10 Cluster Analysis: Basic Concepts and Methods

Imagine that you are the Director of Customer Relationships at *AllElectronics*, and you have five managers working for you. You would like to organize all the company's customers into five 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. Moreover, two given 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 per group. What kind of data mining techniques can help you to accomplish this task?

Unlike in classification, the class label (or *group_ID*) of each customer is unknown. You need to *discover* these groupings. Given a large number of customers and many attributes describing customer profiles, it can be very costly or even infeasible to have a human study the data and manually 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 very 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 10.1, we introduce the topic and study the requirements of clustering methods for massive amounts of data and various applications. You will learn several basic clustering techniques, organized into the following categories: *partitioning methods* (Section 10.2), *hierarchical methods* (Section 10.3), *density-based methods* (Section 10.4), and *grid-based methods* (Section 10.5). In Section 10.6, we briefly discuss how to evaluate

[1] Data similarity and dissimilarity are discussed in detail in Section 2.4. You may want to refer to that section for a quick review.

clustering methods. A discussion of advanced methods of clustering is reserved for Chapter 11.

10.1 Cluster Analysis

This section sets up the groundwork for studying cluster analysis. Section 10.1.1 defines cluster analysis and presents examples of where it is useful. In Section 10.1.2, you will learn aspects for comparing clustering methods, as well as requirements for clustering. An overview of basic clustering techniques is presented in Section 10.1.3.

10.1.1 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 partitioning is not performed by humans, but by the 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. 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, clustering can be applied to partition projects into categories based on similarity so that project auditing and diagnosis (to improve project delivery and outcomes) can be conducted effectively.

In image recognition, clustering can be used to discover clusters or "subclasses" in handwritten character recognition systems. Suppose we have a data set of handwritten digits, where each digit is labeled as either 1, 2, 3, and so on. Note that there can be a large variance in the way in which people write the same digit. Take the number 2, for example. Some people may write it with a small circle at the left bottom part, while some others may not. We can use clustering to determine subclasses for "2," each of which represents a variation on the way in which 2 can be written. Using multiple models based on the subclasses can improve overall recognition accuracy.

Clustering has also found many applications in Web search. For example, a keyword search may often return a very 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**. 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 fraud and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and infrequent purchases, may be of interest as possible fraudulent activities. Outlier detection is the subject of Chapter 12.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine 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 recently 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 databases*. 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 of features), and methods for clustering *mixed numerical and nominal data* in large databases.

10.1.2 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.

The following are typical requirements of clustering in data mining.

- **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, particularly 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.
- **Ability to deal with different types of attributes**: Many algorithms are designed to cluster numeric (interval-based) data. However, applications may require clustering other data types, such as binary, nominal (categorical), and ordinal data, or mixtures of these data types. Recently, more and more applications need clustering techniques for complex data types such as graphs, sequences, images, and documents.
- **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 measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. Consider sensors, for example, which are often deployed for environment surveillance. Cluster analysis on sensor readings can detect interesting phenomena. We may want to use clustering to find the frontier of a running forest fire, 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 high-dimensionality data sets and 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.
- **Ability to deal with noisy data**: Most real-world data sets contain outliers and/or missing, unknown, or erroneous data. Sensor readings, for example, are often noisy—some readings may be inaccurate due to the sensing mechanisms, and some readings may be erroneous due to interferences from surrounding transient objects. 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 job is to choose the locations for a given number of new automatic teller machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks and the types and number of customers per cluster. A challenging task is to find data groups with good clustering behavior 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 in 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.

The following are orthogonal aspects with which clustering methods can be compared:

- **The partitioning criteria**: In some 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. When clustering customers into groups so that each group is taken care of by one manager, each customer may belong to only one group. 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 Euclidean space,

a road network, a vector space, or any other space. In other methods, the similarity may be defined by connectivity based on density or contiguity, and 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 optimization techniques, density- and continuity-based methods can often find clusters of arbitrary shape.

- **Clustering space**: 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's often better to instead search for clusters within different subspaces of the same data set. *Subspace clustering* discovers clusters and subspaces (often of low dimensionality) that manifest object similarity.

To conclude, clustering algorithms have several requirements. These factors include scalability and the ability to deal with different types of attributes, 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 the partitioning level, whether or not clusters are mutually exclusive, the similarity measures used, and whether or not subspace clustering is performed.

10.1.3 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 partitions of the data, where each partition represents a cluster and $k \leq n$. That is, it divides the data into k groups such that each group must contain at least one object. In other words, partitioning methods conduct 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. References to such techniques are given in the bibliographic notes (Section 10.9).

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 are 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-size databases. 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 10.2.

Hierarchical methods: A hierarchical method creates a hierarchical decomposition of the 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-based or density- and 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 undone. 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 10.3.

Density-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. Density-based clustering methods are studied in Section 10.4.

Grid-based methods: Grid-based methods 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). The main advantage of this approach is its 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. Therefore, grid-based methods can be integrated with other clustering methods such as density-based methods and hierarchical methods. Grid-based clustering is studied in Section 10.5.

These methods are briefly summarized in Figure 10.1. 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 the integration of several clustering techniques.

In the following sections, we examine each clustering method in detail. Advanced clustering methods and related issues are discussed in Chapter 11. In general, the notation used is as follows. Let D be a data set of n objects to be clustered. An object is described by d variables, where each variable is also called an attribute or a dimension,

Method	General Characteristics
Partitioning methods	– Find mutually exclusive clusters of spherical shape – Distance-based – May use mean or medoid (etc.) to represent cluster center – Effective for small- to medium-size data sets
Hierarchical methods	– Clustering is a hierarchical decomposition (i.e., multiple levels) – Cannot correct erroneous merges or splits – May incorporate other techniques like microclustering or consider object "linkages"
Density-based methods	– Can find arbitrarily shaped clusters – Clusters are dense regions of objects in space that are separated by low-density regions – Cluster density: Each point must have a minimum number of points within its "neighborhood" – May filter out outliers
Grid-based methods	– Use a multiresolution grid data structure – Fast processing time (typically independent of the number of data objects, yet dependent on grid size)

Figure 10.1 Overview of clustering methods discussed in this chapter. Note that some algorithms may combine various methods.

and therefore may also be referred to as a *point* in a d-dimensional object space. Objects are represented in bold italic font (e.g., $\boldsymbol{p}$).

10.2 Partitioning Methods

The simplest and most fundamental version of cluster analysis is partitioning, which organizes the objects of a set into several exclusive groups or clusters. To keep the problem specification concise, we can assume that the number of clusters is given as background knowledge. This parameter is the starting point for partitioning methods.

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 objects in other clusters in terms of the data set attributes.

In this section you will learn the most well-known and commonly used partitioning methods—*k*-means (Section 10.2.1) and *k*-medoids (Section 10.2.2). You will also learn several variations of these classic partitioning methods and how they can be scaled up to handle large data sets.

10.2.1 *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, $C_i \subset D$ and $C_i \cap C_j = \emptyset$ for $(1 \leq i, j \leq k)$. 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, to represent 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

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(\boldsymbol{p}, \boldsymbol{c_i})^2, \tag{10.1}$$

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 as separate as possible.

Optimizing the within-cluster variation is computationally challenging. In the worst case, we would have to enumerate a number of possible partitionings that are exponential to the number of clusters, and check the within-cluster variation values. It has been shown that the problem is NP-hard in general Euclidean space even for two clusters (i.e., $k = 2$). Moreover, the problem is NP-hard for a general number of clusters k even in the 2-D Euclidean space. If the number of clusters k and the dimensionality of the space d are fixed, the problem can be solved in time $O(n^{dk+1} \log n)$, where n is the number of objects. 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 of the objects in 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 cluster mean. 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 Figure 10.2.

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:

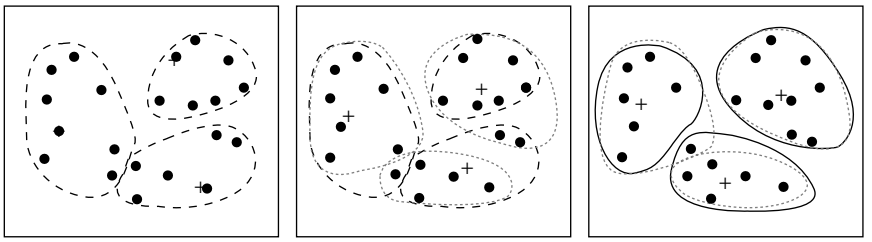
- k: the number of clusters,
- D: a data set containing n objects.

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, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

Figure 10.2 The k-means partitioning algorithm.



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

Figure 10.3 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 +).

Example 10.1 Clustering by *k*-means partitioning. Consider a set of objects located in 2-D space, as depicted in Figure 10.3(a). Let $k = 3$, that is, the user would like the objects to be partitioned into three clusters.

According to the algorithm in Figure 10.2, 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 center to which it is the nearest. Such a distribution forms silhouettes encircled by dotted curves, as shown in Figure 10.3(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 redistributed to the clusters based on which cluster center is the nearest. Such a redistribution forms new silhouettes encircled by dashed curves, as shown in Figure 10.3(b).

This process iterates, leading to Figure 10.3(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 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.

There are several variants of the *k*-means method. These can differ in the selection of the initial *k*-means, the calculation of dissimilarity, and the strategies for calculating cluster means.

The *k*-means method 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 are 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. It uses new dissimilarity measures to deal with nominal objects and a frequency-based method to update modes of clusters. The *k*-means and the *k*-modes methods can be integrated to cluster data with mixed numeric and nominal values.

The necessity for users to specify *k*, the number of clusters, in advance can be seen as a disadvantage. There have been studies on how to overcome this difficulty, however, such as by providing an approximate range of *k* values, and then using an analytical technique to determine the best *k* by comparing the clustering results obtained for the different *k* values. The *k*-means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size. Moreover, it is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value.

"*How can we make the* k*-means algorithm more scalable?*" One approach to making the *k*-means method more efficient on large data sets is to use a good-sized set of samples in clustering. Another is to employ a filtering approach that uses a spatial hierarchical data index to save costs when computing means. A third approach explores the microclustering idea, which first groups nearby objects into "microclusters" and then performs *k*-means clustering on the microclusters. Microclustering is further discussed in Section 10.3.

10.2.2 *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 of Eq. (10.1), as observed in Example 10.2.

Example 10.2 A drawback of *k*-means. Consider six points in 1-D space having the 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. (10.1), 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 lowest within-cluster variation; therefore, the *k*-means method assigns the value 8 to a cluster different from that containing 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

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} dist(\boldsymbol{p}, \boldsymbol{o_i}), \tag{10.2}$$

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. 10.2).

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 Figure 10.5 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 $\boldsymbol{o_j}$ $(1 \leq j \leq k)$, we calculate the distance from every object $\boldsymbol{p}$ to the closest object in the set $\{\boldsymbol{o_1}, \ldots, \boldsymbol{o_{j-1}}, \boldsymbol{o_{random}}, \boldsymbol{o_{j+1}}, \ldots, \boldsymbol{o_k}\}$, and use the distance to update the cost function. The reassignments of objects to $\{\boldsymbol{o_1}, \ldots, \boldsymbol{o_{j-1}}, \boldsymbol{o_{random}}, \boldsymbol{o_{j+1}}, \ldots, \boldsymbol{o_k}\}$ are simple. Suppose object $\boldsymbol{p}$ is currently assigned to a cluster represented by medoid $\boldsymbol{o_j}$ (Figure 10.4a or b). Do we need to reassign $\boldsymbol{p}$ to a different cluster if $\boldsymbol{o_j}$ is being replaced by $\boldsymbol{o_{random}}$? Object $\boldsymbol{p}$ needs to be reassigned to either $\boldsymbol{o_{random}}$ or some other cluster represented by $\boldsymbol{o_i}$ $(i \neq j)$, whichever is the closest. For example, in Figure 10.4(a), $\boldsymbol{p}$ is closest to $\boldsymbol{o_i}$ and therefore is reassigned to $\boldsymbol{o_i}$. In Figure 10.4(b), however, $\boldsymbol{p}$ is closest to $\boldsymbol{o_{random}}$ and so is reassigned to $\boldsymbol{o_{random}}$. What if, instead, $\boldsymbol{p}$ is currently assigned to a cluster represented by some other object $\boldsymbol{o_i}$, $i \neq j$?

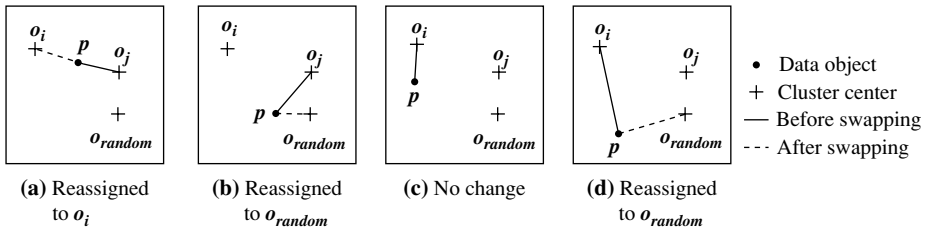


Figure 10.4 Four cases of the cost function for k-medoids clustering.

Object $\boldsymbol{o}$ remains assigned to the cluster represented by $\boldsymbol{o_i}$ as long as $\boldsymbol{o}$ is still closer to $\boldsymbol{o_i}$ than to $\boldsymbol{o_{random}}$ (Figure 10.4c). Otherwise, $\boldsymbol{o}$ is reassigned to $\boldsymbol{o_{random}}$ (Figure 10.4d).

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 $\boldsymbol{o_j}$ is replaced or swapped with $\boldsymbol{o_{random}}$ because the actual absolute-error E is reduced. If the total cost is positive, the current representative object, $\boldsymbol{o_j}$, 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.

"*How can we scale up the k-medoids method?*" A typical k-medoids partitioning algorithm like PAM (Figure 10.5) works effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method called **CLARA** (**Clustering LARge Applications**) can be used. Instead of taking the whole data set into consideration, CLARA uses a random sample of the data set. The PAM algorithm is then applied to compute the best medoids from the sample. Ideally, the sample should closely represent the original data set. In many cases, a large sample works well if it is created so that each object has equal probability of being selected into the sample. The representative objects (medoids) chosen will likely be similar to those that would have been chosen from the whole data set. CLARA builds clusterings from multiple random samples and returns the best clustering as the output. The complexity of computing the medoids on a random sample is $O(ks^2 + k(n-k))$, where s is the size of the sample, k is the number of clusters, and n is the total number of objects. CLARA can deal with larger data sets than PAM.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best k-medoids among a given data set, whereas CLARA searches for the best k-medoids among the *selected sample* of the data set. CLARA cannot find a good clustering if any of the best sampled medoids is far from the best k-medoids. If an object

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 10.5 PAM, a *k*-medoids partitioning algorithm.

is one of the best *k*-medoids but is not selected during sampling, CLARA will never find the best clustering. (You will be asked to provide an example demonstrating this as an exercise.)

"*How might we improve the quality and scalability of CLARA?*" Recall that when searching for better medoids, PAM examines every object in the data set against every current medoid, whereas CLARA confines the candidate medoids to only a random sample of the data set. A randomized algorithm called **CLARANS** (Clustering Large Applications based upon RANdomized Search) presents a trade-off between the cost and the effectiveness of using samples to obtain clustering.

First, it randomly selects *k* objects in the data set as the current medoids. It then randomly selects a current medoid $\boldsymbol{x}$ and an object $\boldsymbol{y}$ that is not one of the current medoids. Can replacing $\boldsymbol{x}$ by $\boldsymbol{y}$ improve the absolute-error criterion? If yes, the replacement is made. CLARANS conducts such a randomized search *l* times. The set of the current medoids after the *l* steps is considered a local optimum. CLARANS repeats this randomized process *m* times and returns the best local optimal as the final result.

10.3 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 such as in a hierarchy. A **hierarchical clustering method** works by grouping data objects into a hierarchy or "tree" of clusters.

Representing data objects in the form of a hierarchy is useful for data summarization and visualization. For example, as the manager of human resources at *AllElectronics*,

you may organize your employees 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 are 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. If necessary, the hierarchical partitioning can be continued recursively until a desired granularity is reached.

In the previous examples, although we partitioned the data hierarchically, we did not assume that the data have a hierarchical structure (e.g., managers are at the same level in our *AllElectronics* hierarchy as staff). 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 *AllElectronics* employees structured on, say, salary. In the study of evolution, hierarchical clustering may group animals 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.

In this section, you will study hierarchical clustering methods. Section 10.3.1 begins with a discussion of agglomerative versus divisive hierarchical clustering, which organize objects into a hierarchy using a bottom-up or top-down strategy, respectively. Agglomerative methods start with individual objects as clusters, which are iteratively merged to form larger clusters. Conversely, divisive methods initially let all the given objects form one cluster, which they iteratively split into smaller clusters.

Hierarchical clustering methods can encounter difficulties regarding the selection of merge or split points. Such a decision is critical, 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** (or **multiphase**) **clustering**. We introduce two such methods, namely BIRCH and Chameleon. BIRCH (Section 10.3.3) 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. Chameleon (Section 10.3.4) explores dynamic modeling in hierarchical clustering.

There are several orthogonal ways to categorize hierarchical clustering methods. For instance, they may be categorized into *algorithmic* methods, *probabilistic* methods, and *Bayesian* methods. Agglomerative, divisive, and multiphase methods are *algorithmic*, meaning 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 10.3.5. *Bayesian methods* compute a distribution of possible clusterings. That is, instead of outputting a single deterministic clustering over a data set, they return a group of clustering structures and their probabilities, conditional on the given data. Bayesian methods are considered an advanced topic and are not discussed in this book.

10.3.1 Agglomerative versus Divisive Hierarchical Clustering

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's 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 continues 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 10.3 **Agglomerative versus divisive hierarchical clustering.** Figure 10.6 shows the application of **AGNES** (AGglomerative NESting), an agglomerative hierarchical clustering method, and **DIANA** (DIvisive ANAlysis), a divisive hierarchical clustering method, on a data set of five objects, $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$. Initially, AGNES, 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

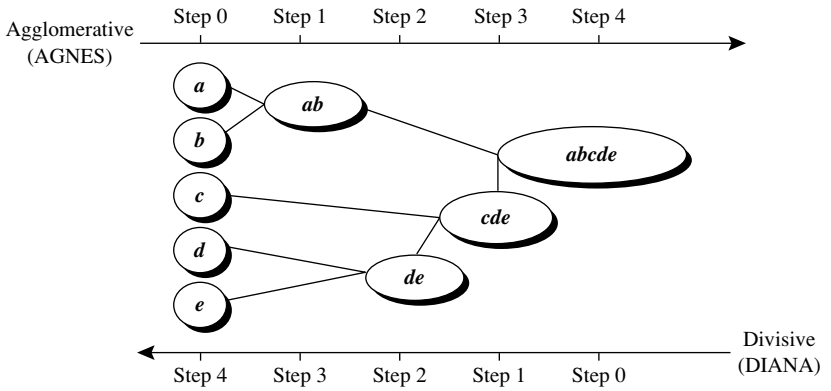


Figure 10.6 Agglomerative and divisive hierarchical clustering on data objects $\{a, b, c, d, e\}$.

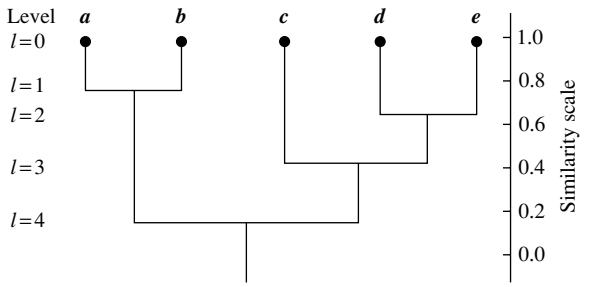


Figure 10.7 Dendrogram representation for hierarchical clustering of data objects $\{a, b, c, d, e\}$.

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.

DIANA, 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. ■

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. Figure 10.7 shows a dendrogram for the five objects presented in Figure 10.6, where $l = 0$ shows the five objects as singleton clusters at level 0. At $l = 1$, objects ***a*** and ***b*** are grouped together to form the

first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects, $\{\boldsymbol{a}, \boldsymbol{b}\}$ and $\{\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$, is roughly 0.16, they are merged together to form a single cluster.

A challenge with divisive methods is how to partition a large cluster into several smaller ones. For example, there are $2^{n-1} - 1$ possible ways to partition a set of n objects into two exclusive subsets, where n is the number of objects. When n is large, it is computationally prohibitive to examine all possibilities. Consequently, a divisive method typically uses heuristics in partitioning, which can lead to inaccurate results. For the sake of efficiency, divisive methods typically do not backtrack on partitioning decisions that have been made. Once a cluster is partitioned, any alternative partitioning of this cluster will not be considered again. Due to the challenges in divisive methods, there are many more agglomerative methods than divisive methods.

10.3.2 Distance Measures in Algorithmic Methods

Whether using an agglomerative method or a divisive method, a core need is to measure the distance 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*.

$$\textbf{Minimum distance:} \quad dist_{min}(C_i, C_j) = \min_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \tag{10.3}$$

$$\textbf{Maximum distance:} \quad dist_{max}(C_i, C_j) = \max_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \tag{10.4}$$

$$\textbf{Mean distance:} \quad dist_{mean}(C_i, C_j) = |\boldsymbol{m_i} - \boldsymbol{m_j}| \tag{10.5}$$

$$\textbf{Average distance:} \quad dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \tag{10.6}$$

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**. Moreover, if the clustering process is terminated when the distance between nearest clusters exceeds a user-defined threshold, it is called a **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. Because edges linking clusters always go between distinct clusters, the resulting graph will generate a tree. Thus, an agglomerative hierarchical clustering algorithm that uses the minimum distance measure is also called a

minimal spanning tree algorithm, where a spanning tree of a graph is a tree that connects all vertices, and a minimal spanning tree is the one with the least sum of edge weights.

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**. If the clustering process is terminated when the maximum distance between nearest clusters exceeds a user-defined threshold, it is called a **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. Farthest-neighbor algorithms tend to minimize the increase in diameter of the clusters at each iteration. If the true clusters are rather compact and approximately equal size, the method will produce high-quality clusters. Otherwise, the clusters produced can be meaningless.

The previous minimum and maximum measures represent two extremes in measuring the distance between clusters. They tend to be overly sensitive to outliers or noisy data. 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 as well as numeric data. The computation of the mean vector for categoric data can be difficult or impossible to define.

Example 10.4 **Single versus complete linkages.** Let us apply hierarchical clustering to the data set of Figure 10.8(a). Figure 10.8(b) shows the dendrogram using single linkage. Figure 10.8(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.

10.3.3 BIRCH: Multiphase Hierarchical Clustering Using Clustering Feature Trees

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) 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). It overcomes the two difficulties in agglomerative clustering methods: (1) scalability and (2) the inability to undo what was done in the previous step.

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

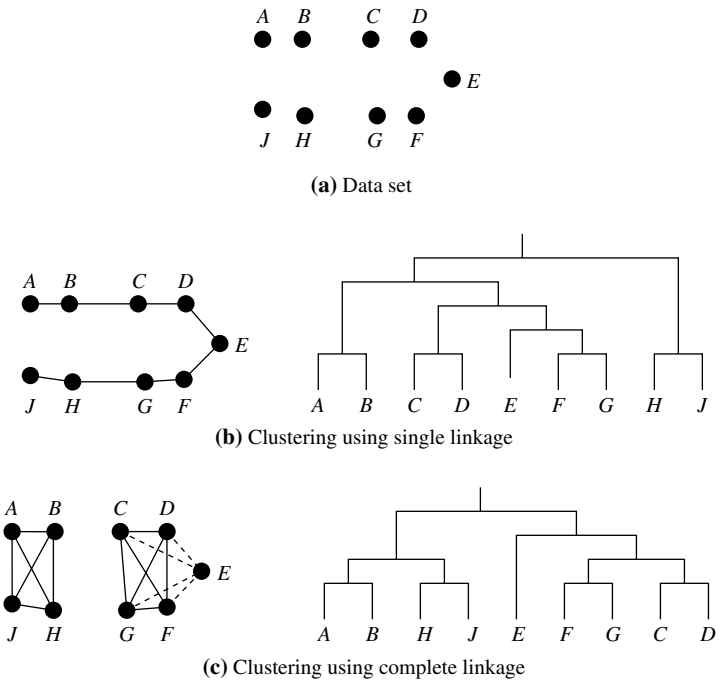


Figure 10.8 Hierarchical clustering using single and complete linkages.

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-dimensional data objects or points. The **clustering feature** (**CF**) of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as

$$CF = \langle n, LS, SS \rangle, \tag{10.7}$$

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

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

$$\boldsymbol{x_0} = \frac{\sum_{i=1}^{n} \boldsymbol{x_i}}{n} = \frac{LS}{n}, \tag{10.8}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n}(\boldsymbol{x_i} - \boldsymbol{x_0})^2}{n}} = \sqrt{\frac{nSS - 2LS^2 + nLS}{n^2}}, \tag{10.9}$$

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{x_i} - \boldsymbol{x_j})^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}. \tag{10.10}$$

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 BIRCH efficiency in space. Moreover, clustering features are *additive*. That is, for two disjoint clusters, C_1 and C_2, with the clustering features $CF_1 = \langle n_1, LS_1, SS_1\rangle$ and $CF_2 = \langle n_2, LS_2, SS_2\rangle$, respectively, the clustering feature for the cluster that formed by merging C_1 and C_2 is simply

$$CF_1 + CF_2 = \langle n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2\rangle. \tag{10.11}$$

Example 10.5 **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

$$CF_1 = \langle 3, (2+3+4, 5+2+3), (2^2+3^2+4^2, 5^2+2^2+3^2)\rangle = \langle 3, (9,10), (29,38)\rangle.$$

Suppose that C_1 is disjoint to a second cluster, C_2, where $CF_2 = \langle 3, (35,36), (417,440)\rangle$. The clustering feature of a new cluster, C_3, that is formed by merging C_1 and C_2, is derived by adding CF_1 and CF_2. That is,

$$CF_3 = \langle 3+3, (9+35, 10+36), (29+417, 38+440)\rangle = \langle 6, (44,46), (446,478)\rangle. \quad \blacksquare$$

A **CF-tree** is a height-balanced tree that stores the clustering features for a hierarchical clustering. An example is shown in Figure 10.9. 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 resulting tree's size.

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

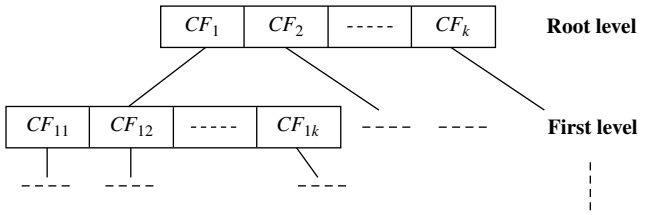


Figure 10.9 CF-tree structure.

one or more additional scans can optionally be used to further improve the quality. The primary phases are

- **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 data's inherent clustering 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, 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 has 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 the 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.

10.3.4 Chameleon: Multiphase Hierarchical Clustering Using Dynamic Modeling

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. In Chameleon, cluster similarity is assessed based on (1) how well connected objects are within a cluster and (2) the proximity of clusters. That is, two clusters are merged if their *interconnectivity* is high and they are *close together*. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all data types as long as a similarity function can be specified.

Figure 10.10 illustrates how Chameleon works. Chameleon uses a k-nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the k-most similar objects to the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the k-nearest-neighbor graph into a large number of relatively small subclusters such that it minimizes the **edge cut**. That is, a cluster C is partitioned into subclusters C_i and C_j so as to minimize the *weight of the edges* that would be cut should C be bisected into C_i and C_j. It assesses the *absolute* interconnectivity between clusters C_i and C_j.

Chameleon then uses an agglomerative hierarchical clustering algorithm that iteratively merges subclusters based on their similarity. To determine the pairs of most similar subclusters, it takes into account both the interconnectivity and the closeness of the clusters. Specifically, Chameleon determines the similarity between each pair of clusters C_i and C_j according to their *relative interconnectivity*, $RI(C_i, C_j)$, and their *relative closeness*, $RC(C_i, C_j)$.

- The **relative interconnectivity**, $RI(C_i, C_j)$, between two clusters, C_i and C_j, is defined as the absolute interconnectivity between C_i and C_j, normalized with respect to the

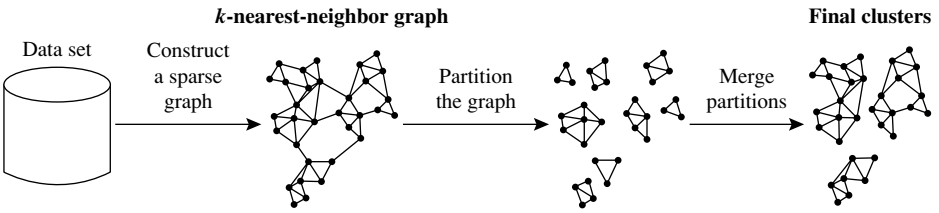


Figure 10.10 Chameleon: hierarchical clustering based on k-nearest neighbors and dynamic modeling. *Source:* Based on Karypis, Han, and Kumar [KHK99].

internal interconnectivity of the two clusters, C_i and C_j. That is,

$$RI(C_i, C_j) = \frac{|EC_{\{C_i,C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)}, \tag{10.12}$$

where $EC_{\{C_i,C_j\}}$ is the edge cut as previously defined for a cluster containing both C_i and C_j. Similarly, EC_{C_i} (or EC_{C_j}) is the minimum sum of the cut edges that partition C_i (or C_j) into two roughly equal parts.

- The **relative closeness**, $RC(C_i, C_j)$, between a pair of clusters, C_i and C_j, is the absolute closeness between C_i and C_j, normalized with respect to the internal closeness of the two clusters, C_i and C_j. It is defined as

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}, \tag{10.13}$$

where $\overline{S}_{EC_{\{C_i,C_j\}}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j, and $\overline{S}_{EC_{C_i}}$ (or $\overline{S}_{EC_{C_j}}$) is the average weight of the edges that belong to the min-cut bisector of cluster C_i (or C_j).

Chameleon has been shown to have greater power at discovering arbitrarily shaped clusters of high quality than several well-known algorithms such as BIRCH and density-based DBSCAN (Section 10.4.1). However, the processing cost for high-dimensional data may require $O(n^2)$ time for n objects in the worst case.

10.3.5 Probabilistic Hierarchical Clustering

Algorithmic 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, algorithmic 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 an algorithmic method, the data objects cannot have any missing attribute values. In the case of data that are partially observed (i.e., some attribute values of some objects are missing), it is not easy to apply an algorithmic hierarchical clustering method because the distance computation cannot be conducted. Third, most of the algorithmic hierarchical clustering methods are heuristic, and at each step locally search 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 10.6 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,

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{10.14}$$

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

$$P(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{10.15}$$

Consequently, the likelihood that X is generated by the model is

$$L(\mathcal{N}(\mu, \sigma^2) : X) = P(X|\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{10.16}$$

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

$$\mathcal{N}(\mu_0, \sigma_0^2) = \arg\max\{L(\mathcal{N}(\mu, \sigma^2) : X)\}, \tag{10.17}$$

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 $C_1, \ldots, C_m$, the quality can be measured by

$$Q(\{C_1, \ldots, C_m\}) = \prod_{i=1}^{m} P(C_i), \tag{10.18}$$

where $P()$ is the maximum likelihood. 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

$$\begin{aligned} &Q((\{C_1, \ldots, C_m\} - \{C_{j_1}, C_{j_2}\}) \cup \{C_{j_1} \cup C_{j_2}\}) - Q(\{C_1, \ldots, C_m\}) \\ &= \frac{\prod_{i=1}^{m} P(C_i) \cdot P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - \prod_{i=1}^{m} P(C_i) \\ &= \prod_{i=1}^{m} P(C_i) \left(\frac{P(C_{j_1} \cup C_{j_2})}{P(C_{j_1})P(C_{j_2})} - 1 \right). \end{aligned} \tag{10.19}$$

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 distance between them can be measured by

$$dist(C_i, C_j) = -\log \frac{P(C_1 \cup C_2)}{P(C_1)P(C_2)}. \tag{10.20}$$

A probabilistic hierarchical clustering method can adopt the agglomerative clustering framework, but use probabilistic models (Eq. 10.20) to measure the distance between clusters.

Upon close observation of Eq. (10.19), 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 Figure 10.11. 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 Figure 10.12.

Probabilistic hierarchical clustering methods are easy to understand, and generally have the same efficiency as algorithmic 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 data set, there may exist multiple hierarchies that

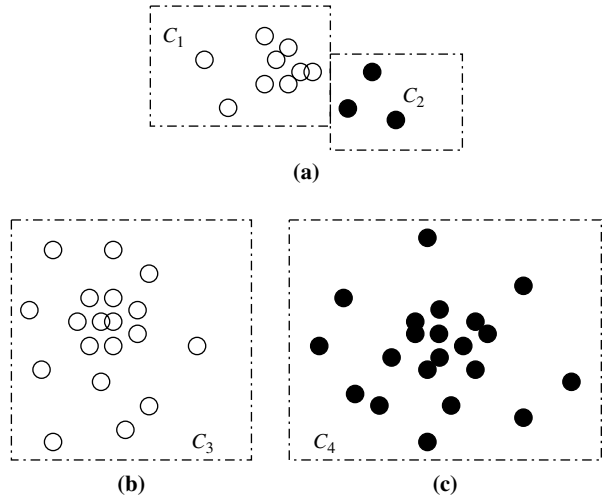


Figure 10.11 Merging clusters in probabilistic hierarchical clustering: (a) Merging clusters C_1 and C_2 leads to an increase in overall cluster quality, but merging clusters (b) C_3 and (c) C_4 does not.

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 10.12 A probabilistic hierarchical clustering algorithm.

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.

10.4 Density-Based Methods

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 Figure 10.13. Given such data, they would likely inaccurately identify convex regions, where noise or outliers are included in the clusters.

To find clusters of arbitrary shape, alternatively, we can model clusters as dense regions in the data space, separated by sparse regions. This is the main strategy behind *density-based clustering methods*, which can discover clusters of nonspherical shape. In this section, you will learn the basic techniques of density-based clustering by studying three representative methods, namely, DBSCAN (Section 10.4.1), OPTICS (Section 10.4.2), and DENCLUE (Section 10.4.3).

10.4.1 DBSCAN: Density-Based Clustering Based on Connected Regions with High Density

"*How can we find dense regions in density-based clustering?*" The *density* of an object $\boldsymbol{o}$ can be measured by the number of objects close to $\boldsymbol{o}$. **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) finds *core objects*, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters.

"*How does* **DBSCAN** *quantify the neighborhood of an object?*" A user-specified parameter $\epsilon > 0$ is used to specify the radius of a neighborhood we consider for every object. The ϵ**-neighborhood** of an object $\boldsymbol{o}$ is the space within a radius ϵ centered at $\boldsymbol{o}$.

Due to the fixed neighborhood size parameterized by ϵ, the **density of a neighborhood** can be measured simply by the number of objects in the neighborhood. To determine whether a neighborhood is dense or not, DBSCAN uses another user-specified

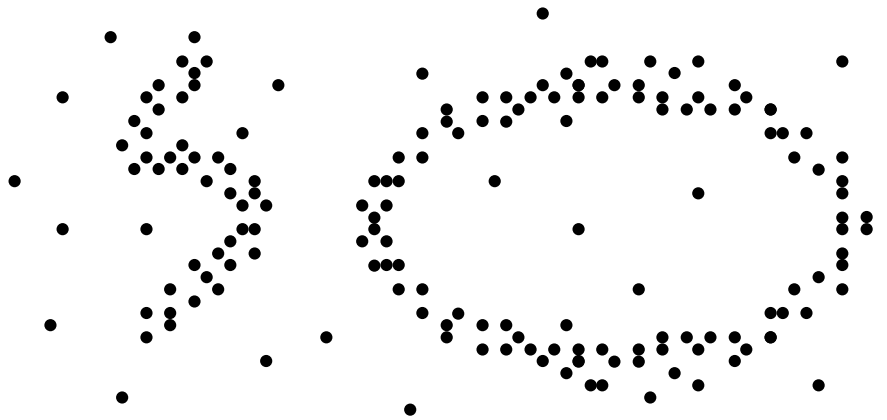


Figure 10.13 Clusters of arbitrary shape.

parameter, *MinPts*, which specifies the density threshold of dense regions. An object is a **core object** if the ϵ-neighborhood of the object contains at least *MinPts* objects. Core objects are the pillars of dense regions.

Given a set, D, of objects, we can identify all core objects with respect to the given parameters, ϵ and *MinPts*. The clustering task is therein reduced to using core objects and their neighborhoods to form dense regions, where the dense regions are clusters. For a core object $\boldsymbol{q}$ and an object $\boldsymbol{p}$, we say that $\boldsymbol{p}$ is **directly density-reachable** from $\boldsymbol{q}$ (with respect to ϵ and *MinPts*) if $\boldsymbol{p}$ is within the ϵ-neighborhood of $\boldsymbol{q}$. Clearly, an object $\boldsymbol{p}$ is directly density-reachable from another object $\boldsymbol{q}$ if and only if $\boldsymbol{q}$ is a core object and $\boldsymbol{p}$ is in the ϵ-neighborhood of $\boldsymbol{q}$. Using the directly density-reachable relation, a core object can "bring" all objects from its ϵ-neighborhood into a dense region.

"*How can we assemble a large dense region using small dense regions centered by core objects?*" In DBSCAN, $\boldsymbol{p}$ is **density-reachable** from $\boldsymbol{q}$ (with respect to ϵ and *MinPts* in D) if there is a chain of objects $\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}$, such that $\boldsymbol{p_1} = \boldsymbol{q}$, $\boldsymbol{p_n} = \boldsymbol{p}$, and $\boldsymbol{p_{i+1}}$ is directly density-reachable from $\boldsymbol{p_i}$ with respect to ϵ and *MinPts*, for $1 \leq i \leq n$, $\boldsymbol{p_i} \in D$. Note that density-reachability is not an equivalence relation because it is not symmetric. If both $\boldsymbol{o_1}$ and $\boldsymbol{o_2}$ are core objects and $\boldsymbol{o_1}$ is density-reachable from $\boldsymbol{o_2}$, then $\boldsymbol{o_2}$ is density-reachable from $\boldsymbol{o_1}$. However, if $\boldsymbol{o_2}$ is a core object but $\boldsymbol{o_1}$ is not, then $\boldsymbol{o_1}$ may be density-reachable from $\boldsymbol{o_2}$, but not vice versa.

To connect core objects as well as their neighbors in a dense region, **DBSCAN** uses the notion of density-connectedness. Two objects $\boldsymbol{p_1}, \boldsymbol{p_2} \in D$ are **density-connected** with respect to ϵ and *MinPts* if there is an object $\boldsymbol{q} \in D$ such that both $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are density-reachable from $\boldsymbol{q}$ with respect to ϵ and *MinPts*. Unlike density-reachability, density-connectedness is an equivalence relation. It is easy to show that, for objects $\boldsymbol{o_1}$, $\boldsymbol{o_2}$, and $\boldsymbol{o_3}$, if $\boldsymbol{o_1}$ and $\boldsymbol{o_2}$ are density-connected, and $\boldsymbol{o_2}$ and $\boldsymbol{o_3}$ are density-connected, then so are $\boldsymbol{o_1}$ and $\boldsymbol{o_3}$.

Example 10.7 **Density-reachability and density-connectivity.** Consider Figure 10.14 for a given ϵ represented by the radius of the circles, and, say, let $MinPts = 3$.

Of the labeled points, $\boldsymbol{m}, \boldsymbol{p}, \boldsymbol{o}, \boldsymbol{r}$ are core objects because each is in an ϵ-neighborhood containing at least three points. Object $\boldsymbol{q}$ is directly density-reachable from $\boldsymbol{m}$. Object $\boldsymbol{m}$ is directly density-reachable from $\boldsymbol{p}$ and vice versa.

Object $\boldsymbol{q}$ is (indirectly) density-reachable from $\boldsymbol{p}$ because $\boldsymbol{q}$ is directly density-reachable from $\boldsymbol{m}$ and $\boldsymbol{m}$ is directly density-reachable from $\boldsymbol{p}$. However, $\boldsymbol{p}$ is not density-reachable from $\boldsymbol{q}$ because $\boldsymbol{q}$ is not a core object. Similarly, $\boldsymbol{r}$ and $\boldsymbol{s}$ are density-reachable from $\boldsymbol{o}$ and $\boldsymbol{o}$ is density-reachable from $\boldsymbol{r}$. Thus, $\boldsymbol{o}$, $\boldsymbol{r}$, and $\boldsymbol{s}$ are all density-connected. ■

We can use the closure of density-connectedness to find connected dense regions as clusters. Each closed set is a **density-based cluster**. A subset $C \subseteq D$ is a cluster if (1) for any two objects $\boldsymbol{o_1}, \boldsymbol{o_2} \in C$, $\boldsymbol{o_1}$ and $\boldsymbol{o_2}$ are density-connected; and (2) there does not exist an object $\boldsymbol{o} \in C$ and another object $\boldsymbol{o'} \in (D - C)$ such that $\boldsymbol{o}$ and $\boldsymbol{o'}$ are density-connected.

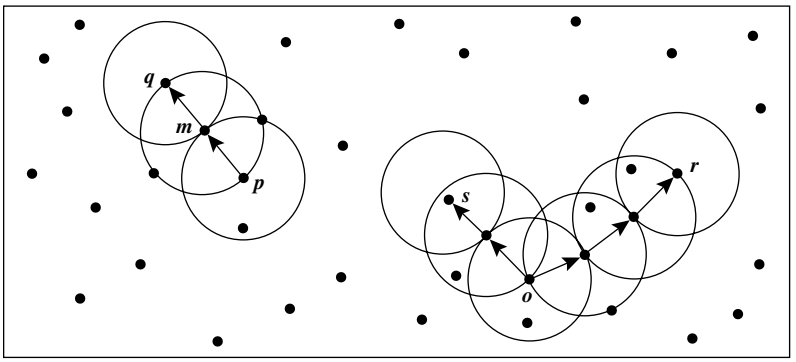


Figure 10.14 Density-reachability and density-connectivity in density-based clustering. *Source:* Based on Ester, Kriegel, Sander, and Xu [EKSX96].

"How does DBSCAN find clusters?" Initially, all objects in a given data set D are marked as "unvisited." DBSCAN randomly selects an unvisited object $\boldsymbol{p}$, marks $\boldsymbol{p}$ as "visited," and checks whether the ϵ-neighborhood of $\boldsymbol{p}$ contains at least *MinPts* objects. If not, $\boldsymbol{p}$ is marked as a noise point. Otherwise, a new cluster C is created for $\boldsymbol{p}$, and all the objects in the ϵ-neighborhood of $\boldsymbol{p}$ are added to a candidate set, N. DBSCAN iteratively adds to C those objects in N that do not belong to any cluster. In this process, for an object $\boldsymbol{p'}$ in N that carries the label "unvisited," DBSCAN marks it as "visited" and checks its ϵ-neighborhood. If the ϵ-neighborhood of $\boldsymbol{p'}$ has at least *MinPts* objects, those objects in the ϵ-neighborhood of $\boldsymbol{p'}$ are added to N. DBSCAN continues adding objects to C until C can no longer be expanded, that is, N is empty. At this time, cluster C is completed, and thus is output.

To find the next cluster, DBSCAN randomly selects an unvisited object from the remaining ones. The clustering process continues until all objects are visited. The pseudocode of the DBSCAN algorithm is given in Figure 10.15.

If a spatial index is used, the computational complexity of DBSCAN is $O(n \log n)$, where n is the number of database objects. Otherwise, the complexity is $O(n^2)$. With appropriate settings of the user-defined parameters, ϵ and *MinPts*, the algorithm is effective in finding arbitrary-shaped clusters.

10.4.2 OPTICS: Ordering Points to Identify the Clustering Structure

Although DBSCAN can cluster objects given input parameters such as ϵ (the maximum radius of a neighborhood) and *MinPts* (the minimum number of points required in the neighborhood of a core object), it encumbers users with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. This is a problem associated with many other clustering algorithms. Such parameter settings

Algorithm: DBSCAN: a density-based clustering algorithm.

Input:

- D: a data set containing n objects,
- ϵ: the radius parameter, and
- *MinPts*: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```
(1)  mark all objects as unvisited;
(2)  do
(3)      randomly select an unvisited object p;
(4)      mark p as visited;
(5)      if the ε-neighborhood of p has at least MinPts objects
(6)          create a new cluster C, and add p to C;
(7)          let N be the set of objects in the ε-neighborhood of p;
(8)          for each point p' in N
(9)              if p' is unvisited
(10)                 mark p' as visited;
(11)                 if the ε-neighborhood of p' has at least MinPts points,
                     add those points to N;
(12)             if p' is not yet a member of any cluster, add p' to C;
(13)         end for
(14)         output C;
(15)     else mark p as noise;
(16) until no object is unvisited;
```

Figure 10.15 DBSCAN algorithm.

are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms are sensitive to these parameter values: Slightly different settings may lead to very different clusterings of the data. Moreover, real-world, high-dimensional data sets often have very skewed distributions such that their intrinsic clustering structure may not be well characterized by a single set of *global* density parameters.

Note that density-based clusters are monotonic with respect to the neighborhood threshold. That is, in DBSCAN, for a fixed *MinPts* value and two neighborhood thresholds, $\epsilon_1 < \epsilon_2$, a cluster C with respect to ϵ_1 and *MinPts* must be a subset of a cluster C' with respect to ϵ_2 and *MinPts*. This means that if two objects are in a density-based cluster, they must also be in a cluster with a lower density requirement.

To overcome the difficulty in using one set of global parameters in clustering analysis, a cluster analysis method called **OPTICS** was proposed. OPTICS does not explicitly produce a data set clustering. Instead, it outputs a **cluster ordering**. This is a linear list

of all objects under analysis and represents the *density-based clustering structure* of the data. Objects in a denser cluster are listed closer to each other in the cluster ordering. This ordering is equivalent to density-based clustering obtained from a wide range of parameter settings. Thus, OPTICS does not require the user to provide a specific density threshold. The cluster ordering can be used to extract basic clustering information (e.g., cluster centers, or arbitrary-shaped clusters), derive the intrinsic clustering structure, as well as provide a visualization of the clustering.

To construct the different clusterings simultaneously, the objects are processed in a specific order. This order selects an object that is density-reachable with respect to the lowest ϵ value so that clusters with higher density (lower ϵ) will be finished first. Based on this idea, OPTICS needs two important pieces of information per object:

- The **core-distance** of an object $\boldsymbol{p}$ is the smallest value ϵ' such that the ϵ'-neighborhood of $\boldsymbol{p}$ has at least *MinPts* objects. That is, ϵ' is the minimum distance threshold that makes $\boldsymbol{p}$ a core object. If $\boldsymbol{p}$ is not a core object with respect to ϵ and *MinPts*, the core-distance of $\boldsymbol{p}$ is undefined.
- The **reachability-distance** to object $\boldsymbol{p}$ from $\boldsymbol{q}$ is the minimum radius value that makes $\boldsymbol{p}$ density-reachable from $\boldsymbol{q}$. According to the definition of density-reachability, $\boldsymbol{q}$ has to be a core object and $\boldsymbol{p}$ must be in the neighborhood of $\boldsymbol{q}$. Therefore, the reachability-distance from $\boldsymbol{q}$ to $\boldsymbol{p}$ is $\max\{\textit{core-distance}(\boldsymbol{q}), \textit{dist}(\boldsymbol{p}, \boldsymbol{q})\}$. If $\boldsymbol{q}$ is not a core object with respect to ϵ and *MinPts*, the reachability-distance to $\boldsymbol{p}$ from $\boldsymbol{q}$ is undefined.

  An object $\boldsymbol{p}$ may be directly reachable from multiple core objects. Therefore, $\boldsymbol{p}$ may have multiple reachability-distances with respect to different core objects. The smallest reachability-distance of $\boldsymbol{p}$ is of particular interest because it gives the shortest path for which $\boldsymbol{p}$ is connected to a dense cluster.

Example 10.8 **Core-distance and reachability-distance.** Figure 10.16 illustrates the concepts of core-distance and reachability-distance. Suppose that $\epsilon = 6$ mm and $\textit{MinPts} = 5$. The core-distance of $\boldsymbol{p}$ is the distance, ϵ', between $\boldsymbol{p}$ and the fourth closest data object from $\boldsymbol{p}$. The reachability-distance of $\boldsymbol{q_1}$ from $\boldsymbol{p}$ is the core-distance of $\boldsymbol{p}$ (i.e., $\epsilon' = 3$ mm) because this is greater than the Euclidean distance from $\boldsymbol{p}$ to $\boldsymbol{q_1}$. The reachability-distance of $\boldsymbol{q_2}$ with respect to $\boldsymbol{p}$ is the Euclidean distance from $\boldsymbol{p}$ to $\boldsymbol{q_2}$ because this is greater than the core-distance of $\boldsymbol{p}$. ■

OPTICS computes an ordering of all objects in a given database and, for each object in the database, stores the core-distance and a suitable reachability-distance. OPTICS maintains a list called OrderSeeds to generate the output ordering. Objects in OrderSeeds are sorted by the reachability-distance from their respective closest core objects, that is, by the smallest reachability-distance of each object.

OPTICS begins with an arbitrary object from the input database as the current object, $\boldsymbol{p}$. It retrieves the ϵ-neighborhood of $\boldsymbol{p}$, determines the core-distance, and sets the reachability-distance to *undefined*. The current object, $\boldsymbol{p}$, is then written to output.

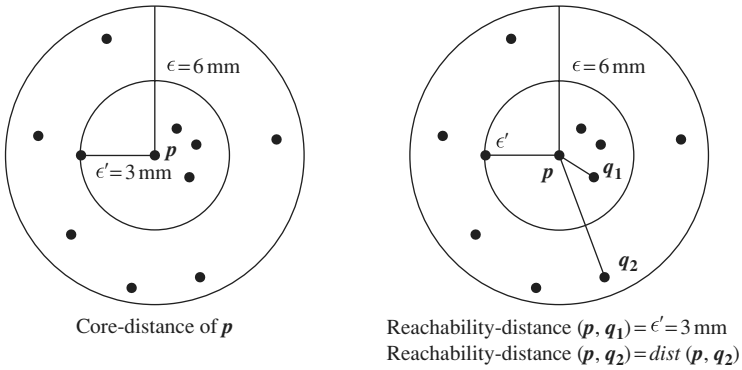


Figure 10.16 OPTICS terminology. *Source:* Based on Ankerst, Breunig, Kriegel, and Sander [ABKS99].

If $\boldsymbol{p}$ is not a core object, OPTICS simply moves on to the next object in the OrderSeeds list (or the input database if OrderSeeds is empty). If $\boldsymbol{p}$ is a core object, then for each object, $\boldsymbol{q}$, in the ϵ-neighborhood of $\boldsymbol{p}$, OPTICS updates its reachability-distance from $\boldsymbol{p}$ and inserts $\boldsymbol{q}$ into OrderSeeds if $\boldsymbol{q}$ has not yet been processed. The iteration continues until the input is fully consumed and OrderSeeds is empty.

A data set's cluster ordering can be represented graphically, which helps to visualize and understand the clustering structure in a data set. For example, Figure 10.17 is the reachability plot for a simple 2-D data set, which presents a general overview of how the data are structured and clustered. The data objects are plotted in the clustering order (horizontal axis) together with their respective reachability-distances (vertical axis). The three Gaussian "bumps" in the plot reflect three clusters in the data set. Methods have also been developed for viewing clustering structures of high-dimensional data at various levels of detail.

The structure of the OPTICS algorithm is very similar to that of DBSCAN. Consequently, the two algorithms have the same time complexity. The complexity is $O(n \log n)$ if a spatial index is used, and $O(n^2)$ otherwise, where n is the number of objects.

10.4.3 DENCLUE: Clustering Based on Density Distribution Functions

Density estimation is a core issue in density-based clustering methods. **DENCLUE** (DENsity-based CLUstEring) is a clustering method based on a set of density distribution functions. We first give some background on density estimation, and then describe the DENCLUE algorithm.

In probability and statistics, **density estimation** is the estimation of an unobservable underlying probability density function based on a set of observed data. In the context of density-based clustering, the unobservable underlying probability density function is the true distribution of the population of all possible objects to be analyzed. The observed data set is regarded as a random sample from that population.

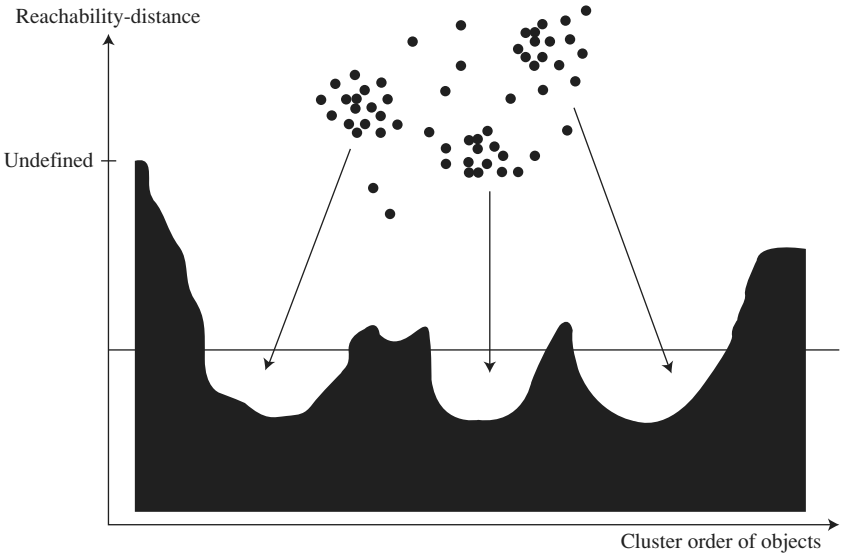


Figure 10.17 Cluster ordering in OPTICS. *Source:* Adapted from Ankerst, Breunig, Kriegel, and Sander [ABKS99].

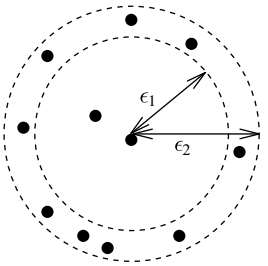


Figure 10.18 The subtlety in density estimation in DBSCAN and OPTICS: Increasing the neighborhood radius slightly from ϵ_1 to ϵ_2 results in a much higher density.

In DBSCAN and OPTICS, density is calculated by counting the number of objects in a neighborhood defined by a radius parameter, ϵ. Such density estimates can be highly sensitive to the radius value used. For example, in Figure 10.18, the density changes significantly as the radius increases by a small amount.

To overcome this problem, **kernel density estimation** can be used, which is a nonparametric density estimation approach from statistics. The general idea behind kernel density estimation is simple. We treat an observed object as an indicator of

high-probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects.

Formally, let $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ be an independent and identically distributed sample of a random variable f. The *kernel density approximation of the probability density function* is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right), \tag{10.21}$$

where $K()$ is a kernel and h is the bandwidth serving as a smoothing parameter. A **kernel** can be regarded as a function modeling the influence of a sample point within its neighborhood. Technically, a kernel $K()$ is a non-negative real-valued integrable function that should satisfy two requirements: $\int_{-\infty}^{+\infty} K(u)du = 1$ and $K(-u) = K(u)$ for all values of u. A frequently used kernel is a standard Gaussian function with a mean of 0 and a variance of 1:

$$K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(\boldsymbol{x} - \boldsymbol{x_i})^2}{2h^2}}. \tag{10.22}$$

DENCLUE uses a Gaussian kernel to estimate density based on the given set of objects to be clustered. A point $\boldsymbol{x^*}$ is called a **density attractor** if it is a local maximum of the estimated density function. To avoid trivial local maximum points, DENCLUE uses a noise threshold, ξ, and only considers those density attractors $\boldsymbol{x^*}$ such that $\hat{f}(\boldsymbol{x^*}) \geq \xi$. These nontrivial density attractors are the centers of clusters.

Objects under analysis are assigned to clusters through density attractors using a stepwise hill-climbing procedure. For an object, $\boldsymbol{x}$, the hill-climbing procedure starts from $\boldsymbol{x}$ and is guided by the gradient of the estimated density function. That is, the density attractor for $\boldsymbol{x}$ is computed as

$$\boldsymbol{x^0} = \boldsymbol{x}$$
$$\boldsymbol{x^{j+1}} = \boldsymbol{x^j} + \delta \frac{\nabla \hat{f}(\boldsymbol{x^j})}{|\nabla \hat{f}(\boldsymbol{x^j})|}, \tag{10.23}$$

where δ is a parameter to control the speed of convergence, and

$$\nabla \hat{f}(\boldsymbol{x}) = \frac{1}{h^{d+2} n \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right)(\boldsymbol{x_i} - \boldsymbol{x})}. \tag{10.24}$$

The hill-climbing procedure stops at step $k > 0$ if $\hat{f}(\boldsymbol{x^{k+1}}) < \hat{f}(\boldsymbol{x^k})$, and assigns $\boldsymbol{x}$ to the density attractor $\boldsymbol{x^*} = \boldsymbol{x^k}$. An object $\boldsymbol{x}$ is an outlier or noise if it converges in the hill-climbing procedure to a local maximum $\boldsymbol{x^*}$ with $\hat{f}(\boldsymbol{x^*}) < \xi$.

A cluster in DENCLUE is a set of density attractors X and a set of input objects C such that each object in C is assigned to a density attractor in X, and there exists a path between every pair of density attractors where the density is above ξ. By using multiple density attractors connected by paths, DENCLUE can find clusters of arbitrary shape.

DENCLUE has several advantages. It can be regarded as a generalization of several well-known clustering methods such as single-linkage approaches and DBSCAN. Moreover, DENCLUE is invariant against noise. The kernel density estimation can effectively reduce the influence of noise by uniformly distributing noise into the input data.

10.5 Grid-Based Methods

The clustering methods discussed so far are data-driven—they partition the set of objects and adapt to the distribution of the objects in the embedding space. Alternatively, a **grid-based clustering** method takes a space-driven approach by partitioning the embedding space into *cells* independent of the distribution of the input objects.

The *grid-based clustering* approach uses a multiresolution grid data structure. It quantizes the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed. The main advantage of the approach is its fast processing time, which is typically independent of the number of data objects, yet dependent on only the number of cells in each dimension in the quantized space.

In this section, we illustrate grid-based clustering using two typical examples. STING (Section 10.5.1) explores statistical information stored in the grid cells. CLIQUE (Section 10.5.2) represents a grid- and density-based approach for subspace clustering in a high-dimensional data space.

10.5.1 STING: STatistical INformation Grid

STING is a grid-based multiresolution clustering technique in which the embedding spatial area of the input objects is divided into rectangular cells. The space can be divided in a hierarchical and recursive way. Several levels of such rectangular cells correspond to different levels of resolution and form a hierarchical structure: Each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell, such as the mean, maximum, and minimum values, is precomputed and stored as *statistical parameters*. These statistical parameters are useful for query processing and for other data analysis tasks.

Figure 10.19 shows a hierarchical structure for STING clustering. The statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; and the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum), and the type of *distribution* that the attribute value in the cell follows such as *normal*, *uniform*, *exponential*, or *none* (if the distribution is unknown). Here, the attribute is a selected measure for analysis such as *price* for house objects. When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known

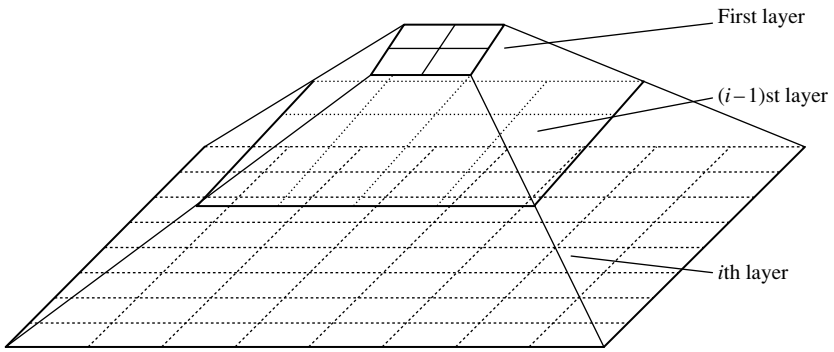


Figure 10.19 Hierarchical structure for STING clustering.

beforehand or obtained by hypothesis tests such as the χ^2 test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If the distributions of the lower-level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

"*How is this statistical information useful for query answering?*" The statistical parameters can be used in a top-down, grid-based manner as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated probability range) reflecting the cell's relevancy to the given query. The irrelevant cells are removed from further consideration. Processing of the next lower level examines only the remaining relevant cells. This process is repeated until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data that fall into the relevant cells are retrieved and further processed until they meet the query's requirements.

An interesting property of STING is that it approaches the clustering result of DBSCAN if the granularity approaches 0 (i.e., toward very low-level data). In other words, using the count and cell size information, dense clusters can be identified approximately using STING. Therefore, STING can also be regarded as a density-based clustering method.

"*What advantages does STING offer over other clustering methods?*" STING offers several advantages: (1) the grid-based computation is *query-independent* because the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query; (2) the grid structure facilitates parallel processing and incremental updating; and (3) the method's efficiency is a major advantage: STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters is $O(n)$, where n is the total number of objects. After generating the hierarchical structure, the query processing time

is $O(g)$, where g is the total number of grid cells at the lowest level, which is usually much smaller than n.

Because STING uses a multiresolution approach to cluster analysis, the quality of STING clustering depends on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of processing will increase substantially; however, if the bottom level of the grid structure is too coarse, it may reduce the quality of cluster analysis. Moreover, STING does not consider the spatial relationship between the children and their neighboring cells for construction of a parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the fast processing time of the technique.

10.5.2 CLIQUE: An Apriori-like Subspace Clustering Method

A data object often has tens of attributes, many of which may be irrelevant. The values of attributes may vary considerably. These factors can make it difficult to locate clusters that span the entire data space. It may be more meaningful to instead search for clusters within different *subspaces* of the data. For example, consider a health-informatics application where patient records contain extensive attributes describing personal information, numerous symptoms, conditions, and family history.

Finding a nontrivial group of patients for which all or even most of the attributes strongly agree is unlikely. In bird flu patients, for instance, the *age*, *gender*, and *job* attributes may vary dramatically within a wide range of values. Thus, it can be difficult to find such a cluster within the entire data space. Instead, by searching in subspaces, we may find a cluster of similar patients in a lower-dimensional space (e.g., patients who are similar to one other with respect to symptoms like high fever, cough but no runny nose, and aged between 3 and 16).

CLIQUE (CLustering In QUEst) is a simple grid-based method for finding density-based clusters in subspaces. CLIQUE partitions each dimension into nonoverlapping intervals, thereby partitioning the entire embedding space of the data objects into cells. It uses a density threshold to identify *dense* cells and *sparse* ones. A cell is dense if the number of objects mapped to it exceeds the density threshold.

The main strategy behind CLIQUE for identifying a candidate search space uses the monotonicity of dense cells with respect to dimensionality. This is based on the *Apriori property* used in frequent pattern and association rule mining (Chapter 6). In the context of clusters in subspaces, the monotonicity says the following. A k-dimensional cell c ($k > 1$) can have at least l points only if every $(k-1)$-dimensional projection of c, which is a cell in a $(k-1)$-dimensional subspace, has at least l points. Consider Figure 10.20, where the embedding data space contains three dimensions: *age, salary*, and *vacation*. A 2-D cell, say in the subspace formed by *age* and *salary*, contains l points only if the projection of this cell in every dimension, that is, *age* and *salary*, respectively, contains at least l points.

CLIQUE performs clustering in two steps. In the first step, CLIQUE partitions the d-dimensional data space into nonoverlapping rectangular units, identifying the dense units among these. CLIQUE finds dense cells in all of the subspaces. To do so,

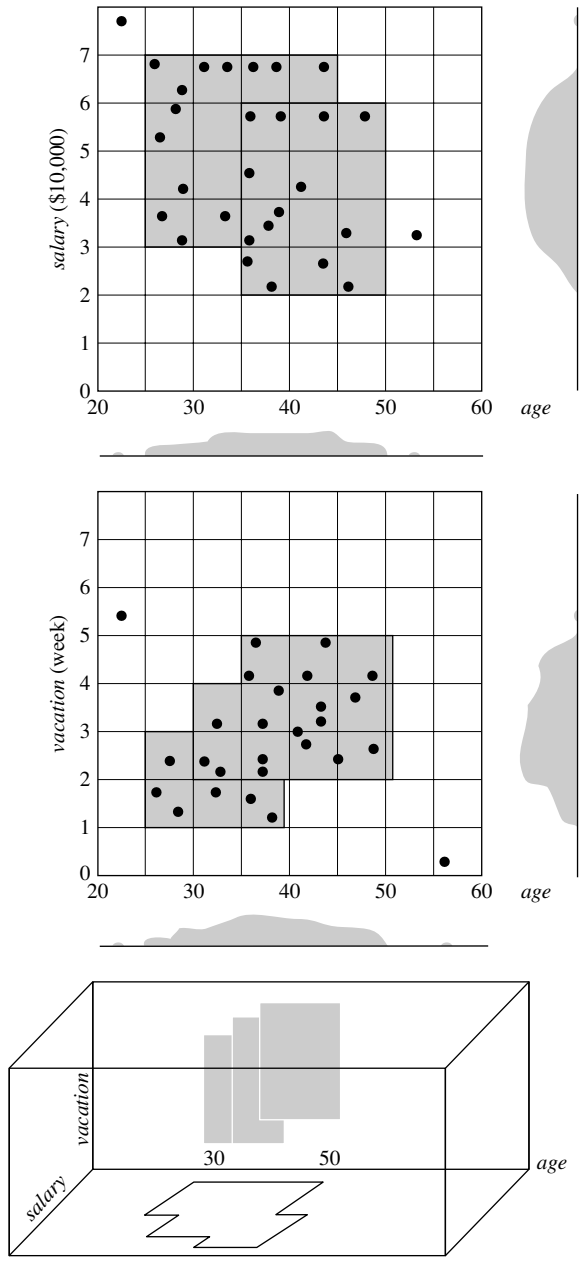


Figure 10.20 Dense units found with respect to *age* for the dimensions *salary* and *vacation* are intersected to provide a candidate search space for dense units of higher dimensionality.

CLIQUE partitions every dimension into intervals, and identifies intervals containing at least l points, where l is the density threshold. CLIQUE then iteratively joins two k-dimensional dense cells, c_1 and c_2, in subspaces $(D_{i_1}, \ldots, D_{i_k})$ and $(D_{j_1}, \ldots, D_{j_k})$, respectively, if $D_{i_1} = D_{j_1}, \ldots, D_{i_{k-1}} = D_{j_{k-1}}$, and c_1 and c_2 share the same intervals in those dimensions. The join operation generates a new $(k+1)$-dimensional candidate cell c in space $(D_{i_1}, \ldots, D_{i_{k-1}}, D_{i_k}, D_{j_k})$. CLIQUE checks whether the number of points in c passes the density threshold. The iteration terminates when no candidates can be generated or no candidate cells are dense.

In the second step, CLIQUE uses the dense cells in each subspace to assemble clusters, which can be of arbitrary shape. The idea is to apply the Minimum Description Length (MDL) principle (Chapter 8) to use the *maximal regions* to cover connected dense cells, where a maximal region is a hyperrectangle where every cell falling into this region is dense, and the region cannot be extended further in any dimension in the subspace. Finding the best description of a cluster in general is NP-Hard. Thus, CLIQUE adopts a simple greedy approach. It starts with an arbitrary dense cell, finds a maximal region covering the cell, and then works on the remaining dense cells that have not yet been covered. The greedy method terminates when all dense cells are covered.

"*How effective is CLIQUE?*" CLIQUE automatically finds subspaces of the highest dimensionality such that high-density clusters exist in those subspaces. It is insensitive to the order of input objects and does not presume any canonical data distribution. It scales linearly with the size of the input and has good scalability as the number of dimensions in the data is increased. However, obtaining a meaningful clustering is dependent on proper tuning of the grid size (which is a stable structure here) and the density threshold. This can be difficult in practice because the grid size and density threshold are used across all combinations of dimensions in the data set. Thus, the accuracy of the clustering results may be degraded at the expense of the method's simplicity. Moreover, for a given dense region, all projections of the region onto lower-dimensionality subspaces will also be dense. This can result in a large overlap among the reported dense regions. Furthermore, it is difficult to find clusters of rather different densities within different dimensional subspaces.

Several extensions to this approach follow a similar philosophy. For example, we can think of a grid as a set of fixed bins. Instead of using fixed bins for each of the dimensions, we can use an adaptive, data-driven strategy to dynamically determine the bins for each dimension based on data distribution statistics. Alternatively, instead of using a density threshold, we may use entropy (Chapter 8) as a measure of the quality of subspace clusters.

10.6 Evaluation of Clustering

By now you have learned what clustering is and know several popular clustering methods. You may ask, "*When I try out a clustering method on a data set, how can I evaluate whether the clustering results are good?*" In general, *cluster evaluation* assesses

the feasibility of clustering analysis on a data set and the quality of the results generated by a clustering method. The major tasks of clustering evaluation include the following:

- *Assessing clustering tendency*. In this task, for a given data set, we assess whether a nonrandom structure exists in the data. Blindly applying a clustering method on a data set will return clusters; however, the clusters mined may be misleading. Clustering analysis on a data set is meaningful only when there is a nonrandom structure in the data.
- *Determining the number of clusters in a data set*. A few algorithms, such as k-means, require the number of clusters in a data set as the parameter. Moreover, the number of clusters can be regarded as an interesting and important summary statistic of a data set. Therefore, it is desirable to estimate this number even before a clustering algorithm is used to derive detailed clusters.
- *Measuring clustering quality*. After applying a clustering method on a data set, we want to assess how good the resulting clusters are. A number of measures can be used. Some methods measure how well the clusters fit the data set, while others measure how well the clusters match the ground truth, if such truth is available. There are also measures that score clusterings and thus can compare two sets of clustering results on the same data set.

In the rest of this section, we discuss each of these three topics.

10.6.1 Assessing Clustering Tendency

Clustering tendency assessment determines whether a given data set has a non-random structure, which may lead to meaningful clusters. Consider a data set that does not have any non-random structure, such as a set of uniformly distributed points in a data space. Even though a clustering algorithm may return clusters for the data, those clusters are random and are not meaningful.

Example 10.9 Clustering requires nonuniform distribution of data. Figure 10.21 shows a data set that is uniformly distributed in 2-D data space. Although a clustering algorithm may still artificially partition the points into groups, the groups will unlikely mean anything significant to the application due to the uniform distribution of the data. ■

"*How can we assess the clustering tendency of a data set?*" Intuitively, we can try to measure the probability that the data set is generated by a uniform data distribution. This can be achieved using statistical tests for spatial randomness. To illustrate this idea, let's look at a simple yet effective statistic called the Hopkins Statistic.

The **Hopkins Statistic** is a spatial statistic that tests the spatial randomness of a variable as distributed in a space. Given a data set, D, which is regarded as a sample of

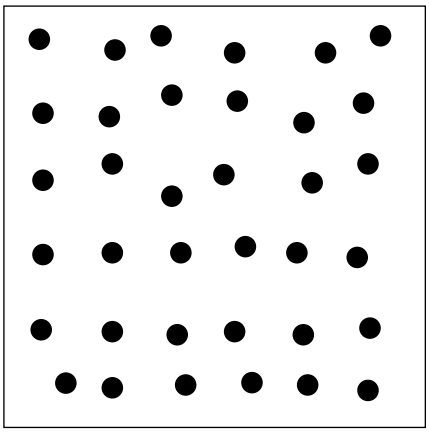


Figure 10.21 A data set that is uniformly distributed in the data space.

a random variable, o, we want to determine how far away o is from being uniformly distributed in the data space. We calculate the Hopkins Statistic as follows:

1. Sample n points, $\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}$, uniformly from D. That is, each point in D has the same probability of being included in this sample. For each point, $\boldsymbol{p_i}$, we find the nearest neighbor of $\boldsymbol{p_i}$ ($1 \leq i \leq n$) in D, and let x_i be the distance between $\boldsymbol{p_i}$ and its nearest neighbor in D. That is,

$$x_i = \min_{\boldsymbol{v} \in D} \{dist(\boldsymbol{p_i}, \boldsymbol{v})\}. \tag{10.25}$$

2. Sample n points, $\boldsymbol{q_1}, \ldots, \boldsymbol{q_n}$, uniformly from D. For each $\boldsymbol{q_i}$ ($1 \leq i \leq n$), we find the nearest neighbor of $\boldsymbol{q_i}$ in $D - \{\boldsymbol{q_i}\}$, and let y_i be the distance between $\boldsymbol{q_i}$ and its nearest neighbor in $D - \{\boldsymbol{q_i}\}$. That is,

$$y_i = \min_{\boldsymbol{v} \in D, \boldsymbol{v} \neq \boldsymbol{q_i}} \{dist(\boldsymbol{q_i}, \boldsymbol{v})\}. \tag{10.26}$$

3. Calculate the Hopkins Statistic, H, as

$$H = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i}. \tag{10.27}$$

"*What does the Hopkins Statistic tