CHAPTER

Cluster analysis: basic concepts and methods

8

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.[1] 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

[1] Data similarity and dissimilarity are discussed in detail in Chapter 2. You may want to refer to the corresponding section for a quick review.

Data Mining. https://doi.org/10.1016/B978-0-12-811760-6.00018-7

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-

sures 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, $C_1, \ldots, C_k$, 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 $\boldsymbol{p} \in C_i$ and $\boldsymbol{c_i}$, the representative of the cluster, is measured by $dist(\boldsymbol{p}, \boldsymbol{c_i})$, where $dist(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance between two points $\boldsymbol{x}$ and $\boldsymbol{y}$. 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 $\boldsymbol{c_i}$, defined as

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} dist(\boldsymbol{p}, \boldsymbol{c_i})^2, \tag{8.1}$$

where E is the sum of the squared error for all objects in the data set; $\boldsymbol{p}$ is the point in space representing a given object; and $\boldsymbol{c_i}$ is the centroid of cluster C_i (both $\boldsymbol{p}$ and $\boldsymbol{c_i}$ 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:

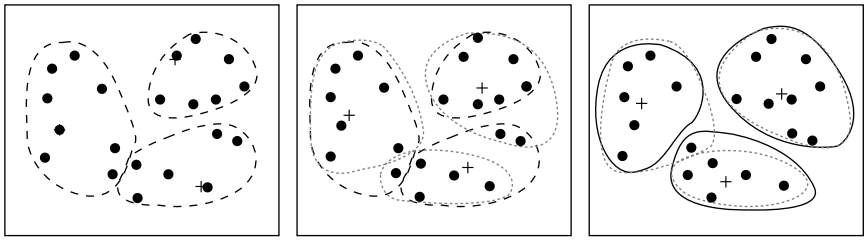
- k: the number of clusters,
- D: a data set containing n objects.

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.



(a) Initial clustering **(b)** Iterate **(c)** Final clustering

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. □

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 many 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

$$\begin{aligned}(1-3.5)^2 + (2-3.5)^2 + (3-3.5)^2 + (8-3.5)^2 + (9-14.67)^2 \\ + (10-14.67)^2 + (25-14.67)^2 = 189.67,\end{aligned}$$

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. □

"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 $\boldsymbol{p}$ and its corresponding representative object. That is, an **absolute-error criterion** is used, defined as

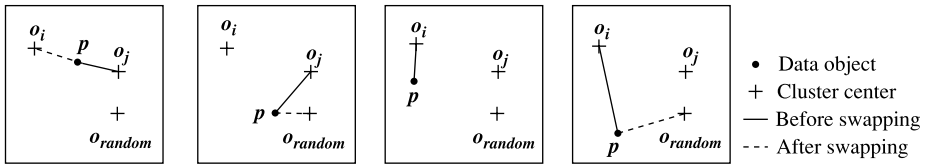
$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} dist(\boldsymbol{p}, \boldsymbol{o}_i), \tag{8.2}$$

where E is the sum of the absolute error for all objects $\boldsymbol{p}$ in the data set, and $\boldsymbol{o_i}$ 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 $\boldsymbol{o_1}, \ldots, \boldsymbol{o_k}$ be the current set of representative objects (i.e., medoids). To determine whether a nonrepresentative object, denoted by $\boldsymbol{o_{random}}$, is a good replacement for a current medoid



(a) Reassigned to o_i **(b)** Reassigned to o_{random} **(c)** No change **(d)** Reassigned to o_{random}

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 $\{o_1, \ldots, o_{j-1}, o_{random}, o_{j+1}, \ldots, o_k\}$, and use the distance to update the cost function. The reassignments of objects to $\{o_1, \ldots, o_{j-1}, o_{random}, o_{j+1}, \ldots, o_k\}$ 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 $\boldsymbol{x} = (x_1, \ldots, x_l)$ and $\boldsymbol{y} = (y_1, \ldots, y_l)$, we define the distance

$$dist(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} d(x_i, y_i), \tag{8.3}$$

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 $\boldsymbol{c_i}$ 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

$$CH(k) = \frac{\frac{B(k)}{k-1}}{\frac{W(k)}{n-k}}. \tag{8.4}$$

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

$$SD_{C_i} = \sum_{\boldsymbol{p}, \boldsymbol{q} \in C_i} dist(\boldsymbol{p}, \boldsymbol{q}).$$

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

$$Gap_n(k) = E_n^*\{\log(W_k)\} - \log(W_k), \tag{8.5}$$

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.

