases, *IEEE Transactions on Knowledge and Data Engineering* 5 (1993) 926–938.
- [DT19] Boxin Du, Hanghang Tong, Mrmine: multi-resolution multi-network embedding, in: Proc. 2019 ACM Int. Conf. Information and Knowledge Management (CIKM'19), Beijing, China, Nov. 2019, pp. 479–488.
- [DTZ18] Ming Ding, Jie Tang, Jie Zhang, Semi-supervised learning on graphs with generative adversarial nets, in: Proc. 2018 Int. Conf. Information and Knowledge Management (CIKM'18), Torino, Italy, Oct. 2018, pp. 913–922.
- [DWA10] Jordi Duch, Joshua S. Waitzman, Luís A. Nunes Amaral, Quantifying the performance of individual players in a team activity, *PLoS ONE* 5 (6) (06 2010) 1–7.
- [DWB06] I. Davidson, K.L. Wagstaff, S. Basu, Measuring constraint-set utility for partitional clustering algorithms, in: Proc. 2006 European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD'06), Berlin, Germany, Sept. 2006, pp. 115–126.
- [DYXY07] W. Dai, Q. Yang, G. Xue, Y. Yu, Boosting for transfer learning, in: Proc. 2007 Int. Conf. Machine Learning (ICML'07), Corvallis, OR, USA, June 2007, pp. 193–200.
- [EAP<sup>+</sup>02] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, S. Stolfo, A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data, in: Proc. 2002 Int. Conf. of Data Mining for Security Applications, 2002.

- [ECBV10] Dumitru Erhan, Aaron Courville, Yoshua Bengio, Pascal Vincent, Why does unsupervised pre-training help deep learning?, in: Proc. 2010 Int. Conf. Artificial Intelligence and Statistics (AISTATS'10), Chia Laguna Resort, Sardinia, Italy, May 2010, pp. 201–208.
- [Ega75] J.P. Egan, Signal Detection Theory and ROC Analysis, Academic Press, 1975.
- [EKSW<sup>+</sup>14] Ahmed El-Kishky, Yanglei Song, Chi Wang, Clare R. Voss, Jiawei Han, Scalable topical phrase mining from text corpora, Proceedings of the VLDB Endowment 8 (3) (2014).
- [EKSX96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases, in: Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96), Portland, OR, USA, Aug. 1996, pp. 226–231.
- [EKX95] M. Ester, H.-P. Kriegel, X. Xu, Knowledge discovery in large spatial databases: focusing techniques for efficient class identification, in: Proc. 1995 Int. Symp. Large Spatial Databases (SSD'95), Portland, ME, USA, Aug. 1995, pp. 67–82.
- [Elk97] C. Elkan, Boosting and naive Bayesian learning, Technical Report CS97-557, Department of Computer Science and Engineering, Univ. Calif. at San Diego, Sept. 1997.
- [Elk01] C. Elkan, The foundations of cost-sensitive learning, in: Proc. 2001 Int. Joint Conf. Artificial Intelligence (IJCAI'01), Seattle, WA, USA, Aug. 2001, pp. 973–978.
- [Elm90] Jeffrey L. Elman, Finding structure in time, Cognitive Science 14 (2) (1990) 179–211.
- [ER60] Paul Erdős, Alfréd Rényi, On the evolution of random graphs, Publications of the Mathematical Institute of the Hungarian Academy of Sciences 5 (1) (1960) 17–60.
- [Esk00] E. Eskin, Anomaly detection over noisy data using learned probability distributions, in: Proc. 17th Int. Conf. Machine Learning (ICML'00), 2000.
- [ET93] B. Efron, R. Tibshirani, An Introduction to the Bootstrap, Chapman & Hall, 1993.
- [Fan15] Huang Fang, Managing data lakes in big data era: what's a data lake and why has it become popular in data management ecosystem, in: Proc. 2015 Int. Conf. Cyber Technology in Automation, Control, and Intelligent Systems (CYBER'15), Shenyang, China, 2015, pp. 820–824.
- [FBF77] J.H. Friedman, J.L. Bentley, R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Transactions on Mathematical Software 3 (1977) 209–226.
- [FCMR08] Maurizio Filippone, Francesco Camastra, Francesco Masulli, Stefano Rovetta, A survey of kernel and spectral methods for clustering, Pattern Recognition 41 (1) (2008) 176–190.
- [FDR<sup>+</sup>14] Mehrdad Farajtabar, Nan Du, Manuel Gomez Rodriguez, Isabel Valera, Hongyuan Zha, Le Song, Shaping social activity by incentivizing users, in: Proc. 2014 Conf. Neural Information Processing Systems (NIP'14), Montreal, QC, Canada, Dec. 2014, pp. 2474–2482.
- [FFW<sup>+</sup>19] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, Pierre-Alain Muller, Deep learning for time series classification: a review, Data Mining and Knowledge Discovery 33 (4) (2019) 917–963.
- [FG02] M. Fishelson, D. Geiger, Exact genetic linkage computations for general pedigrees, Disinformatio 18 (2002) 189–198.
- [FGK<sup>+</sup>05] R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, A. Tomkins, Multi-structural databases, in: Proc. 2005 ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS'05), Baltimore, MD, USA, June 2005, pp. 184–195.
- [FH51] E. Fix, J.L. Hodges Jr., Discriminatory analysis, non-parametric discrimination: consistency properties, Technical Report 21-49-004(4), USAF School of Aviation Medicine, Randolph Field, Texas, 1951.
- [FH87] K. Fukunaga, D. Hummels, Bayes error estimation using Parzen and k-nn procedure, IEEE Transactions on Pattern Analysis and Machine Learning 9 (1987) 634–643.
- [FH95] Y. Fu, J. Han, Meta-rule-guided mining of association rules in relational databases, in: Proc. 1995 Int. Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95), Singapore, Dec. 1995, pp. 39–46.

- [FHT08] Jerome Friedman, Trevor Hastie, Robert Tibshirani, Sparse inverse covariance estimation with the graphical lasso, *Biostatistics* 9 (3) (2008) 432–441.
- [FHZ<sup>+</sup>18] Jun Feng, Minlie Huang, Li Zhao, Yang Yang, Xiaoyan Zhu, Reinforcement learning for relation classification from noisy data, in: *Proc. 2018 Nat. Conf. Artificial Intelligence (AAAI'18)*, New Orleans, LA, USA, Feb. 2018, pp. 5779–5786.
- [FI90] U.M. Fayyad, K.B. Irani, What should be minimized in a decision tree?, in: *Proc. 1990 Nat. Conf. Artificial Intelligence (AAAI'90)*, Boston, MA, USA, Aug. 1990, pp. 749–754.
- [FI92] U.M. Fayyad, K.B. Irani, The attribute selection problem in decision tree generation, in: *Proc. 1992 Nat. Conf. Artificial Intelligence (AAAI'92)*, San Jose, CA, USA, July 1992, pp. 104–110.
- [Fie73] M. Fiedler, Algebraic connectivity of graphs, *Czechoslovak Mathematical Journal* 23 (1973) 298–305.
- [FJ02] Mario A.T. Figueiredo, Anil K. Jain, Unsupervised learning of finite mixture models, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (3) (2002) 381–396.
- [FKE<sup>+</sup>15] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Springenberg, Manuel Blum, Frank Hutter, Efficient and robust automated machine learning, in: *Proc. 2015 Conf. Neural Information Processing Systems (NIPS'15)*, Montreal, QC, Canada, Dec. 2015, pp. 2962–2970.
- [FL90] S. Fahlman, C. Lebiere, The cascade-correlation learning algorithm, Technical Report CMU-CS-90-100, Computer Sciences Department, Carnegie Mellon University, 1990.
- [Fle87] R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, 1987.
- [FMMT96] T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization, in: *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, Montreal, Canada, June 1996, pp. 13–23.
- [Fox97] John Fox, *Applied Regression Analysis, Linear Models, and Related Methods*, Sage Publications, Inc, 1997.
- [FP97] T. Fawcett, F. Provost, Adaptive fraud detection, *Data Mining and Knowledge Discovery* 1 (1997) 291–316.
- [FPP07] D. Freedman, R. Pisani, R. Purves, *Statistics*, 4th ed., W. W. Norton & Co., 2007.
- [FPSSe96] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996.
- [FR02] C. Fraley, A.E. Raftery, Model-based clustering, discriminant analysis, and density estimation, *Journal of the American Statistical Association* 97 (2002) 611–631.
- [Fre09] David A. Freedman, *Statistical Models: Theory and Practice*, Cambridge University Press, 2009.
- [Fri77] J.H. Friedman, A recursive partitioning decision rule for nonparametric classifiers, *IEEE Transactions on Computers* 26 (1977) 404–408.
- [Fri01] J.H. Friedman, Greedy function approximation: a gradient boosting machine, *The Annals of Statistics* 29 (2001) 1189–1232.
- [Fri03] N. Friedman, Pcluster: Probabilistic agglomerative clustering of gene expression profiles, Technical Report 2003-80, Hebrew Univ, 2003.
- [FRZ<sup>+</sup>15] Mehrdad Farajtabar, Manuel Gomez Rodriguez, Mohammad Zamani, Nan Du, Hongyuan Zha, Le Song, Back to the past: source identification in diffusion networks from partially observed cascades, in: *Proc. 2015 Int. Conf. Artificial Intelligence and Statistics (AISTATS'15)*, San Diego, CA, USA, May 2015, pp. 232–240.
- [FS97] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences* 55 (1997) 119–139.
- [FSE11] Anthony Fader, Stephen Soderland, Oren Etzioni, Identifying relations for open information extraction, in: *Proc. 2011 Conf. Empirical Methods in Natural Language Processing (EMNLP'11)*, Edinburgh, UK, July 2011, pp. 1535–1545.

- [FSGM<sup>+</sup>98] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman, Computing iceberg queries efficiently, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 299–310.
- [Fuk75] Kunihiko Fukushima, Cognitron: a self-organizing multilayered neural network, *Biological Cybernetics* 20 (3–4) (1975) 121–136.
- [Fun89] Ken-Ichi Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Networks* 2 (3) (1989) 183–192.
- [FW94] J. Furnkranz, G. Widmer, Incremental reduced error pruning, in: Proc. 1994 Int. Conf. Machine Learning (ICML'94), New Brunswick, NJ, USA, July 1994, pp. 70–77.
- [FYM05] R. Fujimaki, T. Yairi, K. Machida, An approach to spacecraft anomaly detection problem using kernel feature space, in: Proc. 2005 Int. Workshop on Link Discovery (LinkKDD'05), Chicago, IL, USA, 2005, pp. 401–410.
- [G<sup>+</sup>58] Gerald Goertzel, et al., An algorithm for the evaluation of finite trigonometric series, *The American Mathematical Monthly* 65 (1) (1958) 34–35.
- [Gau66] Carl Friedrich Gauss, *Nachlass: Theoria interpolationis methodo nova tractata*, Carl Friedrich Gauss Werke 3 (1866) 265–327.
- [GB10] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proc. 2010 Int. Conf. Artificial Intelligence and Statistics (AISTATS'10), Chia Laguna Resort, Sardinia, Italy, May 2010, pp. 249–256.
- [GB14] Pedram Ghamisi, Jon Atli Benediktsson, Feature selection based on hybridization of genetic algorithm and particle swarm optimization, *IEEE Geoscience and Remote Sensing Letters* 12 (2) (2014) 309–313.
- [GBC16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016.
- [GCB<sup>+</sup>97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals, *Data Mining and Knowledge Discovery* 1 (1997) 29–54.
- [GCB04] Nizar Grira, Michel Crucianu, Nozha Boujemaa, Unsupervised and semi-supervised clustering: a brief survey, in: “A Review of Machine Learning Techniques for Processing Multimedia Content”, Report of the MUSCLE European Network of Excellence (FP6), 2004.
- [GCL<sup>+</sup>20] Ruocheng Guo, Lu Cheng, Jundong Li, P. Richard Hahn, Huan Liu, A survey of learning causality with data: problems and methods, *ACM Computing Surveys* 53 (4) (2020) 1–37.
- [GD91] David E. Goldberg, Kalyanmoy Deb, A comparative analysis of selection schemes used in genetic algorithms, in: *Foundations of Genetic Algorithms*, Vol. 1, Elsevier, 1991, pp. 69–93.
- [GDBFL19] Paula Gordaliza, Eustasio Del Barrio, Gamboa Fabrice, Jean-Michel Loubes, Obtaining fairness using optimal transport theory, in: Proc. 2019 Int. Conf. Machine Learning (ICML'19), Long Beach, CA, USA, June 2019, pp. 2357–2365.
- [GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proc. 2014 Conf. Computer Vision and Pattern Recognition (CVPR'14), Columbus, OH, USA, June 2014, pp. 580–587.
- [GFS<sup>+</sup>01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C.-A. Saita, Declarative data cleaning: language, model, and algorithms, in: Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01), Rome, Italy, Sept. 2001, pp. 371–380.
- [GG92] A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, 1992.
- [GG16] Yarín Gal, Zoubin Ghahramani, Dropout as a bayesian approximation: representing model uncertainty in deep learning, in: Proc. 2016 Int. Conf. Machine Learning (ICML'16), New York City, NY, USA, June 2016, pp. 1050–1059.
- [GGAH13] Manish Gupta, Jing Gao, Charu C. Aggarwal, Jiawei Han, Outlier detection for temporal data: a survey, *IEEE Transactions on Knowledge and Data Engineering* 26 (9) (2013) 2250–2267.

- [GGN<sup>+</sup>14] Liang Ge, Jing Gao, Hung Ngo, Kang Li, Aidong Zhang, On handling negative transfer and imbalanced distributions in multiple source transfer learning, *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7 (4) (2014) 254–271.
- [GGR99] V. Ganti, J.E. Gehrke, R. Ramakrishnan, CACTUS—clustering categorical data using summaries, in: *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, USA, 1999, pp. 73–83.
- [GGRL99] J. Gehrke, V. Ganti, R. Ramakrishnan, W.-Y. Loh, BOAT—optimistic decision tree construction, in: *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, Philadelphia, PA, USA, June 1999, pp. 169–180.
- [GHL06] H. Gonzalez, J. Han, X. Li, Flowcube: constructing RFID flowcubes for multi-dimensional analysis of commodity flows, in: *Proc. 2006 Int. Conf. Very Large Data Bases (VLDB'06)*, Seoul, Korea, Sept. 2006, pp. 834–845.
- [GHLK06] H. Gonzalez, J. Han, X. Li, D. Klabjan, Warehousing and analysis of massive RFID data sets, in: *Proc. 2006 Int. Conf. Data Engineering (ICDE'06)*, Atlanta, GA, USA, April 2006, p. 83.
- [Gir15] Ross Girshick, Fast r-cnn, in: *Proc. 2015 Int. Conf. Computer Vision (ICCV'15)*, Santiago, Chile, Dec. 2015, pp. 1440–1448.
- [GJ19] Hongyang Gao, Shuiwang Ji, Graph u-nets, arXiv preprint, arXiv:1905.05178, 2019.
- [GK13] Raymond Greenlaw, Sanpawat Kantabutra, Survey of clustering: algorithms and applications, *International Journal of Information Retrieval Research* 3 (2) (April 2013) 1–29.
- [GKR98] D. Gibson, J.M. Kleinberg, P. Raghavan, Clustering categorical data: an approach based on dynamical systems, in: *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, New York, NY, USA, Aug. 1998, pp. 311–323.
- [GL16] Aditya Grover, Jure Leskovec, node2vec: scalable feature learning for networks, in: *Proc. 2016 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'16)*, San Francisco, CA, USA, Aug. 2016, pp. 855–864.
- [GLC<sup>+</sup>18] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, Jun Wang, Long text generation via adversarial training with leaked information, in: *Proc. 2018 Nat. Conf. Artificial Intelligence (AAAI'18)*, New Orleans, LA, USA, Feb. 2018, pp. 5141–5148.
- [GLH12] Quanquan Gu, Zhenhui Li, Jiawei Han, Generalized Fisher score for feature selection, arXiv preprint, arXiv:1202.3725, 2012.
- [GLH15] Salvador García, Julián Luengo, Francisco Herrera, *Data Preprocessing in Data Mining*, Springer, 2015.
- [GLW00] G. Grahne, L.V.S. Lakshmanan, X. Wang, Efficient mining of constrained correlated sets, in: *Proc. 2000 Int. Conf. Data Engineering (ICDE'00)*, San Diego, CA, USA, Feb. 2000, pp. 512–521.
- [GLW<sup>+</sup>19] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, Yan Liu, Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting, in: *Proc. 2019 AAAI Conf. Artificial Intelligence (AAAI'19)*, Honolulu, HI, USA, Jan. 2019, pp. 3656–3663.
- [GM99] A. Gupta, I.S. Mumick, *Materialized Views: Techniques, Implementations, and Applications*, MIT Press, 1999.
- [GMH13] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton, Speech recognition with deep recurrent neural networks, in: *Proc. 2013 Int. Conf. Acoustics, Speech and Signal Processing (ICASSP'13)*, Vancouver, BC, Canada, May 2013, pp. 6645–6649.
- [GMMO00] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams, in: *Proc. 2000 Symp. Foundations of Computer Science (FOCS'00)*, Redondo Beach, CA, USA, 2000, pp. 359–366.
- [GMP<sup>+</sup>09] J. Ginsberg, M.H. Mohebbi, R.S. Patel, L. Brammer, M.S. Smolinski, L. Brilliant, Detecting influenza epidemics using search engine query data, *Nature* 457 (Feb. 2009) 1012–1014.
- [GMV96] I. Guyon, N. Matic, V. Vapnik, Discovering informative patterns and data cleaning, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996, pp. 181–203.

- [Gol89] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
- [Goo58] Irving John Good, The interaction algorithm and practical Fourier analysis, *Journal of the Royal Statistical Society, Series B, Methodological* 20 (2) (1958) 361–372.
- [GPAM<sup>+</sup>14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative adversarial nets, in: *Proc. 2014 Conf. Neural Information Processing Systems (NIPS'14)*, Montreal, QC, Canada, Dec. 2014, pp. 2672–2680.
- [GRG98] J. Gehrke, R. Ramakrishnan, V. Ganti, RainForest: a framework for fast decision tree construction of large datasets, in: *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, New York, NY, USA, Aug. 1998, pp. 416–427.
- [GRS98] S. Guha, R. Rastogi, K. Shim, CURE: an efficient clustering algorithm for large databases, in: *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, Seattle, WA, USA, June 1998, pp. 73–84.
- [GRS99] S. Guha, R. Rastogi, K. Shim, ROCK: a robust clustering algorithm for categorical attributes, in: *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, Sydney, Australia, Mar. 1999, pp. 512–521.
- [GRSD<sup>+</sup>16] Manuel Gomez-Rodriguez, Le Song, Nan Du, Hongyuan Zha, Bernhard Schölkopf, Influence estimation and maximization in continuous-time diffusion networks, *ACM Transactions on Information Systems* 34 (2) (2016) 1–33.
- [Gru69] F.E. Grubbs, Procedures for detecting outlying observations in samples, *Technometrics* 11 (1969) 1–21.
- [GS05] Alex Graves, Jürgen Schmidhuber, Framewise phoneme classification with bidirectional lstm and other neural network architectures, *Neural Networks* 18 (5–6) (2005) 602–610.
- [GSPA04] Roger Guimera, Marta Sales-Pardo, Luís A. Nunes Amaral, Modularity from fluctuations in random graphs and complex networks, *Physical Review E* 70 (2) (2004) 025101.
- [GSR<sup>+</sup>17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, George E. Dahl, Neural message passing for quantum chemistry, *arXiv preprint*, arXiv:1704.01212, 2017.
- [GSS14] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, Explaining and harnessing adversarial examples, *arXiv preprint*, arXiv:1412.6572, 2014.
- [Gup97] H. Gupta, Selection of views to materialize in a data warehouse, in: *Proc. 1997 Int. Conf. Database Theory (ICDT'97)*, Delphi, Greece, Jan. 1997, pp. 98–112.
- [GWB<sup>+</sup>21] Xiaotao Gu, Zihan Wang, Zhenyu Bi, Yu Meng, Liyuan Liu, Jiawei Han, Jingbo Shang, UCPhrase: unsupervised context-aware quality phrase tagging, in: *Proc. 2021 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'21)*, 2021, pp. 478–486.
- [GZ03a] B. Goethals, M. Zaki, An introduction to workshop on frequent itemset mining implementations, in: *Proc. 2003 Int. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, FL, USA, Nov. 2003.
- [GZ03b] G. Grahne, J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, in: *Proc. 2003 Int. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, FL, USA, Nov. 2003.
- [GZE19] Aditya Grover, Aaron Zweig, Stefano Ermon, Graphite: iterative generative modeling of graphs, in: *Proc. 2019 Int. Conf. Machine Learning (ICML'19)*, Long Beach, CA, USA, June 2019, pp. 2434–2444.
- [GZN<sup>+</sup>19] Xiaojie Guo, Liang Zhao, Cameron Nowzari, Setareh Rafatirad, Houman Homayoun, Sai Manoj Pudukotai Dinakarrao, Deep multi-attributed graph translation with node-edge co-evolution, in: *Proc. 2019 Int. Conf. Data Mining (ICDM'19)*, Beijing, China, Nov. 2019, pp. 250–259.
- [HA04] V.J. Hodge, J. Austin, A survey of outlier detection methodologies, *Artificial Intelligence Review* 22 (2004) 85–126.
- [HAC<sup>+</sup>99] J.M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, P.J. Haas, Interactive data analysis: the control project, *IEEE Computer* 32 (July 1999) 51–59.



- [Han98] J. Han, Towards on-line analytical mining in large databases, *SIGMOD Record* 27 (1998) 97–107.
- [Har68] P.E. Hart, The condensed nearest neighbor rule, *IEEE Transactions on Information Theory* 14 (1968) 515–516.
- [Har72] J. Hartigan, Direct clustering of a data matrix, *Journal of the American Statistical Association* 67 (1972) 123–129.
- [Har75] J.A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, 1975.
- [Haw80] D.M. Hawkins, *Identification of Outliers*, Chapman and Hall, London, 1980.
- [Hay99] S.S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.
- [Hay08] S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, Saddle River, NJ, USA, 2008.
- [HBV01] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, *Journal of Intelligent Information Systems* 17 (2001) 107–145.
- [HC08] Jingrui He, Jaime G. Carbonell, Nearest-neighbor-based active learning for rare category detection, in: *Proc. 2008 Conf. Neural Information Processing Systems (NIPS'08)*, Vancouver, BC, Canada, Dec. 2008, pp. 633–640.
- [HCN05] Xiaofei He, Deng Cai, Partha Niyogi, Laplacian score for feature selection, in: *Proc. 2005 Conf. Neural Information Processing Systems (NIPS'05)*, Dec. 2005, pp. 507–514.
- [Heb49] Donald Olding Hebb, *The Organization of Behavior: a Neuropsychological Theory*, J. Wiley/Chapman & Hall, 1949.
- [Hec96] D. Heckerman, Bayesian networks for knowledge discovery, in: *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996, pp. 273–305.
- [HF95] J. Han, Y. Fu, Discovery of multiple-level association rules from large databases, in: *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, Zurich, Switzerland, Sept. 1995, pp. 420–431.
- [HFLM<sup>+</sup>18] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, Yoshua Bengio, Learning deep representations by mutual information estimation and maximization, *arXiv preprint*, arXiv:1808.06670, 2018.
- [HFLP01] P.S. Horn, L. Feng, Y. Li, A.J. Pesce, Effect of outliers and nonhealthy individuals on reference interval estimation, *Clinical Chemistry* 47 (2001) 2137–2145.
- [HG05] K.A. Heller, Z. Ghahramani, Bayesian hierarchical clustering, in: *Proc. 22nd Int. Conf. Machine Learning (ICML'05)*, Bonn, Germany, 2005, pp. 297–304.
- [HG07] A. Hinneburg, H.-H. Gabriel, DENCLUE 2.0: fast clustering based on kernel density estimation, in: *Proc. 2007 Int. Conf. Intelligent Data Analysis (IDA'07)*, Ljubljana, Slovenia, 2007, pp. 70–80.
- [HGC95] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, *Machine Learning* 20 (1995) 197–243.
- [HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick, Mask r-cnn, in: *Proc. 2017 Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Oct. 2017, pp. 2961–2969.
- [HGL<sup>+</sup>11] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, Christos Faloutsos, It's who you know: graph mining using recursive structural features, in: *Proc. 2011 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'11)*, San Diego, CA, USA, Aug. 2011, pp. 663–671.
- [HGQ16] Rihan Hai, Sandra Geisler, Christoph Quix, Constance: an intelligent data lake system, in: *Proc. 2016 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'16)*, New York, NY, USA, June 2016, pp. 2097–2100.
- [HGX11] Timothy M. Hospedales, Shaogang Gong, Tao Xiang, Finding rare classes: active learning with generative and discriminative models, *IEEE Transactions on Knowledge and Data Engineering* 25 (2) (2011) 374–386.
- [HH01] R.J. Hilderan, H.J. Hamilton, *Knowledge Discovery and Measures of Interest*, Kluwer Academic, 2001.

- [HHLB11] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proc. 2011 Int. Conf. Learning and Intelligent Optimization (LION'11), Rome, Italy, Jan. 2011, pp. 507–523.
- [HHN<sup>+</sup>19] Ehsan Hajiramezani, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, Xiaoning Qian, Variational graph recurrent neural networks, in: Proc. 2019 Conf. Neural Information Processing Systems (NeurIPS'19), Vancouver, BC, Canada, Dec. 2019, pp. 10701–10711.
- [HHW97] J. Hellerstein, P. Haas, H. Wang, Online aggregation, in: Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97), Tucson, AZ, USA, May 1997, pp. 171–182.
- [Hil11] Jennifer L. Hill, Bayesian nonparametric modeling for causal inference, *Journal of Computational and Graphical Statistics* 20 (1) (2011) 217–240.
- [HIR03] Keisuke Hirano, Guido W. Imbens, Geert Ridder, Efficient estimation of average treatment effects using the estimated propensity score, *Econometrica* 71 (4) (2003) 1161–1189.
- [HK91] P. Hoschka, W. Klösgen, A support system for interpreting statistical data, in: G. Piatetsky-Shapiro, W.J. Flöwley (Eds.), *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991, pp. 325–346.
- [HK98] A. Hinneburg, D.A. Keim, An efficient approach to clustering in large multimedia databases with noise, in: Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98), New York, NY, USA, Aug. 1998, pp. 58–65.
- [HK10] J. Hackman, Nancy Katz, *Group Behavior and Performance*, Vol. 32, 06 2010.
- [HKKR99] F. Höppner, F. Klawonn, R. Kruse, T. Runkler, *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*, Wiley, 1999.
- [HKN<sup>+</sup>16] Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, Goods: organizing Google's datasets, in: Proc. 2016 Int. Conf. Management of Data (SIGMOD'16), New York, NY, USA, June 2016, pp. 795–806.
- [HKP91] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, 1991.
- [HKP11] J. Han, M. Kamber, J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed., Morgan Kaufmann, 2011.
- [HKV19] Frank Hutter, Lars Kotthoff, Joaquin Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*, Springer Nature, 2019.
- [HL18] Yaoyao He, Haiyan Li, Probability density forecasting of wind power using quantile regression neural network and kernel density estimation, *Energy Conversion and Management* 164 (2018) 374–384.
- [HLL83] Paul W. Holland, Kathryn Blackmond Laskey, Samuel Leinhardt, Stochastic blockmodels: first steps, *Social Networks* 5 (2) (1983) 109–137.
- [HLL08] Jingrui He, Yan Liu, Richard Lawrence, Graph-based rare category detection, in: Proc. 2008 Int. Conf. on Data Mining (ICDM'08), Pisa, Italy, Dec. 2008, pp. 833–838.
- [HLLT16] Chen Huang, Yining Li, Chen Change Loy, Xiaoou Tang, Learning deep representation for imbalanced classification, in: Proc. 2016 Conf. Computer Vision and Pattern Recognition (CVPR'16), Las Vegas, NV, USA, June 2016, pp. 5375–5384.
- [HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, Kilian Q. Weinberger, Densely connected convolutional networks, in: Proc. 2017 Conf. Computer Vision and Pattern Recognition (CVPR'17), Honolulu, HI, USA, July 2017, pp. 4700–4708.
- [HMA09] Parisa Haghani, Sebastian Michel, Karl Aberer, Distributed similarity search in high dimensions using locality sensitive hashing, in: Proc. 2009 Int. Conf. Extending Database Technology (EDBT'09), Saint Petersburg, Russia, Mar. 2009, pp. 744–755.
- [HMM86] J. Hong, I. Mozetic, R.S. Michalski, AQ15: Incremental learning of attribute-based descriptions from examples, the method and user's guide, Report ISG 85-5, UIUCDCS-F-86-949, Department of Comp. Science, University of Illinois at Urbana-Champaign, 1986.

- [HMS66] E.B. Hunt, J. Marin, P.T. Stone, *Experiments in Induction*, Academic Press, 1966.
- [HMS01] D.J. Hand, H. Mannila, P. Smyth, *Principles of Data Mining*, MIT Press, 2001.
- [HN90] R. Hecht-Nielsen, *Neurocomputing*, Addison Wesley, 1990.
- [Hol18] Andreas Holzinger, From machine learning to explainable ai, in: Proc. 2018 Symp. Digital Intelligence for Systems and Machines (DISA'18), Kosice, Slovakia, Aug. 2018, pp. 55–66.
- [Hop82] John J. Hopfield, *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, Vol. 79, National Acad Sciences, 1982, pp. 2554–2558.
- [HP07] M. Hua, J. Pei, Cleaning disguised missing data: a heuristic approach, in: Proc. 2007 ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining (KDD'07), San Jose, CA, USA, Aug. 2007.
- [HPDW01] J. Han, J. Pei, G. Dong, K. Wang, Efficient computation of iceberg cubes with complex measures, in: Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01), Santa Barbara, CA, USA, May 2001, pp. 1–12.
- [HPS97] J. Hosking, E. Pednault, M. Sudan, A statistical perspective on data mining, *Future Generations Computer Systems* 13 (1997) 117–134.
- [HPY00] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, USA, May 2000, pp. 1–12.
- [HR02] Geoffrey E. Hinton, Sam T. Roweis, Stochastic neighbor embedding, in: *Advances in Neural Information Processing Systems 15* [Neural Information Processing Systems, NIPS 2002, December 9–14, 2002, Vancouver, BC, Canada], 2002, pp. 833–840.
- [HRU96] V. Harinarayan, A. Rajaraman, J.D. Ullman, Implementing data cubes efficiently, in: Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96), Montreal, Canada, June 1996, pp. 205–216.
- [HS54] Brian Hopkins, J.G. Skellam, A new method for determining the type of distribution of plant individuals, *Annals of Botany* 18 (2) (04 1954) 213–227.
- [HS97] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, *Neural Computation* 9 (8) (1997) 1735–1780.
- [HS00] Richard H.R. Hahnloser, H. Sebastian Seung, Permitted and forbidden sets in symmetric threshold-linear networks, in: Proc. 2000 Conf. Neural Information Processing Systems (NIPS'00), Denver, CO, USA, 2000, pp. 217–223.
- [HSG89] S.A. Harp, T. Samad, A. Guha, Designing application-specific neural networks using the genetic algorithm, in: Proc. 1989 Conf. Neural Information Processing Systems (NIPS'89), Denver, CO, USA, Nov. 1989, pp. 447–454.
- [HSS] Geoffrey Hinton, Nitish Srivastava, Kevin Swersky, *Neural networks for machine learning*, online course material.
- [HSS08] Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Kernel methods in machine learning, *The Annals of Statistics* 36 (3) (2008) 1171–1220.
- [HT98] T. Hastie, R. Tibshirani, Classification by pairwise coupling, *The Annals of Statistics* 26 (1998) 451–471.
- [HTF09] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer Verlag, 2009.
- [HU13] Hannes Heikinheimo, Antti Ukkonen, The crowd-median algorithm, in: Proc. 2013 Conf. Human Computation and Crowdsourcing (HCOMP'13), Palm Springs, CA, USA, Nov. 2013.
- [Hua98] Z. Huang, Extensions to the k-means algorithm for clustering large data sets with categorical values, *Data Mining and Knowledge Discovery* 2 (1998) 283–304.
- [Hub96] B.B. Hubbard, *The World According to Wavelets*, A. K. Peters, 1996.
- [HW62] David H. Hubel, Torsten N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *The Journal of Physiology* 160 (1962) 106.

- [HXD03] Z. He, X. Xu, S. Deng, Discovering cluster-based local outliers, *Pattern Recognition Letters* 24 (June 2003) 1641–1650.
- [HXM<sup>+</sup>20] Jiaxin Huang, Yiqing Xie, Yu Meng, Jiaming Shen, Yunyi Zhang, Jiawei Han, Guiding corpus-based set expansion by auxiliary sets generation and co-expansion, in: *Proc. 2020 the Web Conf. (WWW'20)*, Apr. 2020, pp. 2188–2198.
- [HXSS15] Ruitong Huang, Bing Xu, Dale Schuurmans, Csaba Szepesvári, Learning with a strong adversary, *arXiv preprint*, arXiv:1511.03034, 2015.
- [HYL17] Will Hamilton, Zhitao Ying, Jure Leskovec, Inductive representation learning on large graphs, in: *Proc. 2017 Conf. Neural Information Processing Systems (NIPS'17)*, Long Beach, CA, USA, Dec. 2017, pp. 1024–1034.
- [HZLH17] Rui Huang, Shu Zhang, Tianyu Li, Ran He, Beyond face rotation: global and local perception gan for photorealistic and identity preserving frontal view synthesis, in: *Proc. 2017 Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Oct. 2017, pp. 2439–2448.
- [HZLL19] Xiao Huang, Jingyuan Zhang, Dingcheng Li, Ping Li, Knowledge graph embedding based question answering, in: *Proc. 2019 Int. Conf. Web Search and Data Mining (WSDM'19)*, Melbourne, VIC, Australia, Feb. 2019, pp. 105–113.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: *Proc. 2015 Int. Conf. Computer Vision (ICCV'15)*, Santiago, Chile, Dec. 2015, pp. 1026–1034.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: *Proc. 2016 Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, USA, June 2016, pp. 770–778.
- [IGG03] C. Imhoff, N. Gallemmo, J.G. Geiger, *Mastering Data Warehouse Design: Relational and Dimensional Techniques*, John Wiley & Sons, 2003.
- [IK04] Tsuyoshi Idé, Hisashi Kashima, Eigenspace-based anomaly detection in computer systems, in: *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'04)*, Seattle, WA, USA, Aug. 2004, pp. 440–449.
- [IKA02] T. Imielinski, L. Khachiyan, A. Abdulghani, Cubegrades: generalizing association rules, *Data Mining and Knowledge Discovery* 6 (2002) 219–258.
- [Inm96] W.H. Inmon, *Building the Data Warehouse*, John Wiley & Sons, 1996.
- [Inm16] Bill Inmon, *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*, 1st edition, Technics Publications, LLC, Denville, NJ, USA, 2016.
- [IS15] Sergey Ioffe, Christian Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, *arXiv preprint*, arXiv:1502.03167, 2015.
- [IWM98] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: *Proc. 2000 European Symp. Principles of Data Mining and Knowledge Discovery (PKDD'00)*, Lyon, France, Sept. 1998, pp. 13–23.
- [Jac88] R. Jacobs, Increased rates of convergence through learning rate adaptation, *Neural Networks* 1 (1988) 295–307.
- [Jai10] A.K. Jain, *Data clustering: 50 years beyond k-means*, *Pattern Recognition Letters* 31 (2010).
- [Jam85] M. James, *Classification Algorithms*, John Wiley & Sons, 1985.
- [JBD05] X. Ji, J. Bailey, G. Dong, Mining minimal distinguishing subsequence patterns with gap constraints, in: *Proc. 2005 Int. Conf. Data Mining (ICDM'05)*, Houston, TX, USA, Nov. 2005, pp. 194–201.
- [JD88] A.K. Jain, R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.
- [Jen96] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer Verlag, 1996.
- [JK09] Prateek Jain, Ashish Kapoor, Active learning for large multi-class problems, in: *Proc. 2009 Conf. on Computer Vision and Pattern Recognition (CVPR'09)*, Miami, FL, USA, June 2009, pp. 762–769.

- [JM09] Dan Jurafsky, James H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd edition, Prentice Hall, 2009.
- [JMF99] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a survey, *ACM Computing Surveys* 31 (1999) 264–323.
- [Joh97] G.H. John, *Enhancements to the Data Mining Process*, Ph.D. Thesis, Computer Science Department, Stanford University, 1997.
- [JSD<sup>+</sup>10] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, Jing Gao, Graph regularized transductive classification on heterogeneous information networks, in: *Proc. 2010 Joint European Conf. Machine Learning and Practice of Knowledge Discovery in Databases (ECML-PKDD'10)*, Barcelona, Spain, Sep. 2010, pp. 570–586.
- [JSH19] Haifeng Jin, Qingquan Song, Xia Hu, Auto-keras: an efficient neural architecture search system, in: *Proc. 2019 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'19)*, Anchorage, AK, USA, Aug. 2019, pp. 1946–1956.
- [JTH01] W. Jin, A.K.H. Tung, J. Han, Mining top-n local outliers in large databases, in: *Proc. 2001 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'01)*, San Fransisco, CA, USA, Aug. 2001, pp. 293–298.
- [JTHW06] W. Jin, A.K.H. Tung, J. Han, W. Wang, Ranking outliers using symmetric neighborhood relationship, in: *Proc. 2006 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'06)*, Singapore, April 2006.
- [JTZ21] Baoyu Jing, Hanghang Tong, Yada Zhu, Network of tensor time series, *arXiv preprint*, arXiv: 2102.07736, 2021.
- [JW92] R.A. Johnson, D.A. Wichern, *Applied Multivariate Statistical Analysis*, 3rd ed., Prentice Hall, 1992.
- [JW02] G. Jeh, J. Widom, SimRank: a measure of structural-context similarity, in: *Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'02)*, Edmonton, Canada, July 2002, pp. 538–543.
- [JWHT] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, *An Introduction to Statistical Learning*, Vol. 112, Springer, 2013.
- [JXX18] Baoyu Jing, Pengtao Xie, Eric Xing, On the automatic generation of medical imaging reports, in: *Proc. 2018 Association for Computational Linguistics (ACL'18)*, Melbourne, Australia, July 2018, pp. 2577–2586.
- [JYBJ19] Wengong Jin, Kevin Yang, Regina Barzilay, Tommi Jaakkola, Learning multimodal graph-to-graph translation for molecule optimization, in: *Proc. 2019 Int. Conf. Learning Representations (ICLR'19)*, New Orleans, LA, USA, May 2019.
- [Kas80] G.V. Kass, An exploratory technique for investigating large quantities of categorical data, *Applied Statistics* 29 (1980) 119–127.
- [KB14] Diederik P. Kingma, Jimmy Ba, Adam: a method for stochastic optimization, *arXiv preprint*, arXiv:1412.6980, 2014.
- [Kec01] V. Kecman, *Learning and Soft Computing*, MIT Press, 2001.
- [Ker92] R. Kerber, Discretization of numeric attributes, in: *Proc. 1992 Nat. Conf. Artificial Intelligence (AAAI'92)*, AAAI/MIT Press, 1992, pp. 123–128.
- [KF09] D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, The MIT Press, 2009.
- [KH95] K. Koperski, J. Han, Discovery of spatial association rules in geographic information databases, in: *Proc. 1995 Int. Symp. Large Spatial Databases (SSD'95)*, Portland, ME, USA, Aug. 1995, pp. 47–66.
- [KH97] I. Kononenko, S.J. Hong, Attribute selection for modeling, *Future Generations Computer Systems* 13 (1997) 181–195.

- [KHC97] M. Kamber, J. Han, J.Y. Chiang, Metarule-guided mining of multi-dimensional association rules using data cubes, in: Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, USA, Aug. 1997, pp. 207–210.
- [KHK99] G. Karypis, E.-H. Han, V. Kumar, CHAMELEON: a hierarchical clustering algorithm using dynamic modeling, *Computer* 32 (1999) 68–75.
- [KHMT20] Jian Kang, Jingrui He, Ross Maciejewski, Hanghang Tong, Inform: individual fairness on graph mining, in: Proc. 2020 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'20), Aug. 2020, pp. 379–389.
- [Kim14] Yoon Kim, Convolutional neural networks for sentence classification, in: Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP'14), Doha, Qatar, Oct. 2014, pp. 1746–1751.
- [KJ97] R. Kohavi, G.H. John, Wrappers for feature subset selection, *Artificial Intelligence* 97 (1997) 273–324.
- [KK01] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proc. 2001 Int. Conf. Data Mining (ICDM'01), San Jose, CA, USA, Nov. 2001, pp. 313–320.
- [KKT03] David Kempe, Jon Kleinberg, Éva Tardos, Maximizing the spread of influence through a social network, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug. 2003, pp. 137–146.
- [KKW<sup>+</sup>10] H.S. Kim, S. Kim, T. Weninger, J. Han, T. Abdelzaher, NDPMine: efficiently mining discriminative numerical features for pattern-based classification, in: Proc. 2010 European Conf. Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD'10), Barcelona, Spain, Sept. 2010.
- [KKZ09] H.-P. Kriegel, P. Kroeger, A. Zimek, Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering, *ACM Transactions on Knowledge Discovery from Data* 3 (2009) 1–58.
- [KL17] Pang Wei Koh, Percy Liang, Understanding black-box predictions via influence functions, in: Proc. 2017 Int. Conf. Machine Learning (ICML'17), Sydney, NSW, Australia, Aug. 2017, pp. 1885–1894.
- [KLA<sup>+</sup>08] M. Khan, H. Le, H. Ahmadi, T. Abdelzaher, J. Han, DustMiner: troubleshooting interactive complexity bugs in sensor networks, in: Proc. 2008 ACM Int. Conf. Embedded Networked Sensor Systems (SenSys'08), Raleigh, NC, USA, Nov. 2008.
- [Kle99] Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM* 46 (5) (1999) 604–632.
- [KLV<sup>+</sup>98] R.L. Kennedy, Y. Lee, B. Van Roy, C.D. Reed, R.P. Lippman, *Solving Data Mining Problems Through Pattern Recognition*, Prentice Hall, 1998.
- [KMN<sup>+</sup>02] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002) 881–892.
- [KMR<sup>+</sup>94] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, A.I. Verkamo, Finding interesting rules from large sets of discovered association rules, in: Proc. 1994 Int. Conf. Information and Knowledge Management (CIKM'1994), Gaithersburg, MD, USA, Nov. 1994, pp. 401–408.
- [KMY<sup>+</sup>16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, Dave Bacon, Federated learning: strategies for improving communication efficiency, arXiv preprint, arXiv:1610.05492, 2016.
- [KN97] E. Knorr, R. Ng, A unified notion of outliers: properties and computation, in: Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, USA, Aug. 1997, pp. 219–222.
- [KN98] E. Knorr, R. Ng, Algorithms for mining distance-based outliers in large datasets, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 392–403.

- [KNT00] E.M. Knorr, R.T. Ng, V. Tucakov, Distance-based outliers: algorithms and applications, *The VLDB Journal* 8 (2000) 237–253.
- [KO16] Ashish Khetan, Sewoong Oh, Achieving budget-optimality with adaptive schemes in crowdsourcing, in: *Proc. 2016 Conf. Neural Information Processing Systems (NIP'16)*, Barcelona, Spain, Dec. 2016, pp. 4844–4852.
- [Koh95] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: *Proc. 1995 Joint Int. Conf. Artificial Intelligence (IJCAI'95)*, Montreal, Canada, Aug. 1995, pp. 1137–1143.
- [Kol93] J.L. Kolodner, *Case-Based Reasoning*, Morgan Kaufmann, 1993.
- [Kon95] I. Kononenko, On biases in estimating multi-valued attributes, in: *Proc. 14th Joint Int. Conf. Artificial Intelligence (IJCAI'95)*, Vol. 2, Montreal, Canada, Aug. 1995, pp. 1034–1040.
- [Kot88] P. Koton, Reasoning about evidence in causal explanation, in: *Proc. 1988 Nat. Conf. Artificial Intelligence (AAAI'88)*, Aug. 1988, pp. 256–263.
- [KP00] Eamonn J. Keogh, Michael J. Pazzani, Scaling up dynamic time warping for data mining applications, in: *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, Boston, MA, USA, Aug. 2000, pp. 285–289.
- [KPS03] R.M. Karp, C.H. Papadimitriou, S. Shenker, A simple algorithm for finding frequent elements in streams and bags, *ACM Transactions on Database Systems* 28 (2003).
- [KR90] L. Kaufman, P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.
- [KR02] R. Kimball, M. Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, 2nd ed., John Wiley & Sons, 2002.
- [KRTM08] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, *The Data Warehouse Lifecycle Toolkit*, John Wiley & Sons, Hoboken, NJ, USA, 2008.
- [KS95] Ron Kohavi, Dan Sommerfield, Feature subset selection using the wrapper method: overfitting and dynamic search space topology, in: *Proc. 1995 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'95)*, Montreal, Canada, Aug. 1995, pp. 192–197.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: *Proc. 2012 Conf. Neural Information Processing Systems (NIPS'12)*, Lake Tahoe, NV, USA, Dec. 2012, pp. 1097–1105.
- [KSS16] Gilad Katz, Eui Chul Richard Shin, Dawn Song, Explorekit: automatic feature generation and selection, in: *Proc. 2016 Int. Conf. Data Mining (ICDM'16)*, Barcelona, Spain, Dec. 2016, pp. 979–984.
- [KST18] Udayan Khurana, Horst Samulowitz, Deepak Turaga, Feature engineering for predictive modeling using reinforcement learning, in: *Proc. 2018 AAAI Conf. Artificial Intelligence (AAAI'18)*, New Orleans, LA, USA, Feb. 2018, pp. 3407–3414.
- [KSV<sup>+</sup>16] Danai Koutra, Neil Shah, Joshua T. Vogelstein, Brian Gallagher, Christos Faloutsos, Deltacon: principled massive-graph similarity function with attribution, *ACM Transactions on Knowledge Discovery from Data* 10 (3) (2016) 28:1–28:43.
- [KSW15] Durk P. Kingma, Tim Salimans, Max Welling, Variational dropout and the local reparameterization trick, in: *Proc. 2015 Conf. Neural Information Processing Systems (NIPS'15)*, Montreal, QC, Canada, Dec. 2015, pp. 2575–2583.
- [KSZ08] H.-P. Kriegel, M. Schubert, A. Zimek, Angle-based outlier detection in high-dimensional data, in: *Proc. 2008 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'08)*, Las Vegas, NV, USA, Aug. 2008, pp. 444–452.
- [KV15] James Max Kanter, Kalyan Veeramachaneni, Deep feature synthesis: towards automating data science endeavors, in: *Proc. 2015 Int. Conf. Data Science and Advanced Analytics (NSAA'15)*, Paris, France, Oct. 2015, pp. 1–10.

- [KW<sup>+</sup>52] Jack Kiefer, Jacob Wolfowitz, et al., Stochastic estimation of the maximum of a regression function, *The Annals of Mathematical Statistics* 23 (3) (1952) 462–466.
- [KW13] Diederik P. Kingma, Max Welling, Auto-encoding variational Bayes, arXiv preprint, arXiv:1312.6114, 2013.
- [KW16a] Thomas N. Kipf, Max Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint, arXiv:1609.02907, 2016.
- [KW16b] Thomas N. Kipf, Max Welling, Variational graph auto-encoders, arXiv preprint, arXiv:1611.07308, 2016.
- [Lam98] W. Lam, Bayesian network refinement via machine learning approach, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (1998) 240–252.
- [LAS<sup>+</sup>15] Himabindu Lakkaraju, Everaldo Aguiar, Carl Shan, David Miller, Nasir Bhanpuri, Rayid Ghani, Kecia L. Addison, A machine learning framework to identify students at risk of adverse academic outcomes, in: *Proc. 2015 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'15)*, Sydney, NSW, Australia, Aug. 2015, pp. 1909–1918.
- [Lau95] S.L. Lauritzen, The EM algorithm for graphical association models with missing data, *Computational Statistics & Data Analysis* 19 (1995) 191–201.
- [LB11] G. Linoff, M.J.A. Berry, *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*, 3rd ed., John Wiley & Sons, 2011.
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* 86 (11) (1998) 2278–2324.
- [LBD<sup>+</sup>89] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, Lawrence D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Computation* 1 (4) (1989) 541–551.
- [LBH06] Steven J. Leon, Ion Bica, Tiina Hohn, *Linear Algebra with Applications*, Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2006.
- [LBS<sup>+</sup>16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer, Neural architectures for named entity recognition, in: *Proc. 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'16)*, San Diego, CA, USA, June 2016, pp. 260–270.
- [LCH<sup>+</sup>09] D. Lo, H. Cheng, J. Han, S. Khoo, C. Sun, Classification of software behaviors for failure detection: a discriminative pattern mining approach, in: *Proc. 2009 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'09)*, Paris, France, June 2009.
- [LCK<sup>+</sup>10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, Zoubin Ghahramani, Kronecker graphs: an approach to modeling networks, *Journal of Machine Learning Research* 11 (2) (2010).
- [LCT20] Vivian Lai, Samuel Carton, Chenhao Tan, Harnessing explanations to bridge AI and humans, CoRR, arXiv:2003.07370 [abs], 2020.
- [LCW<sup>+</sup>17] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, Huan Liu, Feature selection: a data perspective, *ACM Computing Surveys* 50 (6) (2017) 1–45.
- [LDH<sup>+</sup>08] C.X. Lin, B. Ding, J. Han, F. Zhu, B. Zhao, Text Cube: computing IR measures for multidimensional text database analysis, in: *Proc. 2008 Int. Conf. Data Mining (ICDM'08)*, Pisa, Italy, Dec. 2008.
- [LDL<sup>+</sup>12] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, Divesh Srivastava, Truth finding on the deep web: is the problem solved?, *Proceedings of the VLDB Endowment* 6 (2) (2012) 97–108.
- [LDL<sup>+</sup>17] Yaliang Li, Nan Du, Chaochun Liu, Yusheng Xie, Wei Fan, Qi Li, Jing Gao, Huan Sun, Reliable medical diagnosis from crowdsourcing: discover trustworthy answers from non-experts, in: *Proc. 2017 Int. Conf. Web Search and Data Mining (WSDM'17)*, Cambridge, United Kingdom, Feb. 2017, pp. 253–261.



- [LDR00] J. Li, G. Dong, K. Ramamohanarao, Making use of the most expressive jumping emerging patterns for classification, in: Proc. 2000 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'00), Kyoto, Japan, April 2000, pp. 220–232.
- [LDS89] Y. Le Cun, J.S. Denker, S.A. Solla, Optimal brain damage, in: Proc. 1989 Conf. Neural Information Processing Systems (NIPS'89), Denver, CO, USA, Nov 1989.
- [Le98] H. Liu, H. Motoda (Eds.), Feature Extraction, Construction, and Selection: A Data Mining Perspective, Kluwer Academic, 1998.
- [Le13] Quoc V. Le, Building high-level features using large scale unsupervised learning, in: Proc. 2013 Int. Conf. Acoustics, Speech and Signal Processing (ICASSP'13), Vancouver, BC, Canada, May 2013, pp. 8595–8598.
- [Lea96] D.B. Leake, CBR in context: the present and future, in: D.B. Leake (Ed.), Cased-Based Reasoning: Experiences, Lessons, and Future Directions, AAAI Press, 1996, pp. 3–30.
- [LFP14] Vince Lyzinski, Donniell E. Fishkind, Carey E. Priebe, Seeded graph matching for correlated Erdős-Rényi graphs, *Journal of Machine Learning Research* 15 (1) (2014) 3513–3540.
- [LGB<sup>+</sup>16] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, Bill Dolan, A diversity-promoting objective function for neural conversation models, in: Proc. 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'16), San Diego, CA, USA, June 2016, pp. 110–119.
- [LGGRG<sup>+</sup>20] Julián Luengo, Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, Francisco Herrera, *Big Data Preprocessing: Enabling Smart Data*, Springer, 2020.
- [LGM<sup>+</sup>15] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, Jiawei Han, A survey on truth discovery, *SIGKDD Explorations* 17 (2) (2015) 1–16.
- [LGWL14] Chen Luo, Renchu Guan, Zhe Wang, Chenghua Lin, Hetpathmine: a novel transductive classification algorithm on heterogeneous information networks, in: Proc. 2014 European Conf. Information Retrieval (ECIR'14), Amsterdam, the Netherlands, Apr. 2014, pp. 210–221.
- [LHC97] B. Liu, W. Hsu, S. Chen, Using general impressions to analyze discovered classification rules, in: Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, USA, Aug. 1997, pp. 31–36.
- [LHF98] H. Lu, J. Han, L. Feng, Stock movement and n-dimensional inter-transaction association rules, in: Proc. 1998 SIGMOD Workshop Data Mining and Knowledge Discovery (DMKD'98), Seattle, WA, USA, June 1998, pp. 12:1–12:7.
- [LHG04] X. Li, J. Han, H. Gonzalez, High-dimensional OLAP: a minimal cubing approach, in: Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04), Toronto, Canada, Aug. 2004, pp. 528–539.
- [LHL<sup>+</sup>17] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, Min Sun, Tactics of adversarial attack on deep reinforcement learning agents, arXiv preprint, arXiv:1703.06748, 2017.
- [LHM98] B. Liu, W. Hsu, Y. Ma, Integrating classification and association rule mining, in: Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98), New York, NY, USA, Aug. 1998, pp. 80–86.
- [LHP01] W. Li, J. Han, J. Pei, CMAR: accurate and efficient classification based on multiple class-association rules, in: Proc. 2001 Int. Conf. Data Mining (ICDM'01), San Jose, CA, USA, Nov. 2001, pp. 369–376.
- [LHP<sup>+</sup>15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, Continuous control with deep reinforcement learning, arXiv preprint, arXiv:1509.02971, 2015.
- [LHTD02] H. Liu, F. Hussain, C.L. Tan, M. Dash, Discretization: an enabling technique, *Data Mining and Knowledge Discovery* 6 (2002) 393–423.

- [LHXS06] H. Liu, J. Han, D. Xin, Z. Shao, Mining frequent patterns on very high dimensional data: a top-down row enumeration approach, in: Proc. 2006 SIAM Int. Conf. Data Mining (SDM'06), Bethesda, MD, USA, April 2006.
- [LHY<sup>+</sup>08] X. Li, J. Han, Z. Yin, J.-G. Lee, Y. Sun, Sampling Cube: a framework for statistical OLAP over sampling data, in: Proc. 2008 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'08), Vancouver, BC, Canada, June 2008.
- [Lik01] Aristidis Likas, Probability density estimation using artificial neural networks, *Computer Physics Communications* 135 (2) (2001) 167–175.
- [Lit74] William A. Little, The existence of persistent states in the brain, *Mathematical Biosciences* 19 (1–2) (1974) 101–120.
- [Liu06] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, Springer, 2006.
- [Liu12] Bing Liu, *Sentiment Analysis and Opinion Mining*, Morgan/Claypool Publishers, 2012.
- [Liu20] B. Liu, *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*, Studies in Natural Language Processing, Cambridge University Press, 2020.
- [LJD<sup>+</sup>17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar, Hyperband: a novel bandit-based approach to hyperparameter optimization, *Journal of Machine Learning Research* 18 (1) (2017) 6765–6816.
- [LJK00] J. Laurikkala, M. Juhola, E. Kentala, Informal identification of outliers in medical data, in: Proc. 5th Int. Workshop on Intelligent Data Analysis in Medicine and Pharmacology, Berlin, Germany, Aug. 2000, pp. 20–24.
- [LKCH03] Y.-K. Lee, W.-Y. Kim, Y.D. Cai, J. Han, CoMine: efficient mining of correlated patterns, in: Proc. 2003 Int. Conf. Data Mining (ICDM'03), Melbourne, FL, USA, Nov. 2003, pp. 581–584.
- [LKEW16] Zachary C. Lipton, David C. Kale, Charles Elkan, Randall Wetzel, Learning to diagnose with lstm recurrent neural networks, in: Proc. 2016 Int. Conf. Learning Representations (ICLR'16), San Juan, Puerto Rico, May 2016.
- [LKF05] Jure Leskovec, Jon Kleinberg, Christos Faloutsos, Graphs over time: densification laws, shrinking diameters and possible explanations, in: Proc. 2005 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'05), Chicago, IL, USA, Aug. 2005, pp. 177–187.
- [LKL14] Martin Längkvist, Lars Karlsson, Amy Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, *Pattern Recognition Letters* 42 (2014) 11–24.
- [LL17] Scott M. Lundberg, Su-In Lee, A unified approach to interpreting model predictions, in: Proc. 2017 Conf. Neural Information Processing Systems (NIPS'17), Long Beach, CA, USA, Dec. 2017, pp. 4768–4777.
- [LLK19] Junhyun Lee, Inyeop Lee, Jaewoo Kang, Self-attention graph pooling, arXiv preprint, arXiv:1904.08082, 2019.
- [LLLY03] G. Liu, H. Lu, W. Lou, J.X. Yu, On computing, storing and querying frequent patterns, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug. 2003, pp. 607–612.
- [LLMZ04] Z. Li, S. Lu, S. Myagmar, Y. Zhou, CP-Miner: a tool for finding copy-paste and related bugs in operating system code, in: Proc. 2004 Symp. Operating Systems Design and Implementation (OSDI'04), San Francisco, CA, USA, Dec. 2004.
- [Llo57] S.P. Lloyd, Least squares quantization in PCM, *IEEE Transactions on Information Theory* 28 (1982) 128–137; original version: Technical Report, Bell Labs, 1957.
- [LLS00] T.-S. Lim, W.-Y. Loh, Y.-S. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, *Machine Learning* 40 (2000) 203–228.
- [LLS20] Zhiyuan Liu, Yankai Lin, Maosong Sun, *Representation Learning for Natural Language Processing*, Springer, 2020.

- [LM97] K. Laskey, S. Mahoney, Network fragments: representing knowledge for constructing probabilistic models, in: Proc. 1997 Conf. Uncertainty in Artificial Intelligence (UAI'97), Morgan Kaufmann, San Francisco, CA, USA, Aug. 1997, pp. 334–341.
- [LM98] H. Liu, H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, Kluwer Academic, 1998.
- [LM05] Richard J. Larsen, Morris L. Marx, An Introduction to Mathematical Statistics, Prentice Hall, 2005.
- [LMS<sup>+</sup>17] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, Dan Jurafsky, Adversarial learning for neural dialogue generation, arXiv preprint, arXiv:1701.06547, 2017.
- [LNHP99] L.V.S. Lakshmanan, R. Ng, J. Han, A. Pang, Optimization of constrained frequent set queries with 2-variable constraints, in: Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99), Philadelphia, PA, USA, June 1999, pp. 157–168.
- [LNV<sup>+</sup>18] Scott M. Lundberg, Bala Nair, Monica S. Vavilala, Mayumi Horibe, Michael J. Eisses, Trevor Adams, David E. Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al., Explainable machine-learning predictions for the prevention of hypoxaemia during surgery, *Nature Biomedical Engineering* 2 (10) (2018) 749–760.
- [LOG<sup>+</sup>19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, RoBERTa: a robustly optimized bert pretraining approach, arXiv preprint, arXiv:1907.11692, 2019.
- [Los01] D. Loshin, Enterprise Knowledge Management: The Data Quality Approach, Morgan Kaufmann, 2001.
- [LPH02] L.V.S. Lakshmanan, J. Pei, J. Han, Quotient cube: how to summarize the semantics of a data cube, in: Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), Hong Kong, China, Aug. 2002, pp. 778–789.
- [LPWH02] J. Liu, Y. Pan, K. Wang, J. Han, Mining frequent item sets by opportunistic projection, in: Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02), Edmonton, Canada, July 2002, pp. 239–248.
- [LPZ03] L.V.S. Lakshmanan, J. Pei, Y. Zhao, QC-Trees: an efficient summary structure for semantic OLAP, in: Proc. 2003 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'03), San Diego, CA, USA, June 2003, pp. 64–75.
- [LRC<sup>+</sup>19] Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wenbing Huang, Junzhou Huang, Semi-supervised graph classification: a hierarchical graph perspective, in: Proc. 2019 the Web Conf. (WWW'19), San Francisco, CA, USA, May 2019, pp. 972–982.
- [LS95] H. Liu, R. Setiono, Chi2: feature selection and discretization of numeric attributes, in: Proc. 1995 IEEE Int. Conf. Tools with AI (ICTAI'95), Washington, DC, USA, Nov. 1995, pp. 388–391.
- [LS97] W.Y. Loh, Y.S. Shih, Split selection methods for classification trees, *Statistica Sinica* 7 (1997) 815–840.
- [LS99] Daniel D. Lee, H. Sebastian Seung, Learning the parts of objects by non-negative matrix factorization, *Nature* 401 (6755) (1999) 788–791.
- [LSBZ87] P. Langley, H.A. Simon, G.L. Bradshaw, J.M. Zytkow, Scientific Discovery: Computational Explorations of the Creative Processes, MIT Press, 1987.
- [LSG11] Annan Li, Shiguang Shan, Wen Gao, Coupled bias–variance tradeoff for cross-pose face recognition, *IEEE Transactions on Image Processing* 21 (1) (2011) 305–315.
- [LSH17] Jialu Liu, Jingbo Shang, Jiawei Han, Phrase Mining from Massive Text and Its Applications, Morgan & Claypool, 2017.
- [LSM<sup>+</sup>17] Christos Louizos, Uri Shalit, Joris Mooij, David Sontag, Richard Zemel, Max Welling, Causal effect inference with deep latent-variable models, arXiv preprint, arXiv:1705.08821, 2017.
- [LSST<sup>+</sup>02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, Chris Watkins, Text classification using string kernels, *Journal of Machine Learning Research* 2 (Feb 2002) 419–444.

- [LSW97] B. Lent, A. Swami, J. Widom, Clustering association rules, in: Proc. 1997 Int. Conf. Data Engineering (ICDE'97), Birmingham, England, April 1997, pp. 220–231.
- [LSW<sup>+</sup>15] Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, Jiawei Han, Mining quality phrases from massive text corpora, in: Proc. 2015 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'15), Melbourne, Australia, May 2015, pp. 1729–1744.
- [LSY18] Hanxiao Liu, Karen Simonyan, Yiming Yang, Darts: differentiable architecture search, arXiv preprint, arXiv:1806.09055, 2018.
- [LT15] Liangyue Li, Hanghang Tong, The child is father of the man: foresee the success at the early stage, in: Proc. 2015 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'15), Sydney, NSW, Australia, Aug. 2015, pp. 655–664.
- [LT20] Liangyue Li, Hanghang Tong, Computational Approaches to the Network Science of Teams, Cambridge University Press, 2020.
- [LTBZ15] Yujia Li, Daniel Tarlow, Marc Brockschmidt, Richard Zemel, Gated graph sequence neural networks, arXiv preprint, arXiv:1511.05493, 2015.
- [LTC<sup>+</sup>15] Liangyue Li, Hanghang Tong, Nan Cao, Kate Ehrlich, Yu-Ru Lin, Norbou Buchler, Replacing the irreplaceable: fast algorithms for team member recommendation, in: Proc. 2015 Int. Conf. World Wide Web (WWW'15), Florence, Italy, May 2015, pp. 636–646.
- [LTC<sup>+</sup>17] Liangyue Li, Hanghang Tong, Nan Cao, Kate Ehrlich, Yu-Ru Lin, Norbou Buchler, Enhancing team composition in professional networks: problem definitions and fast solutions, IEEE Transactions on Knowledge and Data Engineering 29 (3) (2017) 613–626.
- [LTH<sup>+</sup>17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al., Photo-realistic single image super-resolution using a generative adversarial network, in: Proc. 2017 Conf. Computer Vision and Pattern Recognition (CVPR'17), Honolulu, HI, USA, July 2017, pp. 4681–4690.
- [LTL18] Liangyue Li, Hanghang Tong, Huan Liu, Towards explainable networked prediction, in: Proc. 2018 Int. Conf. Information and Knowledge Management (CIKM'18), Torino, Italy, Oct. 2018, pp. 1819–1822.
- [LTQ<sup>+</sup>18] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, Tie-Yan Liu, Neural architecture optimization, arXiv preprint, arXiv:1808.07233, 2018.
- [LTTF16] Liangyue Li, Hanghang Tong, Jie Tang, Wei Fan, *iPath*: forecasting the pathway to impact, in: Proc. 2016 SIAM Int. Conf. Data Mining (SDM'16), May 2016, pp. 468–476.
- [LW<sup>+</sup>17] Liangyue Li, Hanghang Tong, Yong Wang, Conglei Shi, Nan Cao, Norbou Buchler, Is the whole greater than the sum of its parts?, in: Proc. 2017 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'17), Aug. 2017, pp. 295–304.
- [Lux07] U. Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (2007) 395–416.
- [LV88] W.Y. Loh, N. Vanichsetakul, Tree-structured classification via generalized discriminant analysis, Journal of the American Statistical Association 83 (1988) 715–728.
- [LW67] G.N. Lance, W.T. Williams, A general theory of classificatory sorting strategies: 1. Hierarchical systems, Computer Journal 9 (4) (02 1967) 373–380.
- [LWLQ21] Tianyang Lin, Yuxin Wang, Xiangyang Liu, Xipeng Qiu, A survey of transformers, arXiv preprint, arXiv:2106.04554, 2021.
- [LYBP16] Jiasen Lu, Jianwei Yang, Dhruv Batra, Devi Parikh, Hierarchical question-image co-attention for visual question answering, in: Proc. 2016 Conf. Neural Information Processing Systems (NIPS'16), Barcelona, Spain, Dec. 2016, pp. 289–297.
- [LYSL17] Yaguang Li, Rose Yu, Cyrus Shahabi, Yan Liu, Diffusion convolutional recurrent neural network: data-driven traffic forecasting, arXiv preprint, arXiv:1707.01926, 2017.
- [LZ05] Z. Li, Y. Zhou, PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code, in: Proc. 2005 ACM SIGSOFT Symp. Foundations Software Eng. (FSE'05), Lisbon, Portugal, Sept. 2005.

- [LZXZ18] Kaixiang Lin, Renyu Zhao, Zhe Xu, Jiayu Zhou, Efficient large-scale fleet management via multi-agent deep reinforcement learning, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 1774–1783.
- [MA03] S. Mitra, T. Acharya, *Data Mining: Multimedia, Soft Computing, and Bioinformatics*, John Wiley & Sons, 2003.
- [MAA05] A. Metwally, D. Agrawal, A. El Abbadi, Efficient computation of frequent and top-k elements in data streams, in: Proc. 2005 Int. Conf. Database Theory (ICDT'05), Edinburgh, UK, Jan. 2005.
- [Mac67] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proc. 5th Berkeley Symp. Mathematical Statistics and Probability, Vol. 1, 1967, pp. 281–297.
- [Mag94] J. Magidson, The CHAID approach to segmentation modeling: CHI-squared automatic interaction detection, in: R.P. Bagozzi (Ed.), *Advanced Methods of Marketing Research*, Blackwell Business, 1994, pp. 118–159.
- [MAR96] M. Mehta, R. Agrawal, J. Rissanen, SLIQ: a fast scalable classifier for data mining, in: Proc. 1996 Int. Conf. Extending Database Technology (EDBT'96), Avignon, France, Mar. 1996, pp. 18–32.
- [Mar09] S. Marsland, *Machine Learning: An Algorithmic Perspective*, Chapman and Hall/CRC, 2009.
- [MATL<sup>+</sup>17] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, Saurabh Singh, No fuss distance metric learning using proxies, in: Proc. 2017 Int. Conf. Computer Vision (ICCV'17), Venice, Italy, Oct. 2017.
- [MB88] G.J. McLachlan, K.E. Basford, *Mixture Models: Inference and Applications to Clustering*, John Wiley & Sons, 1988.
- [MBA<sup>+</sup>09] Mary McGlohn, Stephen Bay, Markus G. Anderle, David M. Steier, Christos Faloutsos, SNARE: a link analytic system for graph labeling and risk detection, in: Proc. 2009 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'09), Paris, France, June 2009, pp. 1265–1274.
- [MBM<sup>+</sup>16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: Proc. 2016 Int. Conf. Machine Learning (ICML'16), New York City, NY, USA, June 2016, pp. 1928–1937.
- [MBM<sup>+</sup>17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, Michael M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model cnns, in: Proc. 2017 Conf. Computer Vision and Pattern Recognition (CVPR'17), Honolulu, HI, USA, July 2017, pp. 5115–5124.
- [MC03] M.V. Mahoney, P.K. Chan, Learning rules for anomaly detection of hostile network traffic, in: Proc. 2003 Int. Conf. Data Mining (ICDM'03), Melbourne, FL, USA, Nov. 2003.
- [MC12] Fionn Murtagh, Pedro Contreras, Algorithms for hierarchical clustering: an overview, *WIREs Data Mining and Knowledge Discovery* 2 (1) (2012) 86–97.
- [MD88] M. Muralikrishna, D.J. DeWitt, Equi-depth histograms for estimating selectivity factors for multi-dimensional queries, in: Proc. 1988 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'88), Chicago, IL, USA, June 1988, pp. 28–36.
- [MD09] Michael W. Mahoney, Petros Drineas, CUR matrix decompositions for improved data analysis, *Proceedings of the National Academy of Sciences of the United States of America* (2009) 697–702.
- [Mei03] M. Meilă, Comparing clusterings by the variation of information, in: Proc. 16th Annual Conf. Computational Learning Theory (COLT'03), Washington, DC, USA, Aug. 2003, pp. 173–187.
- [Mei05] M. Meilă, Comparing clusterings: an axiomatic view, in: Proc. 22nd Int. Conf. Machine Learning (ICML'05), Bonn, Germany, 2005, pp. 577–584.
- [MFS95] D. Malerba, E. Floriana, G. Semeraro, A further comparison of simplification methods for decision tree induction, in: D. Fisher, H. Lenz (Eds.), *Learning from Data: AI and Statistics*, Springer Verlag, 1995.

- [MGK<sup>+</sup>08] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, Bhavani Thuraisingham, A practical approach to classify evolving data streams: training with limited amount of labeled data, in: Proc. 2008 Int. Conf. Data Mining (ICDM'08), Pisa, Italy, Dec. 2008, pp. 929–934.
- [MGKV06] A. Machanavajjhala, J. Gehrke, D. Kifer, M. Venkatasubramaniam, L-diversity: privacy beyond k-anonymity, in: Proc. 2006 Int. Conf. Data Engineering (ICDE'06), Atlanta, GA, USA, Apr. 2006, p. 24.
- [MH95] J.K. Martin, D.S. Hirschberg, The time complexity of decision tree induction, Technical Report ICS-TR 95-27, Department of Information and Computer Science, Univ. California, Irvine, CA, USA, Aug. 1995.
- [MHN] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proc. ICML (Vol. 30. No. 1), 2013.
- [MHW<sup>+</sup>19] Yu Meng, Jiaxin Huang, Guangyuan Wang, Chao Zhang, Honglei Zhuang, Lance M. Kaplan, Jiawei Han, Spherical text embedding, in: Proc. 2019 Conf. Neural Information Processing Systems (NeurIPS'19), Vancouver, BC, Canada, Dec. 2019, pp. 8206–8215.
- [MHW<sup>+</sup>20] Yu Meng, Jiaxin Huang, Guangyuan Wang, Zihan Wang, Chao Zhang, Yu Zhang, Jiawei Han, Discriminative topic mining via category-name guided text embedding, in: Proc. 2020 the Web Conf. (WWW'20), Apr. 2020, pp. 2121–2132.
- [MIA99] Malik Magdon-Ismail, Amir Atiya, Neural Networks for Density Estimation, 1999.
- [Mic92] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, 1992.
- [Min89] J. Mingers, An empirical comparison of pruning methods for decision-tree induction, *Machine Learning* 4 (1989) 227–243.
- [Mir98] B. Mirkin, Mathematical classification and clustering, *Journal of Global Optimization* 12 (1998) 105–108.
- [Mit96] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.
- [Mit97] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.
- [MK91] M. Manago, Y. Kodratoff, Induction of decision trees from complex structured data, in: G. Piatetsky-Shapiro, W.J. Frawley (Eds.), Knowledge Discovery in Databases, AAAI/MIT Press, 1991, pp. 289–306.
- [MKS<sup>+</sup>13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint, arXiv: 1312.5602, 2013.
- [MKS<sup>+</sup>15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Belle-mare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al., Human-level control through deep reinforcement learning, *Nature* 518 (7540) (2015) 529–533.
- [ML14] Fionn Murtagh, Pierre Legendre, Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion?, *Journal of Classification* 31 (3) (2014) 274–295.
- [MLS<sup>+</sup>18] Chenglin Miao, Qi Li, Lu Su, Mengdi Huai, Wenjun Jiang, Jing Gao, Attack under disguise: an intelligent data poisoning attack mechanism in crowdsourcing, in: Proc. 2018 the Web Conf. (WWW'18), Lyon, France, Apr. 2018, pp. 13–22.
- [MM95] J. Major, J. Mangano, Selecting among rules induced from a hurricane database, *Journal of Intelligent Information Systems* 4 (1995) 39–52.
- [MM02] G. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), Hong Kong, China, Aug. 2002, pp. 346–357.
- [MN89] M. Mézard, J.-P. Nadal, Learning in feedforward layered networks: the tiling algorithm, *Journal of Physics* 22 (1989) 2191–2204.
- [MO04] S.C. Madeira, A.L. Oliveira, Biclustering algorithms for biological data analysis: a survey, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1 (2004) 24–45.

- [Mol20] Christoph Molnar, *Interpretable Machine Learning*, Lulu.com, 2020.
- [MP43] Warren S. McCulloch, Walter Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics* 5 (4) (1943) 115–133.
- [MP69] M.L. Minsky, S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.
- [MS01] Christopher D. Manning, Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 2001.
- [MS03a] M. Markou, S. Singh, Novelty detection: a review—part 1: statistical approaches, *Signal Processing* 83 (2003) 2481–2497.
- [MS03b] M. Markou, S. Singh, Novelty detection: a review—part 2: neural network based approaches, *Signal Processing* 83 (2003) 2499–2521.
- [MSC<sup>+</sup>13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, Jeff Dean, Distributed representations of words and phrases and their compositionality, in: *Proc. 2013 Conf. Neural Information Processing Systems (NIPS'13)*, Lake Tahoe, NV, USA, Dec. 2013, pp. 3111–3119.
- [MSK19] Aman Mehta, Aashay Singhal, Kamalakar Karlapalem, Scalable knowledge graph construction over text using deep learning based predicate mapping, in: *Proc. 2019 the Web Conf. (WWW'19)*, San Francisco, CA, USA, May 2019, pp. 705–713.
- [MSS<sup>+</sup>98] Sebastian Mika, Bernhard Schölkopf, Alexander J. Smola, Klaus-Robert Müller, Matthias Scholz, Gunnar Rätsch, Kernel pca and de-noising in feature spaces, in: *NIPS*, Vol. 11, 1998, pp. 536–542.
- [MST94] D. Michie, D.J. Spiegelhalter, C.C. Taylor, *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, Chichester, UK, 1994.
- [MST20] Ramaravind K. Mothilal, Amit Sharma, Chenhao Tan, Explaining machine learning classifiers through diverse counterfactual explanations, in: *Proc. 2020 Conf. Fairness, Accountability, and Transparency (FAT'20)*, May 2020, pp. 607–617.
- [MSZH18] Yu Meng, Jiaming Shen, Chao Zhang, Jiawei Han, Weakly-supervised neural text classification, in: *Proc. 2018 Int. Conf. Information and Knowledge Management (CIKM'18)*, Torino, Italy, Oct. 2018, pp. 983–992.
- [MTV94] H. Mannila, H. Toivonen, A.I. Verkamo, Efficient algorithms for discovering association rules, in: *Proc. 1994 Workshop Knowledge Discovery in Databases (KDD'94)*, Seattle, WA, USA, July 1994, pp. 181–192.
- [MTV97] H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery* 1 (1997) 259–289.
- [Mur98] S.K. Murthy, Automatic construction of decision trees from data: a multi-disciplinary survey, *Data Mining and Knowledge Discovery* 2 (1998) 345–389.
- [MV13] Fragkiskos D. Malliaros, Michalis Vazirgiannis, Clustering and community detection in directed networks: a survey, *Physics Reports* 533 (4) (2013) 95–142.
- [MVDGB08] Lukas Meier, Sara Van De Geer, Peter Bühlmann, The group lasso for logistic regression, *Journal of the Royal Statistical Society, Series B, Statistical Methodology* 70 (1) (2008) 53–71.
- [MW99] D. Meretakakis, B. Wüthrich, Extending naive Bayes classifiers using long itemsets, in: *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, USA, Aug. 1999, pp. 165–174.
- [MXC<sup>+</sup>07] Q. Mei, D. Xin, H. Cheng, J. Han, C. Zhai, Semantic annotation of frequent patterns, *ACM Transactions on Knowledge Discovery from Data* 15 (2007) 321–348.
- [MY88] Katta G. Murty, Feng-Tien Yu, *Linear Complementarity, Linear and Nonlinear Programming*, Vol. 3, Citeseer, 1988.
- [MY97] R.J. Miller, Y. Yang, Association rules over interval data, in: *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, Tucson, AZ, USA, May 1997, pp. 452–461.

- [MZH<sup>+</sup>20a] Yu Meng, Yunyi Zhang, Jiaxin Huang, Chenyan Xiong, Heng Ji, Chao Zhang, Jiawei Han, Text classification using label names only: a language model self-training approach, in: Proc. 2020 Conf. Empirical Methods in Natural Language Processing (EMNLP'20), Nov. 2020, pp. 9006–9017.
- [MZH<sup>+</sup>20b] Yu Meng, Yunyi Zhang, Jiaxin Huang, Yu Zhang, Chao Zhang, Jiawei Han, Hierarchical topic mining via joint spherical tree and text embedding, in: Proc. 2020 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'20), 2020, pp. 1908–1917.
- [MZH<sup>+</sup>21] Yu Meng, Yunyi Zhang, Jiaxin Huang, Xuan Wang, Yu Zhang, Heng Ji, Jiawei Han, Distantly-supervised named entity recognition with noise-robust learning and language model augmented self-training, in: Proc. 2021 Conf. Empirical Methods in Natural Language Processing (EMNLP'21), Nov. 2021.
- [NAK16] Mathias Niepert, Mohamed Ahmed, Konstantin Kutzkov, Learning convolutional neural networks for graphs, in: Proc. 2016 Int. Conf. Machine Learning (ICML'16), New York City, NY, USA, June 2016, pp. 2014–2023.
- [NB86] T. Niblett, I. Bratko, Learning decision rules in noisy domains, in: M.A. Brammer (Ed.), Expert Systems '86: Research and Development in Expert Systems III, British Computer Society Specialist Group on Expert Systems, Dec. 1986, pp. 25–34.
- [NC03] C.C. Noble, D.J. Cook, Graph-based anomaly detection, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug 2003, pp. 631–636.
- [Nd11] Mariá C.V. Nascimento, André C.P.L.F. de Carvalho, Spectral methods for graph clustering – a survey, *European Journal of Operational Research* 211 (2) (2011) 221–231.
- [Nes83] Yurii E. Nesterov, A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ , *Doklady Akademii Nauk SSSR* 269 (1983) 543–547.
- [NH94] R. Ng, J. Han, Efficient and effective clustering method for spatial data mining, in: Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94), Santiago, Chile, Sept. 1994, pp. 144–155.
- [NH10] Vinod Nair, Geoffrey E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proc. 2010 Int. Conf. Machine Learning (ICML'10), Haifa, Israel, June 2010, pp. 807–814.
- [NH16] Yoshihiro Nakamura, Osamu Hasegawa, Nonparametric density estimation based on self-organizing incremental neural network for large noisy data, *IEEE Transactions on Neural Networks and Learning Systems* 28 (1) (2016) 8–17.
- [NHM<sup>+</sup>18] Preetam Nandy, Alain Hauser, Marloes H. Maathuis, et al., High-dimensional consistency in score-based and hybrid structure learning, *The Annals of Statistics* 46 (6A) (2018) 3151–3183.
- [NHW<sup>+</sup>11] Eric W.T. Ngai, Yong Hu, Yiu Hing Wong, Yijun Chen, Xin Sun, The application of data mining techniques in financial fraud detection: a classification framework and an academic review of literature, *Decision Support Systems* 50 (3) (2011) 559–569.
- [NJ02] Andrew Y. Ng, Michael I. Jordan, On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes, in: *Advances in Neural Information Processing Systems*, 2002, pp. 841–848.
- [NJW01] A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: analysis and an algorithm, in: Proc. 2001 Conf. Neural Information Processing Systems (NIP'01), Vancouver, BC, Canada, Dec. 2001, pp. 849–856.
- [NK17] Maximilian Nickel, Douwe Kiela, Poincaré embeddings for learning hierarchical representations, in: Proc. 2017 Conf. Neural Information Processing Systems (NIP'17), Long Beach, CA, USA, Dec. 2017, pp. 6338–6347.
- [NKNW96] J. Neter, M.H. Kutner, C.J. Nachtsheim, L. Wasserman, *Applied Linear Statistical Models*, 4th ed., Irwin, 1996.



- [NLHP98] R. Ng, L.V.S. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, in: Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA, USA, June 1998, pp. 13–24.
- [Nov63] Albert B. Novikoff, On convergence proofs for perceptrons, Technical report, STANFORD RESEARCH INST MENLO PARK CA, 1963.
- [NTFZ14] Jingchao Ni, Hanghang Tong, Wei Fan, Xiang Zhang, Inside the atoms: ranking on a network of networks, in: Proc. 2014 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'14), New York, NY, USA, Aug. 2014, pp. 1356–1365.
- [NTFZ15] Jingchao Ni, Hanghang Tong, Wei Fan, Xiang Zhang, Flexible and robust multi-network clustering, in: Proc. 2015 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'15), Sydney, NSW, Australia, Aug. 2015, pp. 835–844.
- [NVL<sup>+</sup>15] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, James Martens, Adding gradient noise improves learning for very deep networks, arXiv preprint, arXiv:1511.06807, 2015.
- [NW99] J. Nocedal, S.J. Wright, Numerical Optimization, Springer Verlag, 1999.
- [NWH17] Feiping Nie, Xiaoqian Wang, Heng Huang, Multiclass capped  $\ell_p$ -norm svm for robust classifications, in: Proc. 2017 Nat. Conf. Artificial Intelligence (AAAI'17), San Francisco, CA, USA, Feb. 2017.
- [NXJ<sup>+</sup>08] Feiping Nie, Shiming Xiang, Yangqing Jia, Changshui Zhang, Shuicheng Yan, Trace ratio criterion for feature selection, in: AAAI, Vol. 2, 2008, pp. 671–676.
- [OBUM16] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, Jason H. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proc. 2016 Genetic and Evolutionary Computation Conf. (GECCO'16), Denver, CO, USA, July 2016, pp. 485–492.
- [OFG97] E. Osuna, R. Freund, F. Girosi, An improved training algorithm for support vector machines, in: Proc. 1997 IEEE Workshop Neural Networks for Signal Processing (NNSP'97), Amelia Island, FL, USA, Sept. 1997, pp. 276–285.
- [OG95] P. O'Neil, G. Graefe, Multi-table joins through bitmapped join indices, SIGMOD Record 24 (Sept. 1995) 8–11.
- [Ols03] J.E. Olson, Data Quality: The Accuracy Dimension, Morgan Kaufmann, 2003.
- [Omi03] E. Omiecinski, Alternative interest measures for mining associations, IEEE Transactions on Knowledge and Data Engineering 15 (2003) 57–69.
- [OMM<sup>+</sup>02] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, S. Guha, Streaming-data algorithms for high-quality clustering, in: Proc. 2002 Int. Conf. Data Engineering (ICDE'02), San Fransisco, CA, USA, Apr. 2002, pp. 685–696.
- [OOS17] Augustus Odena, Christopher Olah, Jonathon Shlens, Conditional image synthesis with auxiliary classifier GANs, in: Proc. 2017 Int. Conf. Machine Learning (ICML'17), Sydney, Australia, Aug. 2017, pp. 2642–2651.
- [Opp99] Alan V. Oppenheim, Discrete-Time Signal Processing, Pearson Education India, 1999.
- [OQ97] P. O'Neil, D. Quass, Improved query performance with variant indexes, in: Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97), Tucson, AZ, USA, May 1997, pp. 38–49.
- [ORS98] B. Özden, S. Ramaswamy, A. Silberschatz, Cyclic association rules, in: Proc. 1998 Int. Conf. Data Engineering (ICDE'98), Orlando, FL, USA, Feb. 1998, pp. 412–421.
- [Pag89] G. Pagallo, Learning DNF by decision trees, in: Proc. 1989 Int. Joint Conf. Artificial Intelligence (IJCAI'89), Morgan Kaufmann, 1989, pp. 639–644.
- [PARS14] Bryan Perozzi, Rami Al-Rfou, Steven Skiena, Deepwalk: online learning of social representations, in: Proc. 2014 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'14), New York, NY, USA, Aug. 2014, pp. 701–710.

- [PBKL14] Vu Pham, Théodore Bluche, Christopher Kermorvant, Jérôme Louradour, Dropout improves recurrent neural networks for handwriting recognition, in: Proc. 2014 Int. Conf. Frontiers in Handwriting Recognition (ICFHR'14), Crete, Greece, Sep. 2014, pp. 285–290.
- [PBMW98] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the web, in: Proc. 1998 Int. World Wide Web Conf. (WWW'98), Brisbane, Australia, 1998, pp. 161–172.
- [PBTL99] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, in: Proc. 1999 Int. Conf. Database Theory (ICDT'99), Jerusalem, Israel, Jan. 1999, pp. 398–416.
- [PCT<sup>+</sup>03] F. Pan, G. Cong, A.K.H. Tung, J. Yang, M. Zaki, CARPENTER: finding closed patterns in long biological datasets, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug. 2003, pp. 637–642.
- [PCY95a] J.S. Park, M.S. Chen, P.S. Yu, An effective hash-based algorithm for mining association rules, in: Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'95), San Jose, CA, USA, May 1995, pp. 175–186.
- [PCY95b] J.S. Park, M.S. Chen, P.S. Yu, Efficient parallel mining for association rules, in: Proc. 1995 Int. Conf. Information and Knowledge Management (CIKM'95), Baltimore, MD, USA, Nov. 1995, pp. 31–36.
- [PCZ<sup>+</sup>19] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, Quoc V. Le, SpecAugment: a simple data augmentation method for automatic speech recognition, in: Proc. 2019 Conf. International Speech Communication Association (ISCA'19), Graz, Austria, Sep. 2019, pp. 2613–2617.
- [Pea88] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufman, 1988.
- [PHL01] J. Pei, J. Han, L.V.S. Lakshmanan, Mining frequent itemsets with convertible constraints, in: Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, April 2001, pp. 433–442.
- [PHL04] Lance Parsons, Ehtesham Haque, Huan Liu, Subspace clustering for high dimensional data: a review, SIGKDD Explorations Newsletter 6 (1) (June 2004) 90–105.
- [PHM00] J. Pei, J. Han, R. Mao, CLOSET: an efficient algorithm for mining frequent closed itemsets, in: Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX, USA, May 2000, pp. 11–20.
- [PHMA<sup>+</sup>01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, April 2001, pp. 215–224.
- [PHMA<sup>+</sup>04] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: the PrefixSpan approach, IEEE Transactions on Knowledge and Data Engineering 16 (2004) 1424–1440.
- [PI97] V. Poosala, Y. Ioannidis, Selectivity estimation without the attribute value independence assumption, in: Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97), Athens, Greece, Aug. 1997, pp. 486–495.
- [PKGf03] S. Papadimitriou, H. Kitagawa, P.B. Gibbons, C. Faloutsos, Loci: fast outlier detection using the local correlation integral, in: Proc. 2003 Int. Conf. Data Engineering (ICDE'03), Bangalore, India, March 2003, pp. 315–326.
- [PKMT99] A. Pfeffer, D. Koller, B. Milch, K. Takusagawa, SPOOK: a system for probabilistic object-oriented knowledge representation, in: Proc. 1999 Conf. Uncertainty in Artificial Intelligence (UAI'99), Stockholm, Sweden, 1999, pp. 541–550.
- [PKZT01] D. Papadias, P. Kalnis, J. Zhang, Y. Tao, Efficient OLAP operations in spatial data warehouses, in: Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01), Redondo Beach, CA, USA, July 2001, pp. 443–459.

- [PL08] Bo Pang, Lillian Lee, Opinion mining and sentiment analysis, *Foundations and Trends in Information Retrieval* 2 (1–2) (January 2008) 1–135.
- [Pla98] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schoelkopf, C.J.C. Burges, A. Smola (Eds.), *Advances in Kernel Methods—Support Vector Learning*, MIT Press, 1998, pp. 185–208.
- [PLC<sup>+</sup>20] Canelle Poirier, Dianbo Liu, Leonardo Clemente, Xiyu Ding, Matteo Chinazzi, Jessica Davis, Alessandro Vespignani, Mauricio Santillana, Real-time forecasting of the Covid-19 outbreak in Chinese provinces: machine learning approach using novel digital data and estimates from mechanistic models, *Journal of Medical Internet Research* 22 (8) (2020) e20285.
- [PM11] Jia Pan, Dinesh Manocha, Fast gpu-based locality sensitive hashing for k-nearest neighbor computation, in: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2011, pp. 211–220.
- [PM12] Jia Pan, Dinesh Manocha, Bi-level locality sensitive hashing for k-nearest neighbor computation, in: *2012 IEEE 28th International Conference on Data Engineering*, IEEE, 2012, pp. 378–389.
- [PMSH16] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, Richard Harang, Crafting adversarial input sequences for recurrent neural networks, in: *Proc. 2016 Military Communications Conf. (MILCOM'16)*, Baltimore, MD, USA, Nov. 2016, pp. 49–54.
- [PMW<sup>+</sup>16] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, Ananthram Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: *Proc. 2016 Symp. Security and Privacy (SP'16)*, San Jose, CA, USA, May 2016, pp. 582–597.
- [PNI<sup>+</sup>18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer, Deep contextualized word representations, in: *Proc. 2018 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'18)*, New Orleans, LA, USA, June 2018, pp. 2227–2237.
- [PP07] A. Patcha, J.-M. Park, An overview of anomaly detection techniques: existing solutions and latest technological trends, *Computer Networks* 51 (2007).
- [PPHM09] Mark M. Palatucci, Dean A. Pomerleau, Geoffrey E. Hinton, Tom Mitchell, Zero-shot learning with semantic output codes, in: *Proc. 2009 Conf. Neural Information Processing Systems (NIPS'09)*, Vancouver, BC, Canada, Dec. 2009.
- [Pre98] Lutz Prechelt, Early stopping-but when?, in: *Neural Networks: Tricks of the Trade*, Springer, 1998, pp. 55–69.
- [PS85] F.P. Preparata, M.I. Shamos, *Computational Geometry: An Introduction*, Springer Verlag, 1985.
- [PS91] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules, in: *Proc. 1991 Workshop on Knowledge Discovery in Databases (KDD'91)*, July 1991, pp. 229–248.
- [PSCH20] Guansong Pang, Chunhua Shen, Longbing Cao, Anton van den Hengel, Deep learning for anomaly detection: a review, *arXiv preprint*, arXiv:2007.02500, 2020.
- [PSF91] G. Piatetsky-Shapiro, W.J. Frawley, *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991.
- [PSF05] Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, Streaming pattern discovery in multiple time-series, in: *Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05)*, Trondheim, Norway, Sep. 2005, pp. 697–708.
- [PSM14] Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: global vectors for word representation, in: *Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP'14)*, Doha, Qatar, Oct. 2014, pp. 1532–1543.
- [PT94] Pentti Paatero, Unto Tapper, Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values, *EnvironMetrics* 5 (2) (1994) 111–126.
- [PTCX04] F. Pan, A.K.H. Tung, G. Cong, X. Xu, COBBLER: combining column and row enumeration for closed pattern discovery, in: *Proc. 2004 Int. Conf. Scientific and Statistical Database Management (SSDBM'04)*, Santorini Island, Greece, June 2004, pp. 21–30.

- [PY10] S.J. Pan, Q. Yang, A survey on transfer learning, *IEEE Transactions on Knowledge and Data Engineering* 22 (2010) 1345–1359.
- [Pyl99] D. Pyle, *Data Preparation for Data Mining*, Morgan Kaufmann, 1999.
- [PZC<sup>+</sup>03] J. Pei, X. Zhang, M. Cho, H. Wang, P.S. Yu, Maple: a fast algorithm for maximal pattern-based clustering, in: *Proc. 2003 Int. Conf. Data Mining (ICDM'03)*, Melbourne, FL, USA, Dec. 2003, pp. 259–266.
- [QJCJ93] J.R. Quinlan, R.M. Cameron-Jones, FOIL: a midterm report, in: *Proc. 1993 European Conf. Machine Learning (ECML'93)*, Vienna, Austria, Apr. 1993, pp. 3–20.
- [Qia99] Ning Qian, On the momentum term in gradient descent learning algorithms, *Neural Networks* 12 (1) (1999) 145–151.
- [QR89] J.R. Quinlan, R.L. Rivest, Inferring decision trees using the minimum description length principle, *Information and Computation* 80 (Mar. 1989) 227–248.
- [Qui86] J.R. Quinlan, Induction of decision trees, *Machine Learning* 1 (1986) 81–106.
- [Qui87] J.R. Quinlan, Simplifying decision trees, *International Journal of Man-Machine Studies* 27 (1987) 221–234.
- [Qui88] J.R. Quinlan, An empirical comparison of genetic and decision-tree classifiers, in: *Proc. 1988 Int. Conf. Machine Learning (ICML'88)*, Ann Arbor, MI, USA, June 1988, pp. 135–141.
- [Qui89] J.R. Quinlan, Unknown attribute values in induction, in: *Proc. 1989 Int. Conf. Machine Learning (ICML'89)*, Ithaca, NY, USA, June 1989, pp. 164–168.
- [Qui90] J.R. Quinlan, Learning logic definitions from relations, *Machine Learning* 5 (1990) 139–166.
- [Qui93] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
- [Qui96] J.R. Quinlan, Bagging, boosting, and C4.5, in: *Proc. 1996 Nat. Conf. Artificial Intelligence (AAAI'96)*, Vol. 1, Portland, OR, USA, Aug. 1996, pp. 725–730.
- [RA87] E.L. Rissland, K. Ashley, HYPO: a case-based system for trade secret law, in: *Proc. 1st Int. Conf. Artificial Intelligence and Law*, Boston, MA, USA, May 1987, pp. 60–66.
- [RA15] Shebuti Rayana, Leman Akoglu, Collective opinion spam detection: bridging review networks and metadata, in: *Proc. 2015 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'15)*, Sydney, NSW, Australia, Aug. 2015, pp. 985–994.
- [RAHL19] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V. Le, Regularized evolution for image classifier architecture search, in: *Proc. 2019 AAAI Conf. Artificial Intelligence (AAAI'19)*, Honolulu, HI, USA, Jan. 2019, pp. 4780–4789.
- [RAY<sup>+</sup>16] Scott Reed, Zeynep Akata, Xinchun Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee, Generative adversarial text to image synthesis, in: *Proc. 2016 Int. Conf. Machine Learning (ICML'16)*, New York City, NY, USA, June 2016, pp. 1060–1069.
- [RBE<sup>+</sup>17] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, Christopher Ré, Snorkel: rapid training data creation with weak supervision, in: *Proc. 2017 Int. Conf. Very Large Data Bases (VLDB'17)*, 2017, pp. 269–282.
- [RBKK95] S. Russell, J. Binder, D. Koller, K. Kanazawa, Local learning in probabilistic networks with hidden variables, in: *Proc. 1995 Joint Int. Conf. Artificial Intelligence (IJCAI'95)*, Montreal, Canada, Aug. 1995, pp. 1146–1152.
- [RBM<sup>+</sup>14] Naren Ramakrishnan, Patrick Butler, Sathappan Muthiah, Nathan Self, Rupinder Khandpur, Parang Saraf, Wei Wang, Jose Cadena, Anil Vullikanti, Gizem Korkmaz, et al., 'Beating the news' with embers: forecasting civil unrest using open source indicators, in: *Proc. 2014 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'14)*, New York, NY, USA, Aug. 2014, pp. 1799–1808.
- [RC07] R. Ramakrishnan, B.-C. Chen, Exploratory mining in cube space, *Data Mining and Knowledge Discovery* 15 (2007) 29–54.
- [RCY11] Karl Rohe, Sourav Chatterjee, Bin Yu, Spectral clustering and the high-dimensional stochastic blockmodel, *The Annals of Statistics* 39 (4) (2011) 1878–1915.

- [Red01] T. Redman, *Data Quality: The Field Guide*, Digital Press (Elsevier), 2001.
- [REKW<sup>+</sup>15] Xiang Ren, Ahmed El-Kishky, Chi Wang, Fangbo Tao, Clare R. Voss, Heng Ji, Jiawei Han, ClusType: effective entity recognition and typing by relation phrase-based clustering, in: Proc. 2015 Int. Conf. Knowledge Discovery and Data Mining (KDD'15), Sydney, NSW, Australia, Aug. 2015, pp. 995–1004.
- [RH01] V. Raman, J.M. Hellerstein, Potter's wheel: an interactive data cleaning system, in: Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01), Rome, Italy, Sept. 2001, pp. 381–390.
- [RH07] A. Rosenberg, J. Hirschberg, V-measure: a conditional entropy-based external cluster evaluation measure, in: Proc. 2007 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07), Prague, Czech Republic, June 2007, pp. 410–420.
- [RH18] Xiang Ren, Jiawei Han, *Mining Structures of Factual Knowledge from Text: An Effort-Light Approach*, Morgan & Claypool, 2018.
- [RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Faster r-cnn: towards real-time object detection with region proposal networks, in: Proc. 2015 Conf. Neural Information Processing Systems (NIPS'15), Montreal, QC, Canada, Dec. 2015, pp. 91–99.
- [RHW86a] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland (Eds.), *Parallel Distributed Processing*, MIT Press, 1986.
- [RHW86b] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, Learning representations by back-propagating errors, *Nature* 323 (6088) (1986) 533–536.
- [Rip96] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.
- [RM51] Herbert Robbins, Sutton Monro, A stochastic approximation method, *The Annals of Mathematical Statistics* (1951) 400–407.
- [RM86] D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.
- [RMC15] Alec Radford, Luke Metz, Soumith Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint, arXiv:1511.06434, 2015.
- [RMC16] Alec Radford, Luke Metz, Soumith Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, in: Proc. 2016 Int. Conf. on Learning Representations (ICLR'16), San Juan, Puerto Rico, May 2016.
- [RMKD05] Michael T. Rosenstein, Zvika Marx, Leslie Pack Kaelbling, Thomas G. Dietterich, To transfer or not to transfer, in: Proc. 2005 Conf. Neural Information Processing Systems (NIPS'05), 2005, pp. 1–4.
- [RMS98] S. Ramaswamy, S. Mahajan, A. Silberschatz, On the discovery of interesting patterns in association rules, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 368–379.
- [RMS<sup>+</sup>17] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, Alexey Kurakin, Large-scale evolution of image classifiers, in: Proc. 2017 Int. Conf. Machine Learning (ICML'17), Sydney, NSW, Australia, Aug. 2017, pp. 2902–2911.
- [RN95] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 1995.
- [RNI09] M. Radovanović, A. Nanopoulos, M. Ivanović, Nearest neighbors in high-dimensional data: the emergence and influence of hubs, in: Proc. 2009 Int. Conf. Machine Learning (ICML'09), Montreal, QC, Canada, June 2009, pp. 865–872.
- [RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, *Improving Language Understanding by Generative Pre-Training*, 2018.
- [Roh73] F. Rohlf, Algorithm 76. Hierarchical clustering using the minimum spanning tree, *Computer Journal* 16 (01 1973) 93–95.
- [Ros58] Frank Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review* 65 (6) (1958) 386.

- [RPT15] Bernardino Romera-Paredes, Philip Torr, An embarrassingly simple approach to zero-shot learning, in: Proc. 2015 Int. Conf. Machine Learning (ICML'15), Lille, France, July 2015, pp. 2152–2161.
- [RS89] C. Riesbeck, R. Schank, Inside Case-Based Reasoning, Lawrence Erlbaum, 1989.
- [RS97] K. Ross, D. Srivastava, Fast computation of sparse datacubes, in: Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97), Athens, Greece, Aug. 1997, pp. 116–125.
- [RS98] R. Rastogi, K. Shim, Public: a decision tree classifier that integrates building and pruning, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 404–415.
- [RSC98] K.A. Ross, D. Srivastava, D. Chatziantoniou, Complex aggregation at multiple granularities, in: Proc. 1998 Int. Conf. of Extending Database Technology (EDBT'98), Valencia, Spain, Mar. 1998, pp. 263–277.
- [RSD<sup>+</sup>17] Raghu Ramakrishnan, Baskar Sridharan, John R. Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mítica Manu, Spiro Michaylov, Rogério Ramos, Neil Sharman, Zee Xu, Youssef Barakat, Chris Douglas, Richard Draves, Shrikant S. Naidu, Shankar Shastry, Atul Sikaria, Simon Sun, Ramarathnam Venkatesan, Azure data lake store: a hyperscale distributed file service for big data analytics, in: Proc. 2017 ACM Int. Conf. Management of Data (SIGMOD'17), New York, NY, USA, May 2017, pp. 51–63.
- [RSG16] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, “Why should I trust you?” explaining the predictions of any classifier, in: Proc. 2016 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'16), San Francisco, CA, USA, Aug. 2016, pp. 1135–1144.
- [RWL10] Pradeep Ravikumar, Martin J. Wainwright, John D. Lafferty, High-dimensional Ising model selection using  $\ell_1$ -regularized logistic regression, *The Annals of Statistics* 38 (3) (2010) 1287–1319.
- [RYZ<sup>+</sup>10] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, Linda Moy, Learning from crowds, *Journal of Machine Learning Research* 11 (43) (2010) 1297–1322.
- [RZL17] Prajit Ramachandran, Barret Zoph, Quoc V. Le, Searching for activation functions, arXiv preprint, arXiv:1710.05941, 2017.
- [RZQL20] Kui Ren, Tianhang Zheng, Zhan Qin, Xue Liu, Adversarial attacks and defenses in deep learning, *Engineering* 6 (3) (2020) 346–360.
- [SA95] R. Srikant, R. Agrawal, Mining generalized association rules, in: Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95), Zurich, Switzerland, Sept. 1995, pp. 407–419.
- [SA96] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proc. 5th Int. Conf. Extending Database Technology (EDBT'96), Avignon, France, Mar. 1996, pp. 3–17.
- [SAM96] J. Shafer, R. Agrawal, M. Mehta, SPRINT: a scalable parallel classifier for data mining, in: Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96), Bombay, India, Sept. 1996, pp. 544–555.
- [SAM98] S. Sarawagi, R. Agrawal, N. Megiddo, Discovery-driven exploration of OLAP data cubes, in: Proc. 1998 Int. Conf. of Extending Database Technology (EDBT'98), Valencia, Spain, Mar. 1998, pp. 168–182.
- [San89] Terence D. Sanger, Optimal unsupervised learning in a single-layer linear feedforward neural network, *Neural Networks* 2 (6) (1989) 459–473.
- [SB18] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
- [SBMU98] C. Silverstein, S. Brin, R. Motwani, J.D. Ullman, Scalable techniques for mining causal structures, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 594–605.
- [SBSW98] B. Schölkopf, P.L. Bartlett, A. Smola, R. Williamson, Shrinking the tube: a new support vector regression algorithm, in: Proc. 1998 Conf. Neural Information Processing Systems (NIPS'98), Denver, CO, USA, Dec. 1998, pp. 330–336.

- [Sch86] J.C. Schlimmer, Learning and representation change, in: Proc. 1986 Nat. Conf. Artificial Intelligence (AAAI'86), Philadelphia, PA, USA, 1986, pp. 511–515.
- [Sch07] S.E. Schaeffer, Graph clustering, *Computer Science Review* 1 (2007) 27–64.
- [SCST17] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, Ameet S. Talwalkar, Federated multi-task learning, in: Proc. 2017 Conf. Neural Information Processing Systems (NIPS'17), Long Beach, CA, USA, Dec. 2017, pp. 4424–4434.
- [SCZ98] G. Sheikholeslami, S. Chatterjee, A. Zhang, WaveCluster: a multi-resolution clustering approach for very large spatial databases, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 428–439.
- [SDJL96] D. Srivastava, S. Dar, H.V. Jagadish, A.V. Levy, Answering queries with aggregation using views, in: Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96), Bombay, India, Sept. 1996, pp. 318–329.
- [SDN98] A. Shukla, P.M. Deshpande, J.F. Naughton, Materialized view selection for multidimensional datasets, in: Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98), New York, NY, USA, Aug. 1998, pp. 488–499.
- [SDNT19] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, Jian Tang, Rotate: knowledge graph embedding by relational rotation in complex space, arXiv preprint, arXiv:1902.10197, 2019.
- [SDRK02] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, Y. Kotidis, Dwarf: shrinking the petacube, in: Proc. 2002 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'02), Madison, WI, USA, June 2002, pp. 464–475.
- [SDVB18] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, Xavier Bresson, Structured sequence modeling with graph convolutional recurrent networks, in: Proc. 2018 Int. Conf. on Neural Information Processing (ICONIP'18), Siem Reap, Cambodia, Dec. 2018, pp. 362–373.
- [SE10] G. Seni, J.F. Elder, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, Morgan and Claypool, 2010.
- [Set10] B. Settles, *Active learning literature survey*, Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2010.
- [SF86] J.C. Schlimmer, D. Fisher, A case study of incremental concept induction, in: Proc. 1986 Nat. Conf. Artificial Intelligence (AAAI'86), Philadelphia, PA, USA, 1986, pp. 496–501.
- [SFB99] J. Shanmugasundaram, U.M. Fayyad, P.S. Bradley, Compressed data cubes for OLAP aggregate query approximation on continuous dimensions, in: Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99), San Diego, CA, USA, Aug. 1999, pp. 223–232.
- [SG92] P. Smyth, R.M. Goodman, An information theoretic approach to rule induction, *IEEE Transactions on Knowledge and Data Engineering* 4 (1992) 301–316.
- [SGSH00] Peter Spirtes, Clark N. Glymour, Richard Scheines, David Heckerman, *Causation, Prediction, and Search*, MIT Press, 2000.
- [SGT<sup>+</sup>08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, Gabriele Monfardini, The graph neural network model, *IEEE Transactions on Neural Networks* 20 (1) (2008) 61–80.
- [She31] W.A. Shewhart, *Economic Control of Quality of Manufactured Product*, D. van Nostrand Company, 1931.
- [SHH<sup>+</sup>06] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, Antti Kerminen, Michael Jordan, A linear non-Gaussian acyclic model for causal discovery, *Journal of Machine Learning Research* 7 (10) (2006).
- [Shi99] Y.-S. Shih, Families of splitting criteria for classification trees, *Statistics and Computing* 9 (1999) 309–315.
- [SHK00] N. Stefanovic, J. Han, K. Koperski, Object-based selective materialization for efficient implementation of spatial data cubes, *IEEE Transactions on Knowledge and Data Engineering* 12 (2000) 938–958.

- [SHK<sup>+</sup>14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research* 15 (1) (2014) 1929–1958.
- [Sho97] A. Shoshani, OLAP and statistical databases: similarities and differences, in: *Proc. 1997 ACM Symp. Principles of Database Systems(PODS’97)*, Tucson, AZ, USA, May 1997, pp. 185–196.
- [SHX04] Z. Shao, J. Han, D. Xin, MM-Cubing: computing iceberg cubes by factorizing the lattice space, in: *Proc. 2004 Int. Conf. on Scientific and Statistical Database Management (SSDBM’04)*, Santorini Island, Greece, June 2004, pp. 213–222.
- [SHY<sup>+</sup>11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, Tianyi Wu, Pathsim: meta path-based top-k similarity search in heterogeneous information networks, *Proceedings of the VLDB Endowment* 4 (11) (2011) 992–1003.
- [SHZQ18] Zequn Sun, Wei Hu, Qingheng Zhang, Yuzhong Qu, Bootstrapping entity alignment with knowledge graph embedding, in: *Proc. 2018 Joint Conf. Artificial Intelligence (IJCAI’18)*, Stockholm, Sweden, July 2018, pp. 4396–4402.
- [Sil18] Bernard W. Silverman, *Density Estimation for Statistics and Data Analysis*, Routledge, 2018.
- [SJ91] Simon J. Sheather, Michael C. Jones, A reliable data-based bandwidth selection method for kernel density estimation, *Journal of the Royal Statistical Society, Series B, Methodological* 53 (3) (1991) 683–690.
- [SJ03] Catherine A. Sugar, Gareth M. James, Finding the number of clusters in a dataset: an information-theoretic approach, *Journal of the American Statistical Association* 98 (January 2003) 750–763.
- [SJS17] Uri Shalit, Fredrik D. Johansson, David Sontag, Estimating individual treatment effect: generalization bounds and algorithms, in: *Proc. 2017 Int. Conf. Machine Learning (ICML’17)*, Sydney, NSW, Australia, Aug. 2017, pp. 3076–3085.
- [SK08] J. Shieh, E. Keogh, iSAX: indexing and mining terabyte sized time series, in: *Proc. 2008 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD’08)*, Las Vegas, NV, USA, Aug. 2008.
- [SK18] Martin Simonovsky, Nikos Komodakis, Graphvae: towards generation of small graphs using variational autoencoders, in: *Proc. 2018 Int. Conf. Artificial Neural Networks (ICANN’18)*, Rhodes, Greece, Oct. 2018, pp. 412–422.
- [SK19] Connor Shorten, Taghi M. Khoshgoftaar, A survey on image data augmentation for deep learning, *Journal of Big Data* 6 (2019) 60.
- [SKH<sup>+</sup>14] Chuan Shi, Xiangnan Kong, Yue Huang, S. Yu Philip, Bin Wu, Hetesim: a general framework for relevance measure in heterogeneous networks, *IEEE Transactions on Knowledge and Data Engineering* 26 (10) (2014) 2479–2492.
- [SKP15] Florian Schroff, Dmitry Kalenichenko, James Philbin, Facenet: a unified embedding for face recognition and clustering, in: *Proc. 2015 Conf. Computer Vision and Pattern Recognition (CVPR’15)*, Boston, MA, USA, June 2015, pp. 815–823.
- [SL19] Kai Shu, Huan Liu, *Detecting Fake News on Social Media. Synthesis Lectures on Data Mining and Knowledge Discovery*, Morgan & Claypool Publishers, 2019.
- [SLA12] Jasper Snoek, Hugo Larochelle, Ryan P. Adams, Practical bayesian optimization of machine learning algorithms, *arXiv preprint, arXiv:1206.2944*, 2012.
- [SLJ<sup>+</sup>15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going deeper with convolutions, in: *Proc. 2015 Conf. Computer Vision and Pattern Recognition (CVPR’15)*, Boston, MA, USA, June 2015, pp. 1–9.
- [SLJ<sup>+</sup>18] Jingbo Shang, Jialu Liu, Meng Jiang, Xiang Ren, Clare R. Voss, Jiawei Han, Automated phrase mining from massive text corpora, *IEEE Transactions on Knowledge and Data Engineering* 30 (10) (2018) 1825–1837.



- [SLT<sup>+</sup>01] S. Shekhar, C.-T. Lu, X. Tan, S. Chawla, R.R. Vatsavai, Map cube: a visualization tool for spatial data warehouses, in: H.J. Miller, J. Han (Eds.), *Geographic Data Mining and Knowledge Discovery*, Taylor and Francis, 2001, pp. 73–108.
- [SMG13] Andrew M. Saxe, James L. McClelland, Surya Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, arXiv preprint, arXiv:1312.6120, 2013.
- [SMS15] Nitish Srivastava, Elman Mansimov, Ruslan Salakhudinov, Unsupervised learning of video representations using lstms, in: *Proc. 2015 Int. Conf. Machine Learning (ICML'15)*, Lille, France, July 2015, pp. 843–852.
- [SMT91] J.W. Shavlik, R.J. Mooney, G.G. Towell, Symbolic and neural learning algorithms: an experimental comparison, *Machine Learning* 6 (1991) 111–144.
- [SNF<sup>+</sup>13] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, Pierre Vandergheynst, The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains, *IEEE Signal Processing Magazine* 30 (3) (2013) 83–98.
- [SNH<sup>+</sup>13] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S. Yu, Xiao Yu, Pathselclus: integrating meta-path selection with user-guided object clustering in heterogeneous information networks, *ACM Transactions on Knowledge Discovery from Data* 7 (3) (2013) 1–23.
- [SOMZ96] W. Shen, K. Ong, B. Mitbender, C. Zaniolo, Metaqueries for data mining, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996, pp. 375–398.
- [SON95] A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, in: *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, Zurich, Switzerland, Sept. 1995, pp. 432–443.
- [SON98] A. Savasere, E. Omiecinski, S. Navathe, Mining for strong negative associations in a large database of customer transactions, in: *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, Orlando, FL, USA, Feb. 1998, pp. 494–502.
- [SP97] Mike Schuster, Kuldip K. Paliwal, Bidirectional recurrent neural networks, *IEEE Transactions on Signal Processing* 45 (11) (1997) 2673–2681.
- [SQM<sup>+</sup>21] Jiaming Shen, Wenda Qiu, Yu Meng, Jingbo Shang, Xiang Ren, Jiawei Han, TaxoClass: hierarchical multi-label text classification using only class names, in: *Proc. 2021 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'21)*, 2021, pp. 4239–4249.
- [SQW06] James G. Shanahan, Yan Qu, Janyce Wiebe (Eds.), *Computing Attitude and Affect in Text: Theory and Applications*, The Information Retrieval Series, vol. 20, Springer, 2006.
- [SR81] R. Sokal, F. Rohlf, *Biometry*, Freeman, 1981.
- [SS88] W. Siedlecki, J. Sklansky, On automatic feature selection, *International Journal of Pattern Recognition and Artificial Intelligence* 2 (1988) 197–220.
- [SS94] S. Sarawagi, M. Stonebraker, Efficient organization of large multidimensional arrays, in: *Proc. 1994 Int. Conf. Data Engineering (ICDE'94)*, Houston, TX, USA, Feb. 1994, pp. 328–336.
- [SS98] Pierangela Samarati, Latanya Sweeney, Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement through Generalization and Suppression, Technical report, SRI International, 1998.
- [SS01] G. Sathe, S. Sarawagi, Intelligent rollups in multidimensional OLAP data, in: *Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01)*, Rome, Italy, Sept. 2001, pp. 531–540.
- [SSB<sup>+</sup>12] Michael Schmitz, Stephen Soderland, Robert Bart, Oren Etzioni, et al., Open language learning for information extraction, in: *Proc. 2012 Joint Conf. Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'12)*, Jeju Island, Korea, July 2012, pp. 523–534.
- [SSSSC11] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, Andrew Cotter, Pegasos: primal estimated sub-gradient solver for svm, *Mathematical Programming* 127 (1) (2011) 3–30.

- [SSVL<sup>+</sup>11] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, Karsten M. Borgwardt, Weisfeiler-Lehman graph kernels, *Journal of Machine Learning Research* 12 (9) (2011).
- [SSX<sup>+</sup>20] Jiaming Shen, Zhihong Shen, Chenyan Xiong, Chunxin Wang, Kuansan Wang, Jiawei Han, TaxoExpan: self-supervised taxonomy expansion with position-enhanced graph neural network, in: *Proc. 2020 the Web Conf. (WWW'20)*, Apr. 2020, pp. 486–497.
- [ST96] A. Silberschatz, A. Tuzhilin, What makes patterns interesting in knowledge discovery systems, *IEEE Transactions on Knowledge and Data Engineering* 8 (Dec. 1996) 970–974.
- [ST10] Nambiraj Suguna, Keppana Thanushkodi, An improved k-nearest neighbor classification using genetic algorithm, *International Journal of Computer Science Issues* 7 (2) (2010) 18–21.
- [STA98] S. Sarawagi, S. Thomas, R. Agrawal, Integrating association rule mining with relational database systems: alternatives and implications, in: *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, Seattle, WA, USA, June 1998, pp. 343–354.
- [Ste72] W. Stefansky, Rejecting outliers in factorial designs, *Technometrics* 14 (1972) 469–479.
- [Sto74] M. Stone, Cross-validatory choice and assessment of statistical predictions, *Journal of the Royal Statistical Society* 36 (1974) 111–147.
- [STPH16] Jingbo Shang, Wenzhu Tong, Jian Peng, Jiawei Han, Dpclass: an effective but concise discriminative patterns-based classification framework, in: *Proc. 2016 SIAM Int. Conf. Data Mining (SDM'16)*, Miami, FL, USA, May 2016, pp. 567–575.
- [STR<sup>+</sup>18] Hoo-Chang Shin, Neil A. Tenenholtz, Jameson K. Rogers, Christopher G. Schwarz, Matthew L. Senjem, Jeffrey L. Gunter, Katherine P. Andriole, Mark Michalski, Medical image synthesis for data augmentation and anonymization using generative adversarial networks, in: *Proc. 2018 Conf. Simulation and Synthesis in Medical Imaging (SASHIMI'18)*, Granada, Spain, Sep. 2018, pp. 1–11.
- [Str93] Gilbert Strang, *Introduction to Linear Algebra*, Vol. 3, Wellesley-Cambridge Press, Wellesley, MA, USA, 1993.
- [Str19] Gilbert Strang, *Linear Algebra and Learning from Data*, Wellesley-Cambridge Press, Cambridge, 2019.
- [SVA97] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, CA, USA, Aug. 1997, pp. 67–73.
- [SVC08] Shashi Shekhar, Ranga Vatsavai, Mete Celik, Spatial and spatiotemporal data mining: recent advances, in: *Next Generation of Data Mining*, 2008.
- [SVL14] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to sequence learning with neural networks, in: *Proc. 2014 Conf. Neural Information Processing Systems (NIPS'14)*, Montreal, QC, Canada, Dec. 2014, pp. 3104–3112.
- [SW49] C.E. Shannon, W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949.
- [SW17] Baoxu Shi, Tim Weninger, Proje: embedding projection for knowledge graph completion, in: *Proc. 2017 AAAI Conf. Artificial Intelligence (AAAI'17)*, San Francisco, CA, USA, Feb. 2017, pp. 1236–1242.
- [Swe88] J. Swets, Measuring the accuracy of diagnostic systems, *Science* 240 (1988) 1285–1293.
- [SWF20] Zheyuan Ryan Shi, Claire Wang, Fei Fang, Artificial intelligence for social good: a survey, *arXiv preprint*, arXiv:2001.01818, 2020.
- [SWJR07] X. Song, M. Wu, C. Jermaine, S. Ranka, Conditional anomaly detection, *IEEE Transactions on Knowledge and Data Engineering* 19 (2007).
- [SWL<sup>+</sup>18] Jiaming Shen, Zeqiu Wu, Dongming Lei, Chao Zhang, Xiang Ren, Michelle T. Vanni, Brian M. Sadler, Jiawei Han, HiExpan: task-guided taxonomy construction by hierarchical tree expansion, in: *Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18)*, London, UK, Aug. 2018, pp. 2180–2189.

- [SWM17] Wojciech Samek, Thomas Wiegand, Klaus-Robert Müller, Explainable artificial intelligence: understanding, visualizing and interpreting deep learning models, arXiv preprint, arXiv:1708.08296, 2017.
- [SWS17] Dinggang Shen, Guorong Wu, Heung-Il Suk, Deep learning in medical image analysis, *Annual Review of Biomedical Engineering* 19 (2017) 221–248.
- [SWY75] Gerard Salton, Anita Wong, Chung-Shu Yang, A vector space model for automatic indexing, *Communications of the ACM* 18 (11) (1975) 613–620.
- [SWY19] Siyu Shao, Pu Wang, Ruqiang Yan, Generative adversarial networks for data augmentation in machine fault diagnosis, *Computers in Industry* 106 (2019) 85–93.
- [SZ14] Karen Simonyan, Andrew Zisserman, Two-stream convolutional networks for action recognition in videos, in: *Proc. 2014 Conf. Neural Information Processing Systems (NIPS'14)*, Montreal, QC, Canada, Dec. 2014, pp. 568–576.
- [SZ15] Karen Simonyan, Andrew Zisserman, Very deep convolutional networks for large-scale image recognition, in: *Proc. 2015 Int. Conf. on Learning Representations (ICLR'15)*, San Diego, CA, USA, May 2015.
- [SZS<sup>+</sup>13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, Intriguing properties of neural networks, arXiv preprint, arXiv:1312.6199, 2013.
- [TBLJ13] Ferdian Thung, Tegawend Bissyand, David Lo, Lingxiao Jiang, Network structure of social coding in github, in: *Proc. 2013 Euromicro Conf. Software Maintenance and Reengineering (CSMR'13)*, March 2013, pp. 323–326.
- [TC01] Simon Tong, Edward Y. Chang, Support vector machine active learning for image retrieval, in: *Proc. 2001 Int. Conf. Multimedia (Multimedia'01)*, Ottawa, Ontario, Canada, Oct. 2001, pp. 107–118.
- [TD02] D.M.J. Tax, R.P.W. Duin, Using two-class classifiers for multiclass classification, in: *Proc. 2002 Int. Conf. Pattern Recognition (ICPR'02)*, Quebec, Canada, Aug. 2002, pp. 124–127.
- [TFP06] Hanghang Tong, Christos Faloutsos, Jia-Yu Pan, Fast random walk with restart and its applications, in: *Proc. 2006 Int. Conf. Data Mining (ICDM'06)*, Hong Kong, China, Dec. 2006, pp. 613–622.
- [Tib96] Robert Tibshirani, Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society, Series B, Methodological* 58 (1) (1996) 267–288.
- [Tib11] Robert Tibshirani, Regression shrinkage and selection via the lasso: a retrospective, *Journal of the Royal Statistical Society, Series B, Statistical Methodology* 73 (3) (2011) 273–282.
- [TK08] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2008.
- [TKS02] P.-N. Tan, V. Kumar, J. Srivastava, Selecting the right interestingness measure for association patterns, in: *Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02)*, Edmonton, Canada, July 2002, pp. 32–41.
- [TL11] Hanghang Tong, Ching-Yung Lin, Non-negative residual matrix factorization with application to graph anomaly detection, in: *Proc. 2011 SIAM Int. Conf. Data Mining (SDM'11)*, Mesa, AZ, USA, Apr. 2011, pp. 143–153.
- [Toi96] H. Toivonen, Sampling large databases for association rules, in: *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, Bombay, India, Sept. 1996, pp. 134–145.
- [TPS<sup>+</sup>08] Hanghang Tong, Spiros Papadimitriou, Jimeng Sun, Philip S. Yu, Christos Faloutsos, Colibri: fast mining of large static and dynamic graphs, in: *Proc. 2008 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'08)*, Las Vegas, NV, USA, Aug. 2008, pp. 686–694.
- [TQW<sup>+</sup>15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, Qiaozhu Mei, Line: large-scale information network embedding, in: *Proc. 2015 the Web Conf. (WWW'15)*, Florence, Italy, May 2015, pp. 1067–1077.

- [TS14] Alexander Toshev, Christian Szegedy, Deeppose: human pose estimation via deep neural networks, in: Proc. 2014 Conf. Computer Vision and Pattern Recognition (CVPR'14), Columbus, OH, USA, June 2014, pp. 1653–1660.
- [TSK05] P.N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison Wesley, 2005.
- [TSKK18] P.N. Tan, M. Steinbach, A. Karpatne, V. Kumar, Introduction to Data Mining, 2nd edition, Pearson, 2018.
- [TSR<sup>+</sup>05] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, Keith Knight, Sparsity and smoothness via the fused lasso, *Journal of the Royal Statistical Society, Series B, Statistical Methodology* 67 (1) (2005) 91–108.
- [TSS04] A. Tanay, R. Sharan, R. Shamir, Biclustering algorithms: a survey, in: *Handbook of Computational Molecular Biology*, Chapman & Hall, 2004, pp. 26:1–26:17.
- [TSX15] Youze Tang, Yanchen Shi, Xiaokui Xiao, Influence maximization in near-linear time: a martingale approach, in: Proc. 2015 ACM SIGMOD Int. Conf. Management of Data (MOD'15), Melbourne, Victoria, Australia, June 2015, pp. 1539–1554.
- [TTV16] Pedro Tabacof, Julia Tavares, Eduardo Valle, Adversarial images for variational autoencoders, arXiv preprint, arXiv:1612.00155, 2016.
- [TWTD16] Guangmo Tong, Weili Wu, Shaojie Tang, Ding-Zhu Du, Adaptive influence maximization in dynamic social networks, *IEEE/ACM Transactions on Networking* 25 (1) (2016) 112–125.
- [TXZ06] Y. Tao, X. Xiao, S. Zhou, Mining distance-based outliers from large databases in any metric space, in: Proc. 2006 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'06), Philadelphia, PA, USA, Aug. 2006, pp. 394–403.
- [TXZ16] Binh Tran, Bing Xue, Mengjie Zhang, Genetic programming for feature construction and selection in classification on high-dimensional data, *Memetic Computing* 8 (1) (2016) 3–15.
- [TYRW14] Yaniv Taigman, Ming Yang, Marc Aurelio Ranzato, Lior Wolf, Deepface: closing the gap to human-level performance in face verification, in: Proc. 2014 Conf. Computer Vision and Pattern Recognition (CVPR'14), Columbus, OH, USA, June 2014, pp. 1701–1708.
- [UBC97] P.E. Utgoff, N.C. Berkman, J.A. Clouse, Decision tree induction based on efficient tree restructuring, *Machine Learning* 29 (1997) 5–44.
- [UFS91] R. Uthurusamy, U.M. Fayyad, S. Spangler, Learning useful rules from inconclusive data, in: G. Piatetsky-Shapiro, W.J. Frawley (Eds.), *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991, pp. 141–157.
- [UMSJ13] Brian Uzzi, Satyam Mukherjee, Michael Stringer, Benjamin Jones, Atypical combinations and scientific impact, *Science (New York, N.Y.)* 342 (10 2013) 468–472.
- [Utg88] P.E. Utgoff, An incremental ID3, in: Proc. Fifth Int. Conf. Machine Learning (ICML'88), San Mateo, CA, USA, 1988, pp. 107–120.
- [Val87] P. Valduriez, Join indices, *ACM Transactions on Database Systems* 12 (1987) 218–246.
- [Vap95] V.N. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, 1995.
- [Vap98] V.N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.
- [VC71] V.N. Vapnik, A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and Its Applications* 16 (1971) 264–280.
- [VC06] M. Vuk, T. Curk, ROC curve, lift chart and calibration plot, *Metodološki Zvezki* 3 (2006) 89–108.
- [VCC<sup>+</sup>17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, Graph attention networks, arXiv preprint, arXiv:1710.10903, 2017.
- [vCPM<sup>+</sup>16] Guido van Capelleveen, Mannes Poel, Roland M. Mueller, Dallas Thornton, Jos van Hillegerberg, Outlier detection in healthcare fraud: a case study in the medicaid dental domain, *International Journal of Accounting Information Systems* 21 (2016) 18–31.
- [vdMPvdH09] Laurens van der Maaten, Eric Postma, Jaap van den Herik, Dimensionality reduction: a comparative review, *Tilburg centre for Creative Computing, TiCC TR 2009–005*, 2009.

- [VFH<sup>+</sup>18] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, R. Devon Hjelm, Deep graph infomax, arXiv preprint, arXiv:1809.10341, 2018.
- [VLK19] Arnaud Van Looveren, Janis Klaise, Interpretable counterfactual explanations guided by prototypes, arXiv preprint, arXiv:1907.02584, 2019.
- [VMZ06] A. Veloso, W. Meira, M. Zaki, Lazy associative classification, in: Proc. 2006 Int. Conf. Data Mining (ICDM'06), 2006, pp. 645–654.
- [vR90] C.J. van Rijsbergen, Information Retrieval, Butterworth, 1990.
- [VRG<sup>+</sup>18] Felipe Viegas, Leonardo Rocha, Marcos Gonçalves, Fernando Mourão, Giovanni Sá, Thiago Salles, Guilherme Andrade, Isac Sandin, A genetic programming approach for feature selection in highly dimensional skewed data, *Neurocomputing* 273 (2018) 554–569.
- [VSKB10] S. Vichy N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, Karsten M. Borgwardt, Graph kernels, *Journal of Machine Learning Research* 11 (2010) 1201–1242.
- [VSP<sup>+</sup>17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention is all you need, in: Proc. 2017 Conf. Neural Information Processing Systems (NIP'17), Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.
- [VTBE15] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, Show and tell: a neural image caption generator, in: Proc. 2015 Conf. Computer Vision and Pattern Recognition (CVPR'15), Boston, MA, USA, June 2015, pp. 3156–3164.
- [VvLS11] Jilles Vreeken, Matthijs van Leeuwen, Arno Siebes, Krimp: mining itemsets that compress, *Data Mining and Knowledge Discovery* 23 (1) (2011) 169–214.
- [VWI98] J.S. Vitter, M. Wang, B.R. Iyer, Data cube approximation and histograms via wavelets, in: Proc. 1998 Int. Conf. Information and Knowledge Management (CIKM'98), Washington, DC, USA, Nov. 1998, pp. 96–104.
- [WA18] Stefan Wager, Susan Athey, Estimation and inference of heterogeneous treatment effects using random forests, *Journal of the American Statistical Association* 113 (523) (2018) 1228–1242.
- [War63] Joe H. Ward, Hierarchical grouping to optimize an objective function, *Journal of the American Statistical Association* 58 (301) (1963) 236–244.
- [WCH10] T. Wu, Y. Chen, J. Han, Re-examination of interestingness measures in pattern mining: a unified framework, *Data Mining and Knowledge Discovery* 21 (2010) 371–397.
- [WCRS01] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, Constrained k-means clustering with background knowledge, in: Proc. 2001 Int. Conf. Machine Learning (ICML'01), San Francisco, CA, USA, 2001, pp. 577–584.
- [Wei04] G.M. Weiss, Mining with rarity: a unifying framework, *SIGKDD Explorations* 6 (2004) 7–19.
- [WF05] I.H. Witten, E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann, 2005.
- [WFHP16] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed., Morgan Kaufmann, 2016.
- [WIFYH03] Haixun Wang, Wei Fan, Philip S. Yu, Jiawei Han, Mining concept-drifting data streams using ensemble classifiers, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug. 2003, pp. 226–235.
- [WGH18] Yu-Xiong Wang, Ross B. Girshick, Martial Hebert, Bharath Hariharan, Low-shot learning from imaginary data, in: Proc. 2018 Conf. Computer Vision and Pattern Recognition (CVPR'18), June 2018, pp. 7278–7286.
- [WH60] Bernard Widrow, Marcian E. Hoff, Adaptive switching circuits, Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.
- [WH15] Chi Wang, Jiawei Han, *Mining Latent Entity Structures*, Morgan & Claypool, 2015.
- [WHC<sup>+</sup>19] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, Tat-Seng Chua, Kgat: knowledge graph attention network for recommendation, in: Proc. 2019 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'19), Anchorage, AK, USA, Aug. 2019, pp. 950–958.

- [WHH<sup>+</sup>89] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, Kevin J. Lang, Phoneme recognition using time-delay neural networks, *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37 (3) (1989) 328–339.
- [WHH00] K. Wang, Y. He, J. Han, Mining frequent itemsets using support constraints, in: *Proc. 2000 Int. Conf. Very Large Data Bases (VLDB'00)*, Cairo, Egypt, Sept. 2000, pp. 43–52.
- [WHLT05] J. Wang, J. Han, Y. Lu, P. Tzvetkov, TFP: an efficient algorithm for mining top-k frequent closed itemsets, *IEEE Transactions on Knowledge and Data Engineering* 17 (2005) 652–664.
- [WHP03] J. Wang, J. Han, J. Pei, CLOSET+: searching for the best strategies for mining frequent closed itemsets, in: *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, Washington, DC, USA, Aug. 2003, pp. 236–245.
- [WHS<sup>+</sup>21] Xuan Wang, Vivian Hu, Xiangchen Song, Shweta Garg, Jinfeng Xiao, Jiawei Han, ChemNER: fine-grained chemistry named entity recognition with ontology-guided distant supervision, in: *Proc. 2021 Conf. Empirical Methods in Natural Language Processing (EMNLP'21)*, Nov. 2021.
- [WHW<sup>+</sup>19] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, Tat-Seng Chua, Neural graph collaborative filtering, in: *Proc. 2019 Int. ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR'19)*, Paris, France, July 2019, pp. 165–174.
- [WI98] S.M. Weiss, N. Indurkha, *Predictive Data Mining*, Morgan Kaufmann, 1998.
- [Wid95] J. Widom, Research problems in data warehousing, in: *Proc. 1995 Int. Conf. Information and Knowledge Management (ICKM'95)*, Baltimore, MD, USA, Nov. 1995, pp. 25–30.
- [Win78] Shmuel Winograd, On computing the discrete Fourier transform, *Mathematics of Computation* 32 (141) (1978) 175–199.
- [WJS<sup>+</sup>19] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, Philip S. Yu, Heterogeneous graph attention network, in: *Proc. 2019 the Web Conf. (WWW'19)*, San Francisco, CA, USA, May 2019, pp. 2022–2032.
- [WK91] S.M. Weiss, C.A. Kulikowski, *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, Morgan Kaufmann, 1991.
- [WK05] J. Wang, G. Karypis, HARMONY: efficiently mining the best rules for classification, in: *Proc. 2005 SIAM Conf. Data Mining (SDM'05)*, Newport Beach, CA, USA, April 2005, pp. 205–216.
- [WLFY02] W. Wang, H. Lu, J. Feng, J.X. Yu, Condensed cube: an effective approach to reducing data cube size, in: *Proc. 2002 Int. Conf. Data Engineering (ICDE'02)*, San Fransisco, CA, USA, April 2002, pp. 155–165.
- [WLLZ18] Zhichun Wang, Qingsong Lv, Xiaohan Lan, Yu Zhang, Cross-lingual knowledge graph alignment via graph convolutional networks, in: *Proc. 2018 Conf. Empirical Methods in Natural Language Processing (EMNLP'18)*, Brussels, Belgium, Nov. 2018, pp. 349–357.
- [WM13] Sida Wang, Christopher Manning, Fast dropout training, in: *Proc. 2013 Int. Conf. Machine Learning (ICML'13)*, Atlanta, GA, USA, June 2013, pp. 118–126.
- [WMCL19] Liang Wu, Fred Morstatter, Kathleen M. Carley, Huan Liu, Misinformation in social media: definition, manipulation, and detection, *SIGKDD Explorations* 21 (2) (2019) 80–90.
- [Wri97] Stephen J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, 1997.
- [WS98] Duncan J. Watts, Steven H. Strogatz, Collective dynamics of “small-world” networks, *Nature* 393 (6684) (1998) 440–442.
- [WS09] Kilian Q. Weinberger, Lawrence K. Saul, Distance metric learning for large margin nearest neighbor classification, *Journal of Machine Learning Research* 10 (2) (2009).
- [WS15] Fei Wang, Jimeng Sun, Survey on distance metric learning and dimensionality reduction in data mining, *Data Mining and Knowledge Discovery* 29 (2) (2015) 534–564.
- [WSF95] R. Wang, V. Storey, C. Firth, A framework for analysis of data quality research, *IEEE Transactions on Knowledge and Data Engineering* 7 (1995) 623–640.

- [Wu83] C.F.J. Wu, On the convergence properties of the EM algorithm, *The Annals of Statistics* 11 (1983) 95–103.
- [WVM<sup>+</sup>17] Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, Steve Young, A network-based end-to-end trainable task-oriented dialogue system, in: *Proc. 2017 Conf. European Chapter of the Association for Computational Linguistics (EACL'17)*, Valencia, Spain, Apr. 2017, pp. 438–449.
- [WW96] Y. Wand, R. Wang, Anchoring data quality dimensions in ontological foundations, *Communications of the ACM* 39 (1996) 86–95.
- [WWW<sup>+</sup>18] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, Minyi Guo, Graphgan: graph representation learning with generative adversarial nets, in: *Proc. 2018 AAAI Conf. Artificial Intelligence (AAAI'18)*, New Orleans, LA, USA, Feb. 2018, pp. 2508–2515.
- [WWYY02] H. Wang, W. Wang, J. Yang, P.S. Yu, Clustering by pattern similarity in large data sets, in: *Proc. 2002 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'02)*, Madison, WI, USA, June 2002, pp. 418–427.
- [WXH08] T. Wu, D. Xin, J. Han, ARCube: supporting ranking aggregate queries in partially materialized data cubes, in: *Proc. 2008 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'08)*, Vancouver, BC, Canada, June 2008, pp. 79–92.
- [WXMH09] T. Wu, D. Xin, Q. Mei, J. Han, Promotion analysis in multi-dimensional space, *Proceedings of the VLDB Endowment* 2 (2009) 109–120.
- [WYM97] W. Wang, J. Yang, R. Muntz, STING: a statistical information grid approach to spatial data mining, in: *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, Athens, Greece, Aug. 1997, pp. 186–195.
- [WYO17] Zhiguang Wang, Weizhong Yan, Tim Oates, Time series classification from scratch with deep neural networks: a strong baseline, in: *Proc. 2017 Int. Joint Conf. Neural Networks (IJCNN'17)*, Anchorage, AK, USA, May 2017, pp. 1578–1585.
- [WZFC14] Zhen Wang, Jianwen Zhang, Jianlin Feng, Zheng Chen, Knowledge graph embedding by translating on hyperplanes, in: *Proc. 2014 AAAI Conf. Artificial Intelligence (AAAI'14)*, Québec City, Québec, Canada, July 2014, pp. 1112–1119.
- [WZY16] Ying Wei, Yu Zheng, Qiang Yang, Transfer knowledge between cities, in: *Proc. 2016 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'16)*, San Francisco, CA, USA, Aug. 2016, pp. 1905–1914.
- [WZYM19] Wei Wang, Vincent W. Zheng, Han Yu, Chunyan Miao, A survey of zero-shot learning: settings, methods, and applications, *ACM Transactions on Intelligent Systems and Technology* 10 (2) (2019) 13:1–13:37.
- [XBK<sup>+</sup>15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, Yoshua Bengio, Show, attend and tell: neural image caption generation with visual attention, in: *Proc. 2015 Int. Conf. Machine Learning (ICML'15)*, Lille, France, July 2015, pp. 2048–2057.
- [XCS12] Huan Xu, Constantine Caramanis, Sujay Sanghavi, Robust pca via outlier pursuit, *IEEE Transactions on Information Theory* 58 (5) (2012) 3047–3064.
- [XCYH06] D. Xin, H. Cheng, X. Yan, J. Han, Extracting redundancy-aware top-k patterns, in: *Proc. 2006 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'06)*, Philadelphia, PA, USA, Aug. 2006, pp. 444–453.
- [XGF16] Junyuan Xie, Ross Girshick, Ali Farhadi, Unsupervised deep embedding for clustering analysis, in: *Proc. 2016 Int. Conf. Machine Learning (ICML'16)*, New York City, NY, USA, June 2016, pp. 478–487.

- [XHCL06] D. Xin, J. Han, H. Cheng, X. Li, Answering top-k queries with multi-dimensional selections: the ranking cube approach, in: Proc. 2006 Int. Conf. on Very Large Data Bases (VLDB'06), Seoul, Korea, Sept. 2006.
- [XHLJ18] Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka, How powerful are graph neural networks?, arXiv preprint, arXiv:1810.00826, 2018.
- [XHLW03] D. Xin, J. Han, X. Li, B.W. Wah, Star-cubing: computing iceberg cubes by top-down and bottom-up integration, in: Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03), Berlin, Germany, Sept. 2003, pp. 476–487.
- [XHSL06] D. Xin, J. Han, Z. Shao, H. Liu, C-cubing: efficient computation of closed cubes by aggregation-based checking, in: Proc. 2006 Int. Conf. Data Engineering (ICDE'06), Atlanta, GA, USA, April 2006.
- [XHYC05] D. Xin, J. Han, X. Yan, H. Cheng, Mining compressed frequent-pattern sets, in: Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05), Trondheim, Norway, Aug. 2005, pp. 709–720.
- [XJRN02] Eric Xing, Michael Jordan, Stuart J. Russell, Andrew Ng, Distance metric learning with application to clustering with side-information, in: Proc. 2002 Conf. Neural Information Processing Systems (NIPS'02), Vancouver, BC, Canada, Dec. 2002, pp. 521–528.
- [XOJ00] Y. Xiang, K.G. Olesen, F.V. Jensen, Practical issues in modeling large diagnostic systems with multiply sectioned Bayesian networks, *International Journal of Pattern Recognition and Artificial Intelligence IJPRAI'00* (2000) 59–71.
- [XPK10] Zhengzheng Xing, Jian Pei, Eamonn J. Keogh, A brief survey on sequence classification, *SIGKDD Explorations* 12 (1) (2010) 40–48.
- [XSA17] Yongqin Xian, Bernt Schiele, Zeynep Akata, Zero-shot learning - the good, the bad and the ugly, in: Proc. 2017 Conf. Computer Vision and Pattern Recognition (CVPR'17), Honolulu, HI, USA, July 2017, pp. 3077–3086.
- [XSH<sup>+</sup>04] H. Xiong, S. Shekhar, Y. Huang, V. Kumar, X. Ma, J.S. Yoo, A framework for discovering co-location patterns in data sets with extended spatial objects, in: Proc. 2004 SIAM Int. Conf. Data Mining (SDM'04), Lake Buena Vista, FL, USA, April 2004.
- [XT15] Dongkuan Xu, Yingjie Tian, A comprehensive survey of clustering algorithms, *Annals of Data Science* 2 (2) (2015) 165–193.
- [XW05] Rui Xu, D. Wunsch, Survey of clustering algorithms, *IEEE Transactions on Neural Networks* 16 (3) (2005) 645–678.
- [XWY<sup>+</sup>19] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, Dong Yu, Cross-lingual knowledge graph alignment via graph matching neural network, arXiv preprint, arXiv:1905.11605, 2019.
- [XYFS07] X. Xu, N. Yuruk, Z. Feng, T.A.J. Schweiger, SCAN: a structural clustering algorithm for networks, in: Proc. 2007 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'07), San Jose, CA, USA, Aug. 2007.
- [YC01] N. Ye, Q. Chen, An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems, *Quality and Reliability Engineering International* 17 (2001) 105–112.
- [YCHX05] X. Yan, H. Cheng, J. Han, D. Xin, Summarizing itemset patterns: a profile-based approach, in: Proc. 2005 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'05), Chicago, IL, USA, Aug. 2005, pp. 314–323.
- [YCS<sup>+</sup>20] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, Yang Shen, Graph contrastive learning with augmentations, in: Proc. 2020 Conf. Neural Information Processing Systems (NeurIPS'20), Dec. 2020.
- [YDY<sup>+</sup>19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, Quoc V. Le, XLNet: generalized autoregressive pretraining for language understanding, in: Proc. 2019



- Conf. Neural Information Processing Systems (NeurIPS'19), Vancouver, BC, Canada, Dec. 2019, pp. 5754–5764.
- [YFB01] C. Yang, U. Fayyad, P.S. Bradley, Efficient discovery of error-tolerant frequent itemsets in high dimensions, in: Proc. 2001 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'01), San Francisco, CA, USA, Aug. 2001, pp. 194–203.
- [YFM<sup>+</sup>97] K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Computing optimized rectilinear regions for association rules, in: Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, USA, Aug. 1997, pp. 96–103.
- [YFS16] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, Learning from hidden traits: joint factor analysis and latent clustering, *IEEE Transactions on Signal Processing* 65 (1) (2016) 256–269.
- [YFSH17] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, Mingyi Hong, Towards k-means-friendly spaces: simultaneous deep learning and clustering, in: Proc. 2017 Int. Conf. Machine Learning (ICML'17), Sydney, NSW, Australia, Aug. 2017, pp. 3861–3870.
- [YH02] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, in: Proc. 2002 Int. Conf. Data Mining (ICDM'02), Maebashi, Japan, Dec. 2002, pp. 721–724.
- [YH03] X. Yin, J. Han, CPAR: classification based on predictive association rules, in: Proc. 2003 SIAM Int. Conf. Data Mining (SDM'03), San Francisco, CA, USA, May 2003, pp. 331–335.
- [YHC<sup>+</sup>18] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 974–983.
- [YHNV<sup>+</sup>15] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, George Toderici, Beyond short snippets: deep networks for video classification, in: Proc. 2015 Conf. Computer Vision and Pattern Recognition (CVPR'15), Boston, MA, USA, June 2015.
- [YHY08] Xiaoxin Yin, Jiawei Han, Philip S. Yu, Truth discovery with multiple conflicting information providers on the web, *IEEE Transactions on Knowledge and Data Engineering* 20 (6) (2008) 796–808.
- [YJ06] Liu Yang, Rong Jin, Distance metric learning: a comprehensive survey, *Michigan State University* 2 (2) (2006) 4.
- [YK09] L. Ye, E. Keogh, Time series shapelets: a new primitive for data mining, in: Proc. 2009 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'09), Paris, France, June 2009.
- [YLB<sup>+</sup>21] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, Hanghang Tong, Dynamic knowledge graph alignment, in: Proc. 2021 AAAI Conf. Artificial Intelligence (AAAI'21), Vancouver, BC, Canada, 2021, pp. 4564–4572.
- [YLYY12] Fan Yang, Yang Liu, Xiaohui Yu, Min Yang, Automatic detection of rumor on sina Weibo, in: Proc. 2012 ACM SIGKDD Workshop Mining Data Semantics (MDS'12), Beijing, China, Aug. 2012, pp. 1–7.
- [YML19] Liang Yao, Chengsheng Mao, Yuan Luo, Graph convolutional networks for text classification, in: Proc. 2019 Nat. Conf. Artificial Intelligence (AAAI'19), Honolulu, HI, USA, Jan. 2019, pp. 7370–7377.
- [YP19] Yu Yang, Jian Pei, Influence analysis in evolving networks: a survey, *IEEE Transactions on Knowledge and Data Engineering* (2019).
- [YPB16] Jianwei Yang, Devi Parikh, Dhruv Batra, Joint unsupervised learning of deep representations and image clusters, in: Proc. 2016 Conf. Computer Vision and Pattern Recognition (CVPR'16), Las Vegas, NV, USA, June 2016, pp. 5147–5156.
- [YS11] Dong Yu, Michael L. Seltzer, Improved bottleneck features using pretrained deep neural networks, in: Proc. 2011 Conf. Int. Speech Communication Association (INTERSPEECH'11), Florence, Italy, Aug. 2011, pp. 237–240.

- [YSJ<sup>+</sup>00] B.-K. Yi, N. Sidiropoulos, T. Johnson, H.V. Jagadish, C. Faloutsos, A. Biliris, Online data mining for co-evolving time sequences, in: Proc. 2000 Int. Conf. Data Engineering (ICDE'00), San Diego, CA, USA, Feb. 2000, pp. 13–22.
- [YTL<sup>+</sup>11] Rui Yan, Jie Tang, Xiaobing Liu, Dongdong Shan, Xiaoming Li, Citation Count Prediction: Learning to Estimate Future Citations for Literature, 10 2011, pp. 1247–1252.
- [YWY07] J. Yuan, Y. Wu, M. Yang, Discovery of collocation patterns: from visual words to visual phrases, in: Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR'07), Minneapolis, MN, USA, June 2007.
- [YYH03] H. Yu, J. Yang, J. Han, Classifying large data sets using SVM with hierarchical clusters, in: Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA, Aug. 2003, pp. 306–315.
- [YYH05] X. Yan, P.S. Yu, J. Han, Graph indexing based on discriminative frequent structure analysis, ACM Transactions on Database Systems 30 (2005) 960–993.
- [YYM<sup>+</sup>18] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, Jure Leskovec, Hierarchical graph representation learning with differentiable pooling, in: Proc. 2018 Conf. Neural Information Processing Systems (NeurIPS'18), Montréal, Canada, Dec. 2018, pp. 4800–4810.
- [YYR<sup>+</sup>18] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, Jure Leskovec, Graphrnn: generating realistic graphs with deep auto-regressive models, in: Proc. 2018 Int. Conf. Machine Learning (ICML'18), Stockholm, Sweden, July 2018, pp. 5708–5717.
- [YYY<sup>+</sup>20] Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, Yi Ma, Rethinking bias-variance trade-off for generalization of neural networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 10767–10777.
- [YYZ17] Bing Yu, Haoteng Yin, Zhanxing Zhu, Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting, arXiv preprint, arXiv:1709.04875, 2017.
- [YZWY17] Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu, Seqgan: sequence generative adversarial nets with policy gradient, in: Proc. 2017 Nat. Conf. Artificial Intelligence (AAAI'17), San Francisco, CA, USA, Feb. 2017, pp. 2852–2858.
- [YZYH06] X. Yan, F. Zhu, P.S. Yu, J. Han, Feature-based substructure similarity search, ACM Transactions on Database Systems 31 (2006) 1418–1453.
- [ZAG18] Daniel Zügner, Amir Akbarnejad, Stephan Günnemann, Adversarial attacks on neural networks for graph data, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 2847–2856.
- [Zak00] M.J. Zaki, Scalable algorithms for association mining, IEEE Transactions on Knowledge and Data Engineering 12 (2000) 372–390.
- [Zak01] M. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Machine Learning 40 (2001) 31–60.
- [ZDN97] Y. Zhao, P.M. Deshpande, J.F. Naughton, An array-based algorithm for simultaneous multidimensional aggregates, in: Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97), Tucson, AZ, USA, May 1997, pp. 159–170.
- [Zei12] Matthew D. Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint, arXiv:1212.5701, 2012.
- [ZG20] Xichen Zhang, Ali A. Ghorbani, An overview of online fake news: characterization, detection, and discussion, Information Processing & Management 57 (2) (2020) 102025.
- [ZH02] M.J. Zaki, C.J. Hsiao, CHARM: an efficient algorithm for closed itemset mining, in: Proc. 2002 SIAM Int. Conf. Data Mining (SDM'02), Arlington, VA, USA, April 2002, pp. 457–473.
- [ZH16] Yao Zhou, Jingrui He, Crowdsourcing via tensor augmentation and completion, in: Proc. 2016 Int. Joint Conf. Artificial Intelligence (IJCAI'16), New York, NY, USA, July 2016, pp. 2435–2441.
- [ZH17] Y. Zhou, J. He, A randomized approach for crowdsourcing in the presence of multiple views, in: Proc. 2017 Int. Conf. Data Mining (ICDM'17), New Orleans, LA, USA, Nov. 2017, pp. 685–694.

- [ZH19] Chao Zhang, Jiawei Han, *Multidimensional Mining of Massive Text Data*, Morgan & Claypool, 2019.
- [ZHCD15] Dawei Zhou, Jingrui He, K. Selçuk Candan, Hasan Davulcu, Muvir: multi-view rare category detection, in: Proc. 2015 Int. Joint Conf. Artificial Intelligence (IJCAI'15), Buenos Aires, Argentina, July 2015, pp. 4098–4104.
- [ZHGL13] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, Cheng-Lin Liu, Fast knn graph construction with locality sensitive hashing, in: Proc. 2013 Joint European Conf. Machine Learning and Knowledge Discovery in Databases (ECML-PKDD'13), Prague, Czech Republic, Sep. 2013, pp. 660–674.
- [ZHL<sup>+</sup>98] O.R. Zaiane, J. Han, Z.N. Li, J.Y. Chiang, S. Chee, MultiMedia-Miner: a system prototype for multimedia data mining, in: Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA, USA, June 1998, pp. 581–583.
- [Zhu05] X. Zhu, Semi-supervised learning literature survey, Computer Sciences Technical Report 1530, University of Wisconsin-Madison, 2005.
- [ZHZ00] O.R. Zaiane, J. Han, H. Zhu, Mining recurrent items in multimedia with progressive resolution refinement, in: Proc. 2000 Int. Conf. Data Engineering (ICDE'00), San Diego, CA, USA, Feb. 2000, pp. 461–470.
- [ZJ20] Mohammed J. Zaki, Wagner Meira Jr., *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*, 2nd ed., Cambridge University Press, 2020.
- [ZKF05] Ying Zhao, George Karypis, Usama Fayyad, Hierarchical clustering algorithms for document datasets, *Data Mining and Knowledge Discovery* 10 (2) (2005) 141–168.
- [ZL06] Z.-H. Zhou, X.-Y. Liu, Training cost-sensitive neural networks with methods addressing the class imbalance problem, *IEEE Transactions on Knowledge and Data Engineering* 18 (2006) 63–77.
- [ZL11] Li Zheng, Tao Li, Semi-supervised hierarchical clustering, in: Proc. 2011 Int. Conf. Data Mining (ICDM'11), Vancouver, BC, Canada, Dec. 2011, pp. 982–991.
- [ZL16] Barret Zoph, Quoc V. Le, Neural architecture search with reinforcement learning, arXiv preprint, arXiv:1611.01578, 2016.
- [ZLC<sup>+</sup>18] Qinghai Zhou, Liangyue Li, Nan Cao, Norbou Buchler, Hanghang Tong, Extra: explaining team recommendation in networks, in: Proc. 2018 Conf. Recommender Systems (RecSys'18), Vancouver, BC, Canada, Oct. 2018, pp. 492–493.
- [ZLT19] Qinghai Zhou, Liangyue Li, Hanghang Tong, Towards real time team optimization, in: Proc. 2019 Int. Conf. Big Data (BigData'19), Los Angeles, CA, USA, Dec. 2019, pp. 1008–1017.
- [ZNH18] Yao Zhou, Arun Reddy Nelakurthi, Jingrui He, Unlearn what you have learned: adaptive crowd teaching with exponentially decayed memory learners, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 2817–2826.
- [ZP17] Chong Zhou, Randy C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proc. 2017 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'17), Halifax, NS, Canada, Aug. 2017, pp. 665–674.
- [ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, in: Proc. 2017 Int. Conf. Computer Vision (ICCV'17), Venice, Italy, Oct. 2017, pp. 2223–2232.
- [ZPOL97] M.J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, Parallel algorithm for discovery of association rules, *Data Mining and Knowledge Discovery* 1 (1997) 343–374.
- [ZQL<sup>+</sup>11] F. Zhu, Q. Qu, D. Lo, X. Yan, J. Han, P.S. Yu, Mining top-*k* large structural patterns in a massive network, in: Proc. 2011 Int. Conf. Very Large Data Bases (VLDB'11), Seattle, WA, USA, Aug. 2011, pp. 807–818.
- [ZRGH12] Bo Zhao, Benjamin I.P. Rubinstein, Jim Gemmell, Jiawei Han, A bayesian approach to discovering truth from conflicting sources for data integration, in: Proc. 2012 Int. Conf. Very Large Data Bases (VLDB'12), Istanbul, Turkey, 2012, pp. 550–561.

- [ZRL96] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: an efficient data clustering method for very large databases, in: Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96), Montreal, Canada, June 1996, pp. 103–114.
- [ZS02] N. Zapkowicz, S. Stephen, The class imbalance program: a systematic study, *Intelligence Data Analysis* 6 (2002) 429–450.
- [ZSH<sup>+</sup>19] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, Nitesh V. Chawla, Heterogeneous graph neural network, in: Proc. 2019 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'19), Anchorage, AK, USA, Aug. 2019, pp. 793–803.
- [ZTC<sup>+</sup>18] Chao Zhang, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian M. Sadler, Michelle T. Vanni, Jiawei Han, TaxoGen: constructing topical concept taxonomy by adaptive term embedding and clustering, in: Proc. 2018 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'18), London, UK, Aug. 2018, pp. 2701–2709.
- [ZTX<sup>+</sup>19] Si Zhang, Hanghang Tong, Jiejun Xu, Yifan Hu, Ross Maciejewski, Origin: non-rigid network alignment, in: Proc. 2019 Int. Conf. Big Data (BigData'19), Los Angeles, CA, USA, Dec. 2019, pp. 998–1007.
- [ZTX<sup>+</sup>20] Si Zhang, Hanghang Tong, Yinglong Xia, Liang Xiong, Jiejun Xu, Nettrans: neural cross-network transformation, in: Proc. 2020 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'20), Aug. 2020, pp. 986–996.
- [ZVSL18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le, Learning transferable architectures for scalable image recognition, in: Proc. 2018 Conf. Computer Vision and Pattern Recognition (CVPR'18), Salt Lake City, UT, USA, June 2018, pp. 8697–8710.
- [ZWS<sup>+</sup>13] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, Cynthia Dwork, Learning fair representations, in: Proc. 2013 Int. Conf. Machine Learning (ICML'13), Atlanta, GA, USA, June 2013, pp. 325–333.
- [ZWW17] Honglei Zhuang, Chi Wang, Yifan Wang, Identifying outlier arms in multi-armed bandit, in: Proc. 2017 Conf. Neural Information Processing Systems (NIPS'17), Dec. 2017, pp. 5204–5213.
- [ZXL<sup>+</sup>17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, XiaoLei Huang, Dimitris N. Metaxas, Stackgan: text to photo-realistic image synthesis with stacked generative adversarial networks, in: Proc. 2017 Int. Conf. Computer Vision (ICCV'17), Venice, Italy, Oct. 2017, pp. 5907–5915.
- [ZXLS17] Hao Zhu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, Iterative entity alignment via joint knowledge embeddings, in: Proc. 2017 Int. Joint Conf. Artificial Intelligence (IJCAI'17), Melbourne, Australia, Aug. 2017, pp. 4258–4264.
- [ZXTZ21] Fan Zhou, Xovee Xu, Goce Trajcevski, Kunpeng Zhang, A survey of information cascade analysis: models, predictions, and recent advances, *ACM Computing Surveys* 54 (2) (2021) 1–36.
- [ZXZ21] Chengqing Zong, Rui Xia, Jiajun Zhang, *Text Data Mining*, Springer Singapore, 2021.
- [ZYH<sup>+</sup>07] F. Zhu, X. Yan, J. Han, P.S. Yu, H. Cheng, Mining colossal frequent patterns by core pattern fusion, in: Proc. 2007 Int. Conf. Data Engineering (ICDE'07), Istanbul, Turkey, April 2007.
- [ZYH19] Yao Zhou, Lei Ying, Jingrui He, Multi-task crowdsourcing via an optimization framework, *ACM Transactions on Knowledge Discovery from Data* 13 (3) (2019) 27:1–27:26.
- [ZYHY07] F. Zhu, X. Yan, J. Han, P.S. Yu, gPrune: a constraint pushing framework for graph pattern mining, in: Proc. 2007 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'07), Nanjing, China, May 2007.
- [ZYL<sup>+</sup>16] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, Wei-Ying Ma, Collaborative knowledge base embedding for recommender systems, in: Proc. 2016 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'16), San Francisco, CA, USA, Aug. 2016, pp. 353–362.
- [ZZ06] Jiong Zhang, Mohammad Zulkernine, Anomaly based network intrusion detection with unsupervised outlier detection, in: Proc. 2006 Int. Conf. Communications (ICC'06), Istanbul, Turkey, June 2006, pp. 2388–2393.

- [ZZH09] D. Zhang, C. Zhai, J. Han, Topic cube: topic modeling for OLAP on multidimensional text databases, in: Proc. 2009 SIAM Int. Conf. on Data Mining (SDM'09), Sparks, NV, USA, April 2009.
- [ZZY<sup>+</sup>17] Si Zhang, Dawei Zhou, Mehmet Yigit Yildirim, Scott Alcorn, Jingrui He, Hasan Davulcu, Hanghang Tong, Hidden: hierarchical dense subgraph detection with application to financial fraud detection, in: Proc. 2017 SIAM Int. Conf. Data Mining (SDM'17), Houston, TX, USA, Apr. 2017, pp. 570–578.

# Index

## A

- Accuracy measure, 282, 283, 297, 299
- Activation function, 274, 485–488, 490, 497, 498, 500, 502
- Active learning (AL), 647
- Adaptive
  - learning rate, 501, 503, 514–516, 548
  - probabilistic networks, 318
- Adaptive Boosting (AdaBoost), 292, 293, 296, 304
- Advanced learning methods, 637
- Adversarial
  - learning, 556, 640
  - training, 633, 635
  - training leverage, 555
- Affinity matrix, 480
- Agglomerative
  - clustering framework, 406
  - hierarchical clustering, 397, 399, 400, 429
- Agglomerative hierarchical clustering
  - algorithms, 429
  - framework, 426
  - methods, 395–399, 406, 426
- Algebraic measure, 137
- Algebraic measure closedness, 143
- Algorithmic fairness, 649
- All-versus-all (AVA), 360, 362
- Amortized scans, 120
- Animal classification, 349
- Anomaly mining, 10
- Antimonotonic constraints, 195
- Apriori
  - algorithm, 149–152, 155, 157, 169, 170, 173, 193, 194, 197
  - property, 150–153, 161, 181, 193, 194, 201, 202, 220
  - pruning, 182, 202, 204, 233
  - pruning heuristic, 204
- Artificial neural network (ANN), 485, 552
- Artificial outliers, 563
- Assessing
  - attribute relevance, 84
  - clustering tendency, 417
  - similarity, 23
- Association rules, 7, 11, 145, 147, 148, 150, 152–154, 175, 177–180, 327, 335–337, 369, 375
- Associative
  - classification, 327, 335, 336, 338, 369, 371, 375
  - classification methods, 341
  - classifiers, 341
- Asymmetric
  - attributes, 47
  - binary attributes, 46, 47, 79, 81
  - binary dissimilarity, 47
- Atomic transactions, 88
- Attention, 526, 536–538, 545, 548, 610, 616
- Attributes
  - age, 27, 80, 82, 251, 255, 260, 263, 591
  - binary, 23, 25, 27, 43, 46, 47, 64, 79, 261, 267, 582
  - cluster, 445
  - construction, 83
  - contextual, 559–561, 590–592
  - customers, 57
  - data, 74
  - data set, 385
  - database, 68, 179, 180, 242
  - evaluation measures, 73
  - grouping, 126
  - measurement, 51
  - measures, 86
  - multiple, 43, 67, 84, 328
  - number, 103, 591
  - numeric, 23–29, 31, 33–36, 38, 41, 43, 46, 48–51, 175, 180, 235, 267, 391
  - selection, 84, 244, 296, 303, 334
  - selection measure, 84, 244, 245, 247–249, 255–257, 298, 303
  - selection process, 74, 78
  - set, 71, 100
  - space, 72, 266, 267
  - subset selection, 71–74, 78, 80, 83, 84, 311
  - tests, 328–330, 333, 335
  - types, 27, 43, 50, 51, 79
  - values, 23, 47, 55, 56, 93, 130, 177, 192, 210, 244, 260, 261, 270, 328, 334, 366, 379, 405, 447, 540, 562, 577, 592
  - vector, 24, 261, 273, 275, 276
  - weighting, 268, 303
- Augmentation policy learning, 633, 635
- Auto-correlation, 610
- Autoencoder, 511–514, 549, 597, 598
  - for
    - deep clustering, 512
    - dimensionality reduction, 512
    - unsupervised pretraining, 514
- Automated Machine Learning (Auto-ML), 605, 640–642

## Automatic

- classification, 381
- feature learning, 487

Average treatment effect (ATE), 637

**B**

Background knowledge, 16, 269, 431, 436, 458, 475, 477–479, 560, 561, 643, 647

Backpropagation, 10, 64, 489–495, 498

Bagged classifier, 292

Bagging, 290–292, 296, 506, 507, 596, 597

## Base

- classifiers, 229, 290, 291
- cuboid, 99, 113–115, 118, 121, 125, 129, 130

Bayes classification methods, 259

Bayes classifier, 239, 260–263

- classification error, 261

## Bayesian

- belief networks, 307, 315–317, 365, 369, 374
- classification, 259, 303
- classifiers, 259, 262, 299
- clustering, 236

Bayesian Optimization (BO), 641, 642

Behavioral attribute, 559, 560, 590, 591, 601

Behavioral attribute values, 559

Biclust, 448, 450–453

Biclustering, 447, 449, 450, 463, 483

- methods, 448, 452, 480
- models, 483
- techniques, 448
- useful, 448

## Binary

- attributes, 23, 25, 27, 43, 46, 47, 64, 79, 261, 267, 582
- classification, 359, 361, 365, 492, 525, 550, 651
  - problem, 280, 363
  - schemes, 370
  - task, 272, 274, 275, 509, 510, 549
- classifiers, 300, 360, 370, 552

## Boolean

- attributes, 367
- variables, 315

Bootstrap, 284, 285, 291, 507, 587, 603

Bottom-Up Construction (BUC), 120, 125–127

Business intelligence (BI), 16, 17, 20, 85, 379, 380, 425

Buying patterns, 147

**C**

C4.5, 73, 84, 244, 252, 258, 298, 302, 328, 330, 331, 337

## Candidate

- generation, 150, 157, 169, 173, 201, 204, 213, 214
- generation process, 157, 161
- itemsets, 151, 156, 157, 161, 170, 193, 194
- patterns, 183, 193

CART, *see* Classification and Regression Trees (CART)

Case-based reasoning (CBR), 269, 304

## Categorical

- attributes, 61, 308, 582–584
- classification rules, 302

Categorized data cube, 143

Causality, 36, 366, 605, 635–637

Central tendency measures, 23, 25, 27, 28, 30, 32, 42, 57

Centralized federated learning, 645

Centroid, 386, 398

CF-trees, *see* Clustering feature tree (CF-tree)

Chi-square test, 36, 62, 225

Chunking, 121, 139

Churn prediction, 240, 539

## Class

- imbalance problem, 280, 283, 297–299, 302, 304
- label attribute, 242, 249, 250, 263, 281, 300, 316, 336
- label prediction, 7, 278, 290
- prediction, 243, 292, 329, 330, 337, 365, 487

## Classification

- accuracy, 10, 243, 244, 290, 297, 300, 302, 307, 311, 341, 352, 359, 526, 550, 552
- algorithms, 64, 259, 304, 359
- Bayesian, 259, 303
- binary, 359, 361, 365, 492, 525, 550, 651
- data, 239, 240, 307, 376
- decision, 298
- decision boundary, 552
- error, 263, 319, 353, 360, 499
- error rate, 261, 353
- interpretability, 307, 359, 364
- method, 7, 239, 302, 305, 345, 370, 376, 380, 605
- model, 7, 13, 78, 239, 278, 285–288, 307, 308, 311, 312, 314, 328, 338, 341, 346, 539, 585, 590
- model interpretability, 283
- model learning, 340
- model performance, 311
- naïve Bayesian, 7, 260, 263, 299, 300, 315, 317
- patterns, 429
- performance, 362
- performance improvement, 308
- problems, 260, 282, 489, 491, 511, 617, 623
- process, 242, 316
- results, 274, 364, 365, 651
- rules, 7, 242, 244, 283, 331, 375
- scheme, 300, 338
- semisupervised, 242, 342–344, 369, 371
- sequence, 354, 355
- setting, 307, 352
- SVM, 371
- task, 78, 242, 270, 274, 307, 308, 342, 346, 349, 354, 355, 491, 509, 548, 550, 552, 614
- techniques, 240, 351, 352

- text, 366, 376, 555, 606, 609, 610, 631
- tools, 287
- unsupervised, 381
- Classification and Regression Trees (CART), 73, 244, 248, 253, 256, 258, 296, 298, 302
- Classification Based on Association (CBA), 336
- Classifier
  - accuracy, 243, 304
  - accuracy estimates, 283
  - evaluation measures, 278
  - medical, 282
  - model, 292, 293, 308
  - naïve Bayesian, 259–263, 265, 303, 304, 307, 315, 375
  - perceptron, 273
  - probabilistic, 288
  - quality, 227
  - rule set, 338
  - SVM, 324, 326, 364, 370, 652, 654
  - trained, 276, 307, 311, 349, 351, 352, 359
  - training, 307, 311
- Classifying
  - attribute, 331
  - future data tuples, 319
  - new tuples, 299
  - test tuples, 268
- CLIQUE, 414, 415, 427, 445, 446
- Closed patterns, 174, 187–189, 220, 234, 236, 341
- Cluster
  - analysis, 6, 9, 20, 21, 66, 71, 83, 379–381, 431, 432, 435–438, 441, 442, 447, 449, 454, 463
    - methods, 431
    - quality, 417
    - tools, 381
  - assignment, 431, 432, 438, 479
  - attributes, 445
  - boundaries, 417
  - centers, 386, 387, 391, 411, 419, 427, 434, 438, 440, 445, 460, 476, 513, 514
  - completeness, 421
  - computing environment, 16
  - hierarchy, 402, 405, 406
  - homogeneity, 421, 427
  - labels, 420, 473
  - means, 386, 392, 429
  - memberships, 472–474, 482
  - objects, 426, 431, 447
  - patterns, 188
  - points, 427
  - quality, 397, 426, 479
  - related tuples, 119
  - sample, 70
  - sampling, 83
  - separation, 383
  - similarity, 429
  - size, 411
  - structures, 472
- Clustering
  - algorithms, 182, 344, 380–383, 385, 397, 404, 461, 462, 472, 478–480, 539, 648
  - analysis, 402, 405, 417, 424, 427, 441, 443, 445, 455, 456, 460, 464, 465, 475, 557, 617
  - analysis applications, 404
  - analysis methods, 444
  - approaches, 444
  - assumption, 344
  - Bayesian, 236
  - coefficient, 356
  - criteria, 70, 385
  - data, 182, 381
  - data set, 411
  - deep, 513, 514, 553
  - documents, 382, 383
  - evaluation, 417, 427, 430
  - evaluation methods, 429
  - fuzzy, 433–435, 438, 439, 479
  - graph, 431, 463, 471, 480, 483
  - hierarchical, 384, 394–396, 398, 401–406, 478
  - hybrid data, 429
  - incremental, 382, 426
  - information, 464
  - methods, 189, 379–385, 402, 407, 411, 413, 416, 417, 431, 432, 441, 444, 463, 465, 470–472, 475, 476, 478, 479, 563, 588, 640
  - methods quality, 428
  - models, 420, 449, 465, 480
  - multilevel, 383
  - nominal data, 390, 429, 430
  - objects, 381
  - order, 411
  - output, 423
  - paradigms, 430
  - partitioning, 385, 386, 482
  - patterns, 187
  - phrase, 631
  - problems, 405, 459, 460
  - process, 387, 392, 410, 479
  - quality, 382, 384, 389, 396, 406, 417, 420, 421, 424, 477, 480
    - evaluation, 430
    - measure, 421
    - score, 477
  - results, 382, 391, 400, 402, 415–417, 422–424, 460, 475, 480, 513, 614
  - semisupervised, 431, 475–477, 479, 480, 483
  - steps, 513, 514
  - streaming, 404



- structure, 78, 427, 456, 512–514
  - task, 78, 409, 512, 514, 547
  - techniques, 187, 242, 379, 380, 385, 396, 403, 415, 447, 449
  - tendency, 417–420, 427, 435
  - text, 605, 608, 610
  - tool, 379
  - Clustering feature (CF), 382, 402–404, 426
  - Clustering feature tree (CF-tree), 402–404, 426, 429
  - Code words, 360, 361, 371, 583–585
  - Collective outlier, 557, 559, 560, 563, 590, 592, 593, 600
    - detection, 560, 561, 592, 593, 600, 601, 603
    - mining, 592
  - Colossal patterns, 183, 184, 236
  - Colossal patterns mining, 232
  - Complete set, 157, 162, 169, 183, 205–207, 215, 336, 337, 341
  - Compressed
    - data cubes, 143
    - database, 157
    - patterns, 232
    - transactions, 113
  - Computational epidemiology models, 623, 625
  - Computer
    - clouds, 17, 18
    - communication networks, 5
    - networks, 1
    - science, 16, 173, 237
    - systems, 92
    - vision, 4, 10, 16, 212, 317, 342, 364
  - Concept
    - hierarchy, 59, 65, 66, 97, 103–106, 108, 176, 177, 179
    - hierarchy for income, 180
    - learning, 244, 302
    - learning systems, 244
  - Conditional average treatment effect (CATE), 637
  - Conditional outliers, 559, 590
  - Conditional probability table (CPT), 315, 317
  - Confidence measures, 164, 166, 169
  - Confident
    - label prediction, 343
    - prediction, 343
  - Conflict resolution, 329, 331
  - Confusion matrix, 279–281, 299
  - Connectionist learning, 552
  - Constraint-based clustering, 382, 426
  - Constraint-based pattern mining, 191, 193, 234, 236
  - Constraints
    - contextual, 640
    - multiple, 193, 198, 222
    - pruning, 196, 236
  - Context words, 606
  - Contextual
    - attribute, 559–561, 590–592
    - attribute values, 592
    - constraints, 640
    - outlier, 559, 560, 590–592, 600
    - outlier analysis, 560
    - outlier detection, 559–561, 590–592, 601–603
    - outlier detection methods, 592
    - outlier detection quality, 560
    - outliers, 590
  - Continuous attributes, 27, 308
  - Continuous bag-of-words (CBOW), 606, 607
  - Conventional
    - classifier, 354, 356
    - outlier detection, 594
  - Convertible constraints, 195, 196, 198
  - Convex clusters, 392
  - Convolutional Neural Network (CNN), 10, 485, 488, 499, 514, 517, 525, 526, 529, 543, 548, 553, 554, 625, 634, 651
  - Corporate data warehouses, 134
  - Correlation
    - analysis, 34, 37, 62, 66, 79, 163–165
    - coefficient, 28, 36, 62
    - measure, 164, 165, 174
  - Cosine
    - measure, 53, 79, 166, 168, 173
    - similarity, 52, 53, 81, 469, 607
    - similarity measure, 52, 53
  - Cost complexity pruning, 258
  - Cotraining, 343, 344, 369, 371
  - Covariance, 34, 35, 62, 456, 457, 567
  - Credit card transactions, 21, 557
  - Cross-entropy, 509–511, 514, 516
  - Cross-validation, 283–286, 299, 311, 353, 420, 427
  - Crowdsourcing learning, 343
  - Cube measure, 130
  - Cuboid, 98–100, 108, 113, 114
  - Customer purchasing behavior, 278
- ## D
- Data
    - antimonotonic constraint, 222
    - attributes, 74
    - augmentation, 605, 632–635
    - classification, 239, 240, 307, 376
    - cleaning, 2, 19, 23, 24, 42, 55, 56, 60–63, 68, 86, 90, 92, 133
      - process, 61
      - routines, 79
    - clustering, 182, 381
    - cube, 4, 6, 62, 70, 84, 85, 96–99, 105, 106, 108, 109, 113–118, 120, 125, 129–138, 180, 181, 234, 235
      - aggregation, 78
      - computation, 113, 119, 120, 140, 142, 143, 181
      - lattice, 116, 134

- materialization, 115
- measure, 121
- structure, 134, 138
- lakes, 85, 93–96, 134
- mining, 1–5, 23, 24, 43, 51, 53, 55, 85, 90, 92, 93, 95, 145, 163, 174, 177, 212, 256, 285, 296, 302, 312, 352, 359, 364, 366, 380, 381, 385, 388, 425, 441, 449, 576, 605, 610, 617–620, 623, 624, 633
- algorithms, 11, 14, 277, 425, 638, 646
- applications, 1, 17, 43, 431, 483, 615, 617
- community, 165
- conference, 22, 580
- for social good, 652, 653
- functionalities, 20
- methodologies, 4, 629
- methods, 4, 5, 18, 21, 612, 645, 653
- models transparent, 366
- process, 13, 192, 194, 653
- tool, 61, 379, 381
- venues, 577
- multidimensional, 70, 71, 85, 89, 97, 99, 106, 118, 134, 499, 539
- partition, 244, 253
- partitioning, 285, 302
- pruning, 196
- quality measures, 55
- set
  - attributes, 385
  - clustering, 411
- space pruning, 197
- training, 7
- transactional, 154, 176, 179
- transactions, 20
- tuples, 10, 24, 36, 37, 74, 76, 232, 242, 243, 260, 263, 316, 317, 338, 344, 345, 349, 352, 355, 359, 481, 482, 496, 499, 547, 596
- unlabeled, 342–345, 349, 369, 475, 606, 609, 620, 631
- warehouse, 24, 55, 56, 85–93, 179, 181
  - architecture, 85
  - design, 93, 101
  - environment, 90
  - management tools, 90
  - metadata, 133
  - modeling, 96
  - structure, 90
  - systems, 86, 87, 134
- warehousing, 15, 17, 24, 62, 86, 87, 90, 92, 93, 98, 125, 136, 142
- warehousing schema, 110
- Database
  - attributes, 68, 179, 180, 242
  - compressed, 157
  - design, 87
  - layout, 173
  - mining, 84
  - objects, 410
  - points, 81
  - relational, 4, 5, 20, 28, 43, 86, 88, 93, 99, 118, 133, 135, 145, 174, 179, 224
  - research communities, 119
  - scans, 150, 155–157
  - schema, 88, 99, 103, 104
  - sequence, 199–202, 204, 205, 208
  - server, 92, 119
  - size, 594, 603
  - systems, 15, 55, 63, 83, 87, 118
  - technology, 12, 15, 16, 20
  - transactional, 6, 171, 172
  - transactions, 11, 147, 149, 151, 158, 160, 168, 189, 197, 204
  - tuples, 242
- DBLP data set, 173
- DBSCAN, 472, 482
- Decentralized federated learning, 645
- Decision tree, 7
- Deconvolutional neural network, 554
- Deep
  - autoencoder for outlier detection, 604
  - clustering, 513, 514, 553
  - learning, 4, 9, 10, 13, 55, 78, 348, 355, 356, 381, 485, 489, 497–500, 507, 508, 513, 538, 539, 548, 552, 594, 597–599, 601, 610, 614, 625, 626, 637, 640
  - algorithms, 485
  - approaches, 646, 654
  - architectures, 500, 514
  - methods, 277, 597, 598, 642
  - models, 76, 497, 514, 517, 543, 598, 625, 636, 642
  - models predictions, 652
  - techniques, 10, 308, 352, 362, 364
  - neural network, 274, 488, 489, 498, 499, 501, 504, 505, 509, 515, 547, 606, 639, 646
  - residual learning, 554
- Deep CNN (DCNN), 554
- Deep graph infomax (DGI), 555
- Dendrogram, 401, 402, 426
- Density estimation, 303, 411, 571, 601
- Density-based clustering, 385, 408, 411, 416, 472, 575, 586
- Density-based outlier detection, 564, 572–574, 601–603
- Descriptive mining, 5
- Determiner, 229
- Dimensionality reduction, 24, 71, 74, 76–78, 80, 431, 454, 455, 458, 480, 512–514, 594, 595
  - autoencoder for, 512
  - for clustering, 454
  - methods, 455, 456, 479, 480
  - step, 513
  - techniques, 68, 78

## Discovering

- biclusters, 452
- clusters, 384
- frequent itemsets, 177
- interesting patterns, 1, 2, 4, 19

## Discrete attributes, 27

## Discrete Fourier transform (DFT), 69

## Discrete wavelet transform (DWT), 68

## Discretized attribute, 309

## Discretizing numeric attributes, 180

## Discriminative pattern mining, 237

## Dispersion measures, 28

## Dissimilar tuples, 363, 370

## Dissimilarity

- computation, 50
- for attributes, 50
- function, 385
- matrix, 43–46, 50–52, 83
- measures, 43
- objects, 43, 48
- value, 43

## Distance

- measurements, 24, 64, 268, 360
- measures, 10, 43, 48–50, 52, 54, 83, 188, 189, 191, 234, 355, 362, 369, 379, 382, 388, 424, 431, 442, 443, 462, 465, 561, 572, 574
- metric learning, 362, 363, 370

## Distant

- supervision, 342, 348, 349, 369, 372, 376, 631
- training, 229

## Distributive measures, 105

## Divisive hierarchical clustering, 395, 396, 400, 401, 426

## Document

- classification, 342, 362, 605, 629, 631
- similarity, 55
- similarity measure, 52

## Domain knowledge, 10, 16, 382, 388, 426, 475, 476, 478, 480, 614, 646

## Dropout, 240, 504–507, 548

## Dummy attribute, 272, 274

## Dunn index, 424, 425, 427

## Dynamic itemset counting, 157, 173

**E**

## Eager

- classification, 300
- learning methods, 266

## Eclat algorithm, 160

## Editing method, 268, 303

## Elbow method, 419, 425, 427

## Electronics store, 239, 240, 243, 250, 346, 348, 364, 367, 560, 569, 589, 591, 595

## Embedded method, 308, 312, 369, 370

## Ensemble classifier, 229

## Enterprise data warehouses, 88, 91, 92, 135

## Entropy Balancing (EB), 637

## Euclidean distance, 48, 50, 81, 355, 362, 363, 386, 388, 392, 393, 439, 442, 443, 465, 481

## Evaluation measures, 163, 165–167, 170, 278, 280, 283

## attributes, 73

## classifier, 278

## Evolutionary Algorithm (EA), 641

## Expectation maximization (EM) algorithm, 438, 479, 569

## Explainable learning, 640

## Exploitation, 368, 370

## Exploration, 85, 93, 120, 125, 163, 173, 181, 368, 370, 374, 593, 594

## Extracting

- classification rules, 330
- clusters, 462

**F**

## Facilitated prediction, 144

## Fact

- constellation, 96, 99, 101–103, 134
- table, 97, 99–101, 105, 108, 110, 111

## Fairness constraints, 649

## False negative, 279, 286, 297–299, 301

## False positive, 279, 286–288, 297, 299, 301

## Feature engineering, 10, 307, 308, 354, 356, 369

## Feature engineering methods, 308

## Federated learning, 556, 645

## Filter methods, 308, 311, 312, 369, 370

## FP-growth, 157, 159, 160, 169, 171, 183, 204, 337

## FP-tree, 157–159, 204, 337, 341

## Frequent closed patterns, 236

## Frequent itemset mining, 7, 145, 146, 149, 162, 169, 171–174, 181, 199–201, 214, 222, 335–339

## Frequent patterns, 6, 7, 15, 20, 145, 149, 157, 175, 182, 183, 327, 335, 338, 339

## Frequent subgraphs, 19, 212, 218, 220, 233

## FSG algorithm, 214

## Fuzzy

- clustering, 433–435, 438, 439, 479
- clustering methods, 430, 441
- clusters, 385, 432–435

**G**

## Gain ratio, 84, 249, 252, 253, 255, 303

## Gated Recurrent Unit (GRU), 554

## Gaussian similarity function, 461

## Generalized linear models, 653, 654

## Generative adversarial network (GAN), 556, 633, 634

## Generative models, 405, 406, 473, 556, 565, 613, 614, 616, 624

## Generic graph clustering methods, 480

## Genetic learning, 367

- Geodesic distance, 465, 466, 480
  - Gini
    - impurity, 245, 249, 253–255, 257
    - index, 84, 400
  - Gradient
    - exploding problem, 514, 515, 533
    - vanishing problem, 500, 501, 510, 514, 516, 533, 547, 548
  - Gradient Approach (GA), 641
  - Graph
    - cuts, 480
    - data classification, 352, 355, 356, 369
    - pattern mining, 4, 222
  - Graph convolutional network (GCN), 540, 550, 551
  - Graph isomorphism network (GIN), 555
  - Graph neural network (GNN), 499, 539, 540, 548, 554
  - Graphical user interface (GUI), 61
  - Greedy supervised training, 508
  - Grid-based methods, 379, 384, 385, 407, 414, 426, 427, 445
  - Grouping
    - attributes, 126
    - customers, 209
    - data objects, 394
  - Grubb's test, 567, 568, 599, 603
- ## H
- Hamming distance, 360, 361
  - Heterogeneity, 62, 79, 610, 629, 647, 648
  - Heterogeneous
    - federated learning, 645
    - network, 555, 613, 614, 616, 640
    - network mining, 613, 614
    - transfer learning, 348
  - Heterogeneous graph neural network (HetGNN), 555
  - HetPathMine, 616
  - Heuristic clustering methods, 384
  - Hidden Markov model, 355
  - Hierarchical
    - clustering, 384, 394–396, 398, 401–406, 478
      - algorithms, 399, 429
      - methods, 394, 396, 397
      - methods clustering quality, 429
      - quality, 384
    - clusters, 398
    - partitioning, 395
  - High-dimensional clustering, 182, 381, 430, 443, 444, 471
  - Histogram, 23, 28, 38, 40–42, 66–68, 419, 565, 569, 570
  - Holdout method, 283
  - Holistic measures, 106, 137
  - Hopkins statistic, 419, 427
  - Human computer interaction (HCI), 12, 16
  - Human learning, 552
  - Human-algorithm interaction, 646
  - Human-in-the-loop, 646
  - Human-machine teaming, 647, 648
  - Hybrid OLAP (HOLAP), 117, 118, 134
- ## I
- Iceberg cube, 116, 120, 125, 126, 129, 135, 138, 139
  - ID3, 73, 244, 249, 252, 298, 302, 334
  - Imbalance ratio (IR), 168
  - Importance Sampling Learning Ensembles (ISLE), 304
  - Inconvertible constraints, 193, 196
  - Incremental
    - clustering, 382, 426
      - clustering algorithms, 382
      - learning, 266
      - mining, 183
  - Independent attributes, 270, 442, 443
  - Indexes, 108, 109, 134
  - Indexing, 15, 88, 92, 108–110, 212, 237, 269, 299, 459, 518
  - Information
    - clustering, 464
    - diffusion, 623–626
    - gain, 245, 249–255, 257, 310, 333–335, 338, 340
    - gain measure, 73, 84, 252
    - network, 4, 5, 15, 21, 629, 632
    - retrieval, 16, 43, 53, 129, 191, 226, 304, 380, 381
    - theoretic measure, 255
    - theory based measures, 423
  - Information extraction (IE), 608
  - Inherent
    - clustering structure, 403
    - similarity, 443
  - Instrumental variable (IV), 637
  - Intelligent learning system, 552
  - Interactive data
    - cleaning tool, 83
    - mining, 16
  - Interactive mining, 16, 85
  - Intercluster
    - separation, 425
    - similarity, 386
  - Interestingness
    - constraints, 192
    - measures, 148, 163, 167, 169, 174, 178, 186
    - measures quality, 186
  - Interpretability, 55, 56, 79, 283, 359, 365, 366, 382, 383, 426,
    - 562, 584, 614, 625, 636, 642, 648–650
    - classification, 307, 359, 364
    - outliers detection, 580
  - Interpretable
    - data mining, 650
    - machine learning, 377, 652
    - predictions, 650
  - Intertransaction association rule mining, 237
  - Interval-scaled attribute, 79

- Intracluster similarity, 386
- Inverse document frequency (IDF), 226
- Invisible data mining, 16
- Irrelevant attributes, 72, 268
- Itemset
  - association, 157
  - frequency, 162
  - frequent, 156
  - merging, 162
  - mining, 178, 481
  - subsets, 161
  - support, 148, 153
- J**
- Jaccard similarity, 481
- Joint predictive model, 629, 640
- K**
- k*-anonymity, 643
- k*-fold cross-validation, 283, 284
- k*-means, 182, 189, 381, 384–389, 438, 443, 444, 454, 459, 461, 462, 475, 476, 513, 586, 644, 645
- k*-medoids, 381, 384, 388–390, 426, 445, 481
- k*-nearest neighbor, 7, 266–269, 307, 355, 373, 461, 572, 574, 575, 594
- Kernel
  - density, 303, 411–413, 565, 571, 601
  - function, 75, 326, 355, 359, 393, 408, 412, 426, 652, 654
- Kernel PCA (KPCA), 74
- KL divergence, 43, 53–55, 76, 514, 553, 555
- Knowledge
  - bases, 3, 14, 96, 228, 229, 348, 606, 608, 631
  - graph, 212, 546, 613, 615, 616, 629, 632, 640
  - graph mining, 613, 615
  - mining, 2
- Knowledge discovery from data (KDD), 1
- Kulc measure, 168
- Kulczynski measure, 166, 170, 174, 186
- L**
- Language model, 237, 607–609, 615, 616, 632
- Laplacian, 265, 374, 462, 541, 543–545, 551
- LASSO, 304, 313, 314, 342, 365
- Latent
  - clustering, 553
  - clusters, 441
- Lattice of cuboids, 99, 100, 113, 115, 125, 134, 181
- Lazy classification, 300
- Learned classifier, 243, 324
- Learning
  - agent, 368, 370
  - algorithms, 266, 272, 273, 275, 276, 278, 340, 342, 345, 369, 381, 646–649
  - approach, 375
  - belief networks, 374
  - capability, 614
  - concept, 244, 302, 304
  - curves, 345
  - deep, 4, 9, 10, 13, 55, 78, 348, 355, 356, 381, 485, 489, 497–500, 507, 508, 513, 538, 539, 548, 552, 594, 597–599, 601, 610, 614, 625, 626, 637, 640
  - framework, 648
  - hierarchical representations, 609
  - incremental, 266
  - methods, 242, 292, 364, 635
  - models, 571, 635, 640, 641, 646, 650
  - neural network, 490, 552
  - node, 555
  - performance, 295
  - phase, 485
  - process, 339, 489
  - rate, 300, 301, 318, 492, 495, 498, 501, 503, 516, 548, 549, 552, 553
  - rules, 333, 334
  - semisupervised, 14, 242, 344, 364, 553, 589, 602, 635
  - sequential, 331
  - sparse, 312
  - stage, 598
  - step, 240, 242
  - supervised, 13, 239, 242, 342, 345, 359, 381, 553, 620, 646
  - system, 646
  - task, 348
  - techniques, 606
  - unsupervised, 13, 14, 242, 381, 429, 509, 512, 555, 563, 653
- Least angle regression (LAR), 314
- Least square
  - linear regression, 312
  - linear regression model, 294
  - prediction error, 314
  - regression, 271, 272, 314
  - regression model, 314
- Leave-one-out, 284, 304
- Lift, 164–166
- Likelihood, 163, 164, 225, 260, 262, 275, 277, 312, 335, 344, 361, 405, 406, 435–438, 441, 509, 539, 562, 566, 626
- Linear
  - classifiers, 239, 269, 270, 274, 298, 299, 320, 322, 323, 342, 361, 365, 487, 488
  - classifiers sparse, 365, 366
- Linear regression, 59, 240, 269–272, 294, 308, 312–314, 491
- Link prediction, 541, 615, 616
- Linkage measures, 397, 404, 426
- Local
  - optimal biclusters, 452
  - outlier, 559, 574, 575

- outlier analysis, 559
  - outlier factor, 575
  - Logistic regression, 261, 270, 274, 276, 277, 287, 298, 301, 312, 314, 317, 320, 342, 352, 356, 359, 361, 485, 487, 489, 539, 606, 653, 654
  - classifier, 274–277, 300, 311, 314, 317, 318, 326, 342, 349, 485, 491, 509
  - learning algorithm, 276
  - Long short-term memory (LSTM), 534–536, 538, 545, 548
  - Lossless compression, 187, 188, 236
  - Lossy compression, 69, 70, 236
- M**
- Machine learning
    - algorithms, 376, 556, 648
    - approach, 372
    - community, 261
    - interpretable, 377, 652
    - methods, 15
    - predictions, 372
  - Machine-in-the-loop, 646, 647
  - Magnetic resonance imaging (MRI), 633
  - Mahalanobis distance, 363, 364, 371, 567, 568, 601
  - Majority voting, 247, 267, 290
  - Manhattan distance measures, 381
  - MaPLe, 452–454
  - Market basket analysis, 145–147, 149, 169
  - Markov chains, 474
  - Materialization, 108, 115, 117, 121, 134
  - Materializing data cubes, 129
  - Max patterns, 162, 174, 187
  - Maximal frequent itemsets, 148, 149, 162, 188
  - Maximum iteration number, 273, 275, 314, 357
  - Maximum likelihood estimation (MLE), 275
  - Maximum marginal hyperplane (MMH), 319, 321, 322
  - Mean, 23–30, 32–34
  - Measure
    - clustering quality, 421
    - data cube, 121
    - document similarity, 52
    - numeric, 28
  - Median, 23, 25–31
  - Metadata, 60, 62, 79, 85, 89, 90, 133
  - Metric
    - learning, 362
    - learning distance, 362, 363
    - measure, 54
  - Microclusters, 397
  - Min-max normalization, 64, 82, 267, 597
  - Minimum Description Length (MDL), 222, 256, 415, 584
  - Minimum spanning tree, 400, 401, 411, 426
  - Minimum support, 116, 125–127, 148, 150–154, 156, 157, 163, 177, 178, 181, 182, 189, 200, 201, 336, 337, 339, 375
    - count, 126, 139, 148, 150, 151, 156–158, 161
    - count threshold, 148, 149
    - threshold, 116, 120, 135, 147–150, 169, 170, 172, 177, 178, 182, 185–187, 194, 199, 212, 220, 337
  - Mining
    - algorithms, 14, 24, 198, 214, 221
    - alternative substructure patterns, 220
    - approach, 236
    - approximate, 222, 233
    - association rules, 148
    - associations, 145
    - biomedical data, 16
    - biomedical networks, 19
    - closed
      - frequent graphs, 220
      - frequent patterns, 207
      - graphs, 220
      - itemsets, 162, 174
      - subgraph patterns, 233
    - coherent substructures, 223, 233
    - collective outliers, 592
    - colossal patterns, 232
    - complex data types, 605
    - compressed patterns, 187
    - criteria, 180
    - cube space, 144
    - cyclic, 237
    - data, 1–5, 23, 24, 43, 51, 53, 55, 85, 90, 92, 93, 95, 145, 163, 174, 177, 212, 256, 285, 296, 302, 312, 352, 359, 364, 366, 380, 381, 385, 388, 425, 441, 449, 576, 605, 610, 617–620, 623, 624, 633
    - data sets, 232
    - database, 84
    - efficiency, 221
    - frequent
      - episodes, 237
      - itemsets, 7, 148–150, 157, 160, 161, 169, 170, 173, 220, 232
      - patterns, 6, 15, 145, 165, 185, 193, 223, 237
      - subgraph, 212, 218
      - subgraph patterns, 19, 212
      - substructures, 233
      - subtrees, 221
    - geospatial data, 16
    - graph patterns, 220
    - incremental, 183
    - intent, 618
    - itemsets, 178, 193, 194, 481
    - large substructural patterns, 237

- long itemsets, 375
- meaningful patterns, 176
- methodology, 21, 183
- methods, 4, 196, 202, 210, 219, 232, 234
- models, 614, 640, 651
- multidimensional associations, 179, 185
- multidimensional patterns, 183
- multilevel
  - association rules, 176, 178, 233
  - associations, 175, 233
  - patterns, 175, 178
  - rules, 178
- multiple intercorrelated signals, 639
- negative association rules, 236
- negative patterns, 187
- network, 638, 639
- nonsimple graphs, 221
- OLAP, 144
- participar kinds, 21
- patterns, 4, 5, 11, 13, 175, 183, 197, 198, 200, 202, 210, 211, 223
- performance, 10, 638
- personal data, 642
- quality, 10
- quantitative association rules, 180, 235
- quantitative rules, 235
- query, 193, 198
- rare patterns, 175, 185, 236
- request, 221
- rich data types, 605
- sequence, 237
- sequential patterns, 7, 175, 198, 200, 204, 205, 220, 237
- social media, 16, 18
- software bugs, 16, 230
- space pruning, 197
- spatial association rules, 237
- stream data, 5
- structural patterns, 237
- subgraph patterns, 175, 211
- substructure patterns, 221
- system, 192
- tasks, 57, 72, 181, 192, 616, 639, 652
- text, 4, 19, 21, 431, 605, 606, 608, 609
- Minkowski distance, 48, 49, 81
- Misclassification, 556
  - error, 294
  - rate, 280
- Misclassified tuples, 293, 294, 296
- Misinformation, 604, 605, 620–623
- Missing values, 13, 23, 56, 57, 60, 92, 267, 283, 317, 406, 554, 562
- Mixture models, 436–438, 440, 479, 569, 571, 591, 603
- Mode, 23, 25–28, 30, 31
- Model
  - classification, 7, 13, 78, 239, 278, 285–288, 307, 308, 311, 312, 314, 328, 338, 341, 346, 539, 585, 590
  - classifier, 292, 293, 308
  - constructed predicts, 240
  - learning, 571
  - perceptron, 300
  - prediction, 13, 144, 302, 625
  - prediction interpretability, 646
  - selection, 278, 285, 299
- Modularity, 471, 480
- MOLAP, *see* Multidimensional OLAP (MOLAP)
- Monotonic constraints, 196, 210
- Motifs, 223
- Multi-instance learning, 343
- Multiagent reinforcement learning, 653, 654
- Multiarmed bandit (MAB), 374
- Multiclass classification
  - problem, 362, 371
  - scheme, 631
  - task, 510, 525
- Multiclass classifier, 351
- Multidimensional
  - association rule mining, 180
  - data, 70, 71, 85, 89, 97, 99, 106, 118, 134, 499, 539
    - analysis, 97
    - cubes, 6
    - mining, 15
    - models, 96, 97
    - set, 406
    - storage, 98
    - stores, 118, 119
  - databases, 7, 15, 136, 137, 141–143, 179
  - patterns, 175
- Multidimensional OLAP (MOLAP), 89, 117, 134
- Multifeature data cubes, 144
- Multiinstance learning, 343
- Multilabel
  - classification, 362
  - classification problem, 362
- Multilayer neural networks, 552, 607
- Multilayer perceptron (MLP), 488, 555
- Multilevel
  - association mining, 235
  - clustering, 383
  - concept hierarchy, 67
  - mining, 200, 235
  - mining methods, 233
- Multimedia data cube, 143
- Multiresolution clustering approach, 430
- Multisource transfer learning, 376
- Multitask learning, 348
- Multivalued attributes, 255, 256, 303

- Multivariate
  - data set, 567, 568
  - outlier, 567, 568, 603
  - outlier detection, 567–569
- Mutual information (MI), 310
- N**
- Naïve Bayes classifier, 301, 307, 353
- Naïve Bayesian
  - classification, 7, 260, 262, 263, 298–300, 315, 317, 369
  - classifier, 259–263, 265, 303, 304, 307, 315, 375
  - classifier predictive power, 303
- Named entity recognition (NER), 608
- Natural language processing, 10, 12, 16, 349, 354, 355, 364, 485, 526, 552, 554, 608–610, 619, 622, 640
  - tasks, 538
  - techniques, 617
- Nearest-neighbor clustering, 397
- Negative
  - correlation, 41, 42, 164, 175, 185, 187
  - patterns, 175, 185, 232
  - training, 269
  - training example, 323
  - tuples, 278, 279, 281, 287, 297, 310, 326, 337, 344, 487
  - tuples attribute vectors, 276
- Neighborhoods, 58, 66, 356, 384, 408, 409, 461, 462, 467, 472, 518, 522, 571, 572, 614
- Nesterov accelerated gradient (NAG), 553
- Network
  - mining, 619
  - mining social, 18
  - representation learning, 540
- Network of networks, 638–640
- Network of time series, 638–640
- Neural network, 7, 10, 261, 298, 300, 485–488, 606, 608, 614, 616, 637, 642
  - architecture, 554
  - capability, 552
  - classifiers, 259, 263
  - deep, 274, 488, 489, 498, 499, 501, 504, 505, 509, 515, 547, 606, 639, 646
  - learning, 490, 552
  - models, 543, 545
- Ng-Jordan-Weiss algorithm, 461, 462
- Noise, 2, 10, 11, 13, 21, 23, 27, 42, 56, 58, 60, 68, 137, 229, 244, 257, 276, 283, 298, 307, 326, 360, 382, 384, 388, 431, 442, 443, 506, 517, 524, 553, 558, 560, 562, 619, 637, 644
- Nominal attributes, 23–25, 37, 44–46, 55, 81, 177, 180, 181, 267, 336, 390, 391, 429
- Nonantimonotonic constraint, 196
- Nonbase cuboid, 113
- Noncancer tuples, 281
- Nonclosed itemset, 188
- Noncore objects, 409
- Nonhierarchical clustering methods, 429
- Nonlinear
  - classification models, 365
  - classifier, 326, 487, 552
- Nonmetric measure, 53
- Nonnegative matrix factorization (NMF), 454, 458, 480, 514
- Nonnegativity constraints, 581
- Nonneural network, 489
- Nonoverlapping partitions, 156, 170
- Nonparametric statistical methods, 564, 565, 569, 570
- Nonrepresentative object, 389, 390
- Normalization, 24, 50, 61, 63–65, 99, 252, 357, 446, 457, 515, 553, 579, 597
- Numeric
  - attribute, 23–29, 31, 33–36, 38, 41, 43, 46, 48–51, 175, 180, 235, 267, 391
  - attribute values, 180
  - data sets, 30, 458
  - measure, 28
  - prediction, 7, 240, 266, 267, 298, 319, 489
- Numerical
  - attributes, 577
  - prediction, 240
  - prediction model, 242
- O**
- Objective interestingness measures, 163
- Objects
  - allocation, 391
  - assignment, 426
  - class label, 7
  - cluster, 9, 402, 426, 431, 447
  - clustering, 381
  - data, 7, 23, 24, 43, 193, 380–385, 388, 431, 432, 441, 445–447, 456, 459, 557–561
  - database, 410
  - dissimilarity, 43, 48
  - identifier, 47
  - matching, 62
  - multiple, 574
  - outliers, 562, 563, 570, 572, 574, 589, 602
  - similarity, 43, 55, 79, 438
  - unlabeled, 564
- OLAP
  - data cubes, 106, 144
  - databases, 113
  - mining, 144
  - query, 88, 110, 112, 129, 134
  - query processing, 117, 133, 142, 143
- OLTP database, 90



- One-class model, 588, 589, 601
  - Online
    - social networks, 431
    - store, 33, 39, 591
    - webstore, 55
  - Online analytical processing (OLAP), 15
  - Operational databases, 86, 88, 92, 96, 133
  - Opinion mining, 617–619
    - applications, 619
    - techniques, 617, 619
  - OPTICS, 411, 412, 427
  - Optimal
    - biclusters, 453
    - biclusters local, 452
    - clustering, 471
  - Optimization objective, 481
  - Ordinal attributes, 23, 25, 26, 43, 49, 50
  - Organization donation database, 141
  - Outlier detection
    - methods, 562, 564
    - techniques, 585
  - Outliers
    - analysis, 6, 10, 20, 43, 55, 59, 79
    - contextual, 559, 560, 590–592, 600
    - detection, 10, 20, 21, 78, 381, 485, 530, 539, 541, 557–566, 569–571, 575, 579, 580, 585, 589
      - applications, 563
      - cost, 571
      - ensemble, 596
      - interpretability, 580
      - methods, 559, 561, 562, 564, 566, 587, 591, 593–596, 598, 599, 601
      - models, 562
      - problem, 591, 594
      - process, 590, 592
      - quality, 561, 564
      - setting, 585
      - stage, 598
      - techniques, 557
    - distribution, 563
    - local, 559, 574, 575
    - multivariate, 567, 568, 603
    - objects, 562, 563, 570, 572, 574, 589, 602
    - samples, 563, 588
    - score, 570, 586, 591, 599, 602
    - sensitivity, 397
    - structures, 593
    - vertex, 470
  - Outweighing attributes, 64, 267
  - Overlapping biclusters, 454
- P**
- Pairwise distance measurements, 442
  - PAM, *see* Partitioning Around Medoids (PAM) algorithm
  - Parametric statistical methods, 564, 565
  - Partition
    - boundaries, 128
    - data, 244, 253
    - data objects, 383
    - photos, 380
    - projects, 380
    - quality, 384
    - size, 156
    - tuples, 249
  - Partitioned cube shell fragments, 135
  - Partitioning
    - algorithm, 385, 404
    - clustering, 385, 386, 482
    - clustering approaches, 478
    - cost, 119
    - criterion, 385, 426
    - customers, 383
    - data, 285, 302
    - hierarchical, 395
    - methods, 379, 383–386, 388, 389, 391, 394, 398, 399, 426
    - process, 395
    - quality, 386
    - rules, 67
    - technique, 156, 173
    - tuples, 256
  - Partitioning Around Medoids (PAM) algorithm, 389, 391, 428, 429
  - Patient object, 25
  - Pattern
    - antimonotonic constraint, 196, 222
    - candidate, 183, 193
    - classification, 429
    - cluster, 188
    - clustering, 187
    - complete set, 183
    - compressed, 232
    - compression problem, 189
    - growth mining, 173
    - interestingness measure, 169
    - mining, 4, 5, 11, 13, 175, 183, 197, 198, 200, 202, 210, 211, 223
      - algorithm, 194, 202, 226
      - constraints, 193
      - frequent, 6, 15, 145, 165, 185, 193, 223, 237
      - methods, 175, 183, 236
      - multilevel, 175, 178
      - process, 191, 193, 197, 232
    - monotonic constraint, 194, 196, 222
    - multidimensional, 175
    - negative, 175, 185, 232
    - pruning, 196

- pruning constraints, 193, 236
  - sequential, 6, 15, 175, 198–202, 205, 207
- Pearson's correlation coefficient, 36, 82
- Peer prediction, 646
- Percentiles, 28, 31, 32, 39
- Perceptron, 272, 274, 276, 298, 304, 320, 365, 485, 487, 488, 552
  - algorithm, 273, 276
  - classifier, 273
  - learning algorithm, 273
  - model, 300
- Performance prediction, 239, 628, 639
- Pessimistic pruning, 258, 303
- Phrase
  - clustering, 631
  - mining, 223, 224, 226–229, 233, 237, 608, 631, 632
    - framework, 228
    - methods, 226, 229
    - process, 227
  - quality, 224, 225
    - estimation, 227, 228
    - score, 229
- Pivot (rotate) operation, 108, 134
- Pool-based approach, 345
- Pooling, 523–525
- Postpruning, 257, 258
- Postpruning approach, 258
- Potential outliers, 43, 60
- Power law distribution, 614, 615
- Precaution measures, 240
- Predefined concept hierarchies, 180, 181
- Predetermined significance threshold, 227
- Predictability, 625
- Prediction
  - accuracy, 549, 652
  - class, 243, 292, 329, 330, 337, 365, 487
  - class label, 7, 278, 290
  - consistency, 629
  - cube, 141
  - error, 314, 360
  - methods, 640
  - model, 13, 144, 302, 625
  - numeric, 7, 240, 266, 267, 298, 319, 489
  - performance, 639
  - performance improvement, 628
  - problems, 240, 242, 298, 623, 626, 654
  - quality, 295
  - results, 628, 646, 649
  - tasks, 625
  - techniques, 17, 239
- Predictive
  - abilities, 278, 299
  - accuracy, 243, 278
  - analysis, 7, 19
  - analytics, 17
  - data mining, 5
  - modeling, 16
  - models, 629, 640
  - power, 338
  - rate, 649
  - rate parity, 649
  - score, 649
  - statistics, 13
- Predictor, 283
- Prepruning, 257, 258
- Prepruning approach, 257
- Pretrained language model (PLM), 607–610, 629–631
- Pretrained model, 508
- Pretraining, 507–509, 514, 516, 550, 553, 608
  - method, 509
  - strategy, 548
  - supervised, 507, 514
  - unsupervised, 509, 512, 514
- Price attribute, 40
- Principal component analysis (PCA), 71, 454, 456, 480, 595, 603
- Privacy-preserving data mining, 642
- Probabilistic
  - classifier, 287, 288
  - clustering methods, 513
  - clusters, 432, 435–438, 479
  - hierarchical clustering, 404
- PROCLUS, 446
- Proximity measures, 23, 44, 46, 356, 357, 359, 369, 594, 599
  - for
    - binary attributes, 46
    - ordinal attributes, 49
- Prune
  - actions, 150
  - component, 153
  - rules, 337
  - step, 150, 151, 170
- Pruning
  - algorithm, 258
  - Apriori, 182, 202, 204, 233
  - constraints, 196, 236
  - data, 196
  - methodology, 194
  - methods, 258, 303
  - pattern, 193, 196
  - pattern search space, 193
  - power, 236
  - rules, 210
  - set, 258, 335
  - strategies, 193, 375
  - trees, 258

**Q**

- Qualitative attributes, 26
- Quality
  - classification models, 14, 609
  - classifier, 227
  - cluster, 397, 426, 479
  - cluster analysis, 417
  - clustering, 382, 384, 389, 396, 406, 417, 420, 421, 424, 477, 480
  - control, 380
  - estimation, 227, 228
  - estimator, 229
  - hierarchical clustering, 384
  - interestingness measures, 186
  - measures, 226, 420
  - mining, 10
  - outliers detection, 561, 564
  - partition, 384
  - partitioning, 386
  - phrases, 224–229
  - prediction, 295
  - segmentation, 226
- Quantile plots, 23, 28, 38, 40, 79
- Quantile-quantile plots, 28, 38, 39, 79
- Quantitative attributes, 30, 175, 180, 181, 232
  - clusters, 182
- Quartiles, 28, 31, 32
- Query
  - approximation, 143
  - language, 118, 192
  - mining, 193, 198
  - node, 358
  - OLAP, 88, 110, 112, 129, 134
  - optimizer, 192
  - processing, 92, 113, 118, 132, 133, 415
  - processing in data cubes, 132
  - processing time, 416
  - specification, 416
  - tuple, 363
  - user, 17, 464
  - vector, 357
  - words, 52
- Querying, 90
- Querying function, 345

**R**

- Radial basis function (RBF), 75, 371, 393
- Random forest, 227, 290, 296, 297, 299, 304, 342, 637, 642, 652, 654
- Random walk, 356–359, 465–469, 555, 616
- Ranking, 25, 26, 49, 50, 108, 134, 140, 240, 242, 249, 331, 364, 366, 649
  - cube, 140, 141, 144

- query, 140
- query processing, 141
  - results, 614
  - scores, 640
  - task, 363
- Rare patterns, 175, 185, 232
  - mining, 175, 185, 236
- Raster data set, 611
- Ratio-scaled attributes, 79
- Reachability
  - density, 575, 602
  - distance, 411, 574, 575
- Receiver operating characteristic (ROC) curves, 278, 286–288
- Recidivism prediction, 649
- Recognition rate, 278, 280, 281
- Recurrent Neural Network (RNN), 10, 485, 489, 499, 526, 527, 538, 548, 554, 555, 633, 640
  - model, 527–530, 532, 533, 536–538
- Recursive partitioning, 247
- Redundancy measure, 191
- Redundant attributes, 71, 72
- Regression, 6–9, 57, 59, 71, 144, 240, 242, 247, 248, 308, 311, 312
- Regularized autoencoder, 514
- Reinforcement Learning (RL), 307, 359, 367, 368, 370, 374, 377, 485, 556, 628, 641
- Relational
  - data store, 119
  - database, 4, 5, 20, 28, 43, 86, 88, 93, 99, 118, 133, 135, 145, 174, 179, 224
  - query processing, 142
- Relational OLAP (ROLAP), 117, 134
- Relationship extraction (RE), 608
- Representation learning, 489, 555, 606, 614, 615
- Retraining, 228, 598
- RFID data warehouse, 137
- Robustness, 88, 94, 183, 283, 366, 383, 633, 635–637, 648–651
- ROC curves, *see* Receiver operating characteristic (ROC)
- ROLAP data store, 118
- Rule pruning, 331, 335
- Rule pruning strategies, 337
- Rule-based classification, 307, 327, 329, 365, 375

**S**

- Sampling, 24, 63, 70, 78, 83, 156, 173, 196, 284, 291, 293, 345, 555, 616, 626
- Saturation, 547
- Scalable
  - data mining tools, 1
  - database technologies, 15
  - hierarchical clustering, 394

- Scan
  - algorithm, 482, 483
  - database, 157
- Scatter plots, 28, 36, 38, 41, 42, 79, 271, 310
- Scattered patterns, 176
- Self-training, 343, 344, 369, 371, 631
- Semantic attribute, 350, 351
  - classifier, 350, 351, 369
  - classifier output, 351
  - vector, 351
- Semantic classifier, 351
- Semisupervised
  - classification, 242, 342–344, 369, 371
  - clustering, 431, 475–477, 479, 480, 483
  - clustering methods, 475, 476
  - hierarchical clustering, 478, 483
  - learning, 14, 242, 344, 364, 553, 589, 602, 635
  - learning methods, 563
  - node classification, 358
  - outlier detection, 563, 564, 602
- Sensitive attributes, 649
- Sentiment classification, 240, 346, 348, 526, 618
  - accuracy, 364
  - example, 348
  - method, 376
- Sentiment classifier, 348
- Sequence
  - classification, 354, 355
  - data set, 4
  - database, 199–202, 204, 205, 208
  - mining, 237
  - mining methods, 5
- Sequential
  - learning, 331
  - mining, 200
  - mining algorithms, 233
  - pattern mining, 4, 198–201, 207, 209–211, 230–233
    - algorithm, 231, 235
    - applications, 200
    - methods, 200
    - process, 211
  - patterns, 6, 15, 175, 198–202, 205, 207
    - complete set, 205, 208
- Shell
  - cubes, 140
  - fragments, 117, 120, 129–132, 135
- Sigmoid function, 274, 275, 485, 491, 492, 500, 501, 542
- Significance
  - measures, 191
  - tests, 285, 299, 304
- Similarity
  - assessing, 23
  - cluster, 429
  - function, 53, 399, 461
  - graph, 455, 461, 462
  - learning problem, 364
  - matrix, 394, 462, 471, 480–482, 614
  - measure for clustering graph, 482
  - measurements, 431, 446, 561
  - measurements unreliable, 383
  - measures, 43, 52, 55, 188, 383, 395, 443, 463, 465, 466, 608, 616, 639
  - objects, 43, 55, 79, 438
  - scale, 401
  - search, 614
  - SimRank measure, 483
  - threshold, 472
  - values, 43, 50
- Simple random sample with replacement (SRSWR), 70
- SimRank, 466–468, 480, 482
- Singleton clusters, 401
- Singular value decomposition (SVD), 579
- Skyline query, 141
- Slice operation, 108
- Snowflake schema, 96, 100–102, 118, 134, 135
- Social
  - event mining, 198
  - media, 17, 18, 90, 96, 605, 618, 619, 621–623
    - analysis, 608
    - applications, 648
    - data, 18, 21, 617
    - layout, 90
    - mining, 18
    - networks, 18
    - repository, 90
    - sites, 18
    - tools, 1
    - trustability analysis, 18
    - usage, 18
  - network, 1, 5, 16–18, 464, 466, 539, 540, 547, 593, 613, 626
    - analysis, 10, 18, 480, 485, 622
    - data, 18, 21
    - mining, 18
    - sites, 539, 622
    - structures, 18
  - networking sites, 621
- Soft clustering, 435
- Software bug mining program, 232
- Sorted data set, 32
- Space pruning, 196, 197
  - data, 197
  - mining, 197
- Spam prediction task, 549
- Sparse
  - clusters, 404, 587, 601
  - data cubes, 136, 143

- data sets, 118
  - learning, 312
  - linear classifiers, 365, 366
  - Spatiotemporal data mining, 5
  - Specificity, 278, 281, 283, 297, 299
  - Spectral clustering, 455, 460, 461, 463, 471
    - approaches, 461
    - methods, 460, 463, 471, 481
  - Spherical clusters, 382
  - SQL query, 116, 135
  - Stacked autoencoder, 511, 512
  - Standard deviation, 28, 31, 33, 34, 36, 60, 64, 65, 79, 137, 263, 415, 437, 440, 457, 515, 566, 567, 569, 595
  - Star schema, 96, 99–101, 134, 136
  - Statistics, 9, 11–13, 15, 23, 27, 29, 34, 38, 163, 198, 226, 239, 302, 366, 381, 388, 402, 411, 441, 602, 628, 650
  - Stepwise
    - backward elimination, 73, 311
    - forward selection, 73, 311
  - STING clustering, 415, 417
  - Stochastic block model (SBM), 480
  - Stochastic gradient descent (SGD), 553
  - Stochastic neighbor embedding (SNE), 74, 75
  - Stopwords, 226
  - Stratified cross-validation, 284
  - Stream data
    - classification, 352
    - mining tasks, 376
  - Stream mining algorithms, 20
  - Streaming data mining algorithms, 16
  - Structure patterns, 214
  - Structuring unstructured data, 605, 629, 632
  - Student's *t*-test, 285
  - Subcluster, 404
  - Subcube query, 132
  - Subfrequent items, 233
  - Subgraph
    - mining, 230
    - mining algorithms, 233
    - pattern, 220
    - pattern mining, 175, 211, 233
    - pattern mining closed, 233
    - pattern mining frequent, 19, 212
  - Subitemset, 149
  - Subjective measures, 191
  - Subspace clustering, 383
    - algorithm, 427
    - methods, 445, 454, 480
  - Substructure
    - mining, 215, 223
    - mining algorithms, 213
    - patterns, 223
    - patterns mining, 221
  - Subtree pruning, 258
  - Succinct constraints, 210, 222
  - Succinctness constraints, 197
  - Sum of squared errors (SSE), 399
  - Superitemset, 149, 188
  - Superpatterns, 197
  - Supervised
    - data mining methods, 625, 653
    - greedy pretraining, 548
    - learning, 13, 239, 242, 342, 345, 359, 381, 553, 620, 646
    - learning algorithms, 244, 652
    - pretraining, 507, 514
  - Support
    - association rule, 147
    - group-based, 178, 185
    - reduced, 178
    - uniform, 177, 178
  - Support vector data description (SVDD), 589, 602
  - Support vector machine (SVM), 318, 323, 371, 374
    - classification, 371
    - classifier, 324, 326, 364, 370, 652, 654
  - Suspected outliers, 587
  - Symbolic patterns, 198
  - Symmetric binary
    - attributes, 46
    - dissimilarity, 46
  - Synthetic data sets, 283
- ## T
- t*-test, 225, 285, 286
  - Tables, 4, 6, 63, 86, 95, 97, 98, 154, 167, 173, 315, 585, 596
  - Taxonomy, 9, 53, 605, 608, 624, 629, 630
  - Taxonomy construction, 608, 610, 629
  - Team performance prediction, 627–629, 639
  - Temporal data mining tasks, 611
  - Term-frequency vectors, 43, 52, 53, 79
  - Test
    - sets, 243, 258, 278, 280, 283–286, 335, 353, 420
    - tuples, 243, 248, 266, 268, 286, 287, 323, 499, 506, 515, 547
  - Text
    - classification, 366, 376, 555, 606, 609, 610, 631
    - classifiers, 609
    - clustering, 605, 608, 610
    - clustering algorithms, 608
    - clustering analysis, 608
    - databases, 143
    - document clustering, 52
    - mining, 4, 19, 21, 431, 605, 606, 608, 609
    - mining tasks, 605, 610, 615
  - Tightness measure, 187
  - Time Delayed Neural Network (TDNN), 554
  - Time-series data, 4, 5, 70, 144, 237, 517, 638
  - Token classification, 526

- ToPMine, 226, 227
  - TrAdaBoost, 347, 369, 376
  - Trained
    - Bayesian belief networks, 315, 369
    - binary classifiers, 360, 371
    - classifier, 276, 307, 311, 349, 351, 352, 359
    - deep neural networks, 547, 552
    - models, 515, 635
    - network, 495
  - Training
    - cases, 269
    - data, 7
  - Transactional
    - data, 154, 176, 179
    - data analysis, 172
    - data sets, 4, 167
    - database, 6, 171, 172
  - Transactions
    - compressed, 113
    - customers, 85, 146, 569
    - data, 20
    - data set, 145, 170, 196, 233
    - database, 11, 147, 149, 151, 158, 160, 168, 189, 197, 204
    - database mining, 157
    - identifiers, 160, 171
    - pattern, 557
  - Transfer learning, 14, 342, 346–348, 509, 635, 653
  - Transitive similarity, 472
  - Tree pruning, 244, 257, 299, 303
    - algorithms, 298
    - methods, 257, 330
    - process, 312
  - Trend analysis, 90, 108, 198
  - True negative, 279, 286, 299, 301
  - True positive, 279, 286, 287, 299, 301
  - Truth discovery, 605, 620–622
  - Tuples
    - class label, 278
    - data, 10, 24, 36, 37, 74, 76, 232, 242, 243, 260, 263, 316, 317, 338, 344, 345, 349, 352, 355, 359, 481, 482, 496, 499, 547, 596
    - data set, 336
    - database, 242
    - negative, 278, 279, 281, 287, 297, 310, 326, 337, 344, 487
    - nonunique classification, 282
    - outlying, 582
    - partition, 249
    - partitioning, 256
    - unlabeled, 344
  - Two sample *t*-test, 286
- U**
- UCI data sets, 338
  - Unimodal, 30, 285
  - Uninteresting patterns, 187
  - Univariate distribution, 38, 39
  - Univariate outlier, 565
    - detection, 566, 567, 599
  - Unlabeled
    - data, 342–345, 349, 369, 475, 606, 609, 620, 631
    - data tuples, 342
    - objects, 564
    - training tuples, 242
    - tuples, 344
    - tuples clustering structure, 344
  - Unordered attributes, 71
  - Unpruned
    - decision tree, 229
    - tree, 330
  - Unstructured data, 4, 6, 15, 20, 93, 94, 605, 629, 632
  - Unsupervised
    - attribute subset selection, 84
    - classification, 381
    - data mining tasks, 308
    - deep learning, 511, 582
    - feature learning, 554
    - learning, 13, 14, 242, 381, 425, 429, 509, 512, 555, 563, 653
    - method ToPMine, 233
    - outlier detection methods, 563
    - phrase mining, 226
    - phrase mining method ToPMine, 237
    - pretraining, 509, 512, 514
  - User
    - behavior, 432
    - feedback, 479
    - knowledge, 431
    - query, 17, 464
- V**
- Variance, 28, 31, 33, 34, 108, 136, 261, 284–286, 292, 309, 310, 405, 412, 419, 420, 425, 516, 558, 569, 595
  - Variational autoencoder (VAE), 556
  - Variational graph autoencoder (VGAE), 555
  - Vertical data format, 149, 160, 161, 169–171, 183, 201, 202, 204
  - Very fast decision tree (VFDT), 354
  - Visual question answering (VQA), 526
  - Visualizing spatial data warehouses, 143
- W**
- Warehouse database server, 88, 90, 133
  - WaveCluster, 430
  - Weak supervision, 242, 307, 342, 343, 364, 607, 629, 632, 633
  - Weakly supervised, 14, 226, 227, 233, 237, 242, 343, 348, 605, 609, 620, 631

Webpage

- classification, 355
- ranking, 622

Weight attribute, 63

Weighted Euclidean distance, 49

World Wide Web (WWW), 1

Wrapper, 84, 90, 311

- framework, 652
- methods, 308, 311, 370

Wrong predictions, 261, 530

**X**

XGBoost, 295, 296, 302, 652, 654

**Z**

z-score normalization, 64, 65, 82, 515, 579, 597

Zero-shot learning, 242, 343, 349, 351, 360, 369

This page intentionally left blank



This page intentionally left blank

# DATA MINING

CONCEPTS AND  
TECHNIQUES

FOURTH EDITION

JIAWEI HAN ■ JIAN PEI ■ HANGHANG TONG

**Data Mining: Concepts and Techniques, Fourth Edition** introduces concepts, principles, and methods for mining patterns, knowledge, and models from various kinds of data for diverse applications. Specifically, it delves into the processes for uncovering patterns and knowledge from massive collections of data, known as *knowledge discovery from data* or *KDD*. It focuses on the feasibility, usefulness, effectiveness, and scalability of data mining techniques for large data sets.

After an introduction to the concept of data mining, the authors explain the methods for preprocessing, characterizing, and warehousing data. Then they partition the data mining methods into several major tasks, introducing concepts and methods for mining frequent patterns, associations, and correlations for large data sets; data classification and model construction; cluster analysis; and outlier detection. Concepts and methods for deep learning are systematically introduced as one chapter. Finally, the book covers the trends, applications, and research frontiers in data mining.

This new edition:

- Presents a comprehensive new chapter on deep learning, including improving training of deep learning models, convolutional neural networks, recurrent neural networks, and graph neural networks.
- Addresses advanced topics in one dedicated chapter: data mining trends and research frontiers, including mining rich data types (text, spatiotemporal data, and graph/networks), data mining applications (such as sentiment analysis, truth discovery, and information propagation), data mining methodologies and systems, and data mining and society.
- Provides a comprehensive, practical look at the concepts and techniques needed to get the most out of your data.

This book is intended as a textbook on data mining for students in computer science, statistics, business, and data science and may serve as a reference book for application developers, business professionals, and researchers who enjoy learning concepts and principles of data mining.

## About the Authors

**Jiawei Han** is a Michael Aiken Chair Professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. He is a fellow of ACM and IEEE. He received the 2004 ACM SIGKDD Innovation Award.

**Jian Pei** is currently a professor at Duke University. He is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, ACM, and IEEE. He received the 2017 ACM SIGKDD Innovation Award and the 2015 ACM SIGKDD Service Award.

**Hanghang Tong** is currently an associate professor at the Department of Computer Science, University of Illinois at Urbana-Champaign. He is a distinguished member of ACM and a fellow of IEEE. He is the Editor-in-Chief of *SIGKDD Explorations* (ACM) and an associate editor of several journals.



ELSEVIER

MK

MORGAN KAUFMANN PUBLISHERS

An imprint of Elsevier

[elsevier.com/books-and-journals](http://elsevier.com/books-and-journals)

ISBN 978-0-12-811760-6



9 780128 117606