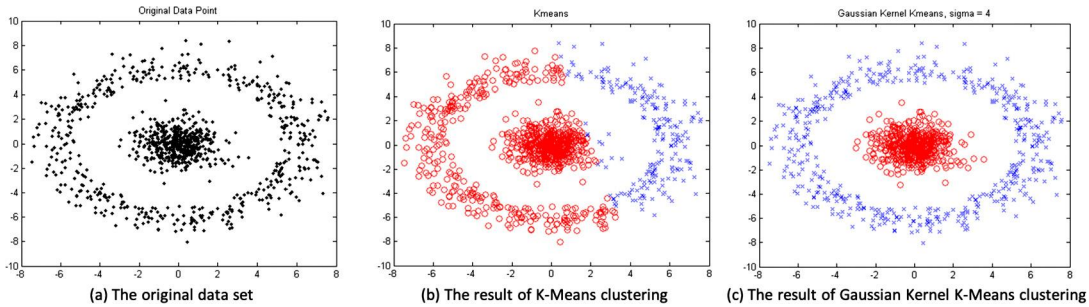


FIGURE 8.5

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 σ 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 ith row and the jth 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

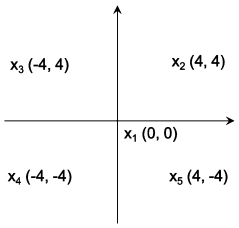


FIGURE 8.6

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

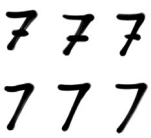


FIGURE 8.7

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-

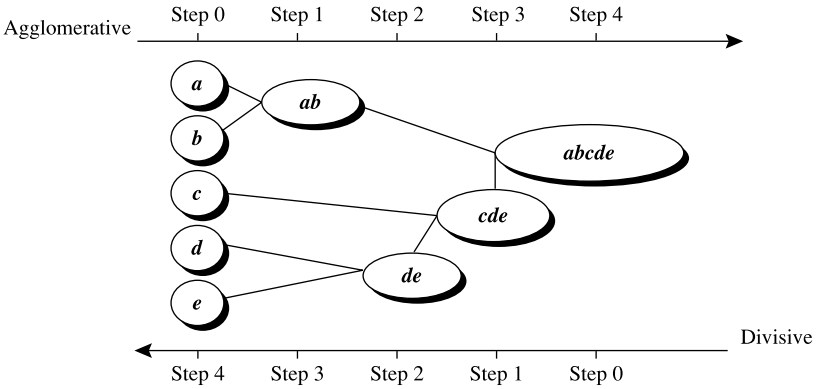


FIGURE 8.8

Agglomerative and divisive hierarchical clustering on data objects $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$.

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, $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$. 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. □

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 $|\boldsymbol{p} - \boldsymbol{p'}|$ is the distance between two objects or points, $\boldsymbol{p}$ and $\boldsymbol{p'}$; $\boldsymbol{m_i}$ 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*.

$$\textbf{Minimum distance}: \quad dist_{min}(C_i, C_j) = \min_{\boldsymbol{p} \in C_i, \boldsymbol{p'} \in C_j} \{\|\boldsymbol{p} - \boldsymbol{p'}\|\} \tag{8.6}$$

$$\textbf{Maximum distance}: \quad dist_{max}(C_i, C_j) = \max_{\boldsymbol{p} \in C_i, \boldsymbol{p'} \in C_j} \{\|\boldsymbol{p} - \boldsymbol{p'}\|\} \tag{8.7}$$

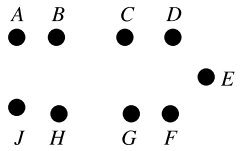
$$\textbf{Mean distance}: \quad dist_{mean}(C_i, C_j) = \|\boldsymbol{m_i} - \boldsymbol{m_j}\| \tag{8.8}$$

$$\textbf{Average distance}: \quad dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\boldsymbol{p} \in C_i, \boldsymbol{p'} \in C_j} \|\boldsymbol{p} - \boldsymbol{p'}\| \tag{8.9}$$

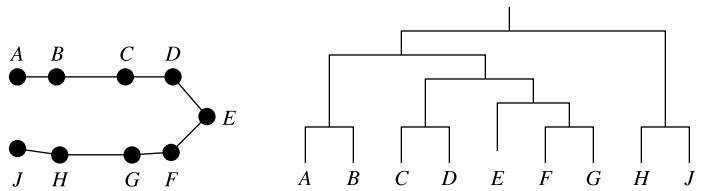
When an algorithm uses the *minimum distance*, $d_{min}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **nearest-neighbor clustering algorithm** or **single-linkage algorithm**. If we view the data points as nodes of a graph, with edges forming a path between the nodes 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.

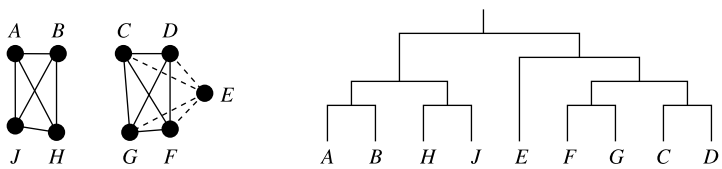
Similar to the situation in the k-means method, single-linkage and complete-linkage methods are sensitive to outliers. The use of *mean* or *average distance* is a compromise between the minimum and maximum distances and overcomes the outlier sensitivity problem. Whereas the *mean distance* is the simplest to compute, the *average distance* is advantageous in that it can handle categoric data and



(a) Data set



(b) Clustering using single linkage



(c) Clustering using complete linkage

FIGURE 8.9

Hierarchical clustering using single and complete linkages.

numeric data. The computation of the mean vector for categoric 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. □

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,

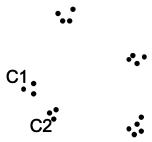


FIGURE 8.10

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

$$\begin{aligned} W(C_i, C_j) &= \sum_{\mathbf{x} \in C_i \cup C_j} \|\mathbf{x} - \boldsymbol{m}_{(ij)}\|^2 - \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{m}_i\|^2 - \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \boldsymbol{m}_j\|^2 \\ &= \frac{n_i n_j}{n_i + n_j} \|\boldsymbol{m}_i - \boldsymbol{m}_j\|^2. \end{aligned}$$

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 α_i, α_j, β, and γ 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, $E_{C_i} = \frac{1}{|C_i|}\sum_{\mathbf{x}\in C_i}(\mathbf{x} - \boldsymbol{m_i})^2$, 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

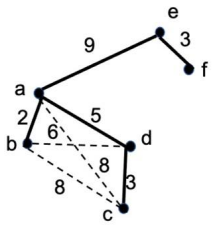


FIGURE 8.11

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 $\boldsymbol{a}$ and $\boldsymbol{b}$ 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, $\{\boldsymbol{a}, \boldsymbol{b}\}$ and $\{\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$, 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

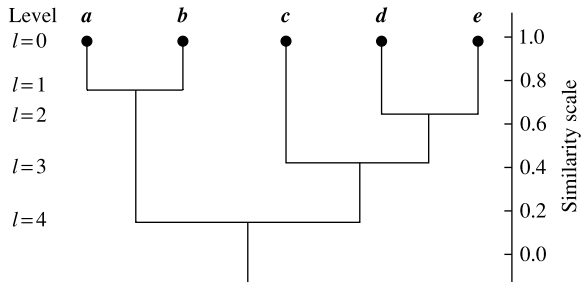


FIGURE 8.12

Dendrogram representation for hierarchical clustering of data objects $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$.

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

$$\boldsymbol{CF} = \langle n, \boldsymbol{LS}, SS \rangle, \tag{8.10}$$

where $\boldsymbol{LS}$ is the linear sum of the n points (i.e., $\boldsymbol{LS} = \sum_{i=1}^{n} \boldsymbol{x_i}$) and SS is the square sum of the data points (i.e., $SS = \sum_{i=1}^{n} \|\boldsymbol{x}_i\|^2$).

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, $\boldsymbol{x_0}$, radius, R, and diameter, D, are

$$\boldsymbol{x}_0 = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i}{n} = \frac{\boldsymbol{LS}}{n}, \tag{8.11}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n}(\boldsymbol{x_i} - \boldsymbol{x_0})^2}{n}} = \sqrt{\frac{SS}{n} - (\frac{\|\boldsymbol{LS}\|}{n})^2}, \tag{8.12}$$

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{x_i} - \boldsymbol{x_j})^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2\|\boldsymbol{LS}\|^2}{n(n-1)}}. \tag{8.13}$$

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 $\boldsymbol{CF}_1 = \langle n_1, \boldsymbol{LS}_1, SS_1\rangle$ and $\boldsymbol{CF}_2 = \langle n_2, \boldsymbol{LS}_2, SS_2\rangle$, respectively, the clustering feature for the cluster that formed by merging C_1 and C_2 is simply

$$\boldsymbol{CF}_1 + \boldsymbol{CF}_2 = \langle n_1 + n_2, \boldsymbol{LS}_1 + \boldsymbol{LS}_2, SS_1 + SS_2\rangle. \tag{8.14}$$

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

$$\boldsymbol{CF}_1 = \langle 3, (2+3+4, 5+2+3), (2^2+3^2+4^2) + (5^2+2^2+3^2)\rangle = \langle 3, (9, 10), 67\rangle.$$

Suppose that C_1 is disjoint to a second cluster, C_2, where $\boldsymbol{CF}_2 = \langle 3, (35, 36), 857\rangle$. The clustering feature of a new cluster, C_3, that is formed by merging C_1 and C_2, is derived by adding $\boldsymbol{CF}_1$ and $\boldsymbol{CF}_2$. That is,

$$\boldsymbol{CF}_3 = \langle 3+3, (9+35, 10+36), 67+857\rangle = \langle 6, (44, 46), 924\rangle.$$

□

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.

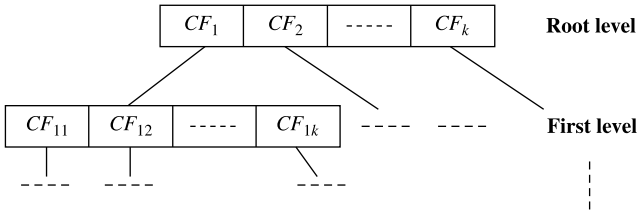


FIGURE 8.13

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 $X = \{x_1, \ldots, x_n\}$ for clustering analysis. Let us assume that the data points are generated by a Gaussian distribution,

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{8.15}$$

where the parameters are μ (the mean) and σ^2 (the variance).

The probability that a point $x_i \in X$ is then generated by the model is

$$P(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{8.16}$$

Consequently, the likelihood that the data set X observed is generated by the model is

$$L(\mathcal{N}(\mu, \sigma^2) : X) = P(X|\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{8.17}$$

The task of learning the generative model is to find the parameters μ and σ^2 such that the likelihood $L(\mathcal{N}(\mu, \sigma^2) : X)$ is maximized, that is, finding

$$\mathcal{N}(\mu_0, \sigma_0^2) = \arg\max\{L(\mathcal{N}(\mu, \sigma^2) : X)\}, \tag{8.18}$$

where $\max\{L(\mathcal{N}(\mu, \sigma^2) : X)\}$ is called the *maximum likelihood*. □

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 $C_1, \ldots, C_m$. Then the quality can

be measured by

$$Q(\{C_1, \ldots, C_m\}) = \prod_{i=1}^{m} P(C_i), \tag{8.19}$$

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

$$\begin{aligned} &Q((\{C_1, \ldots, C_m\} - \{C_{j_1}, C_{j_2}\}) \cup \{C_{j_1} \cup C_{j_2}\}) - Q(\{C_1, \ldots, C_m\}) \\ &= \frac{\prod_{i=1}^{m} P(C_i) \cdot P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - \prod_{i=1}^{m} P(C_i) \\ &= \prod_{i=1}^{m} P(C_i) \left(\frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - 1 \right). \end{aligned} \tag{8.20}$$

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

$$dist(C_1, C_2) = -\log \frac{P(C_1 \cup C_2)}{P(C_1)P(C_2)}. \tag{8.21}$$

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

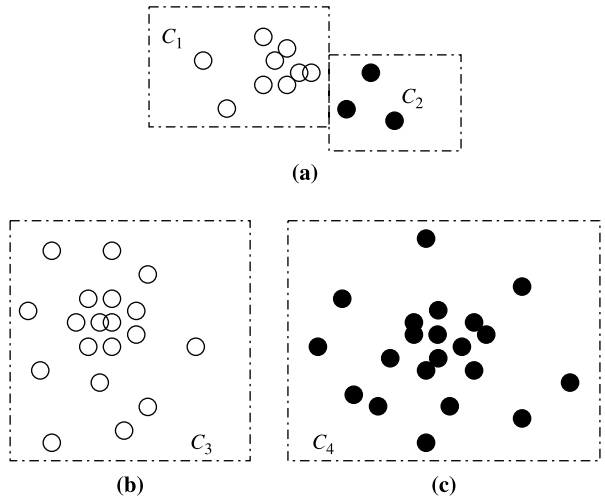


FIGURE 8.14

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:

- $D = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}\}$: a data set containing n objects;

Output: A hierarchy of clusters.
Method:

(1) **create** a cluster for each object $C_i = \{\boldsymbol{o_i}\}$, $1 \leq i \leq n$;
(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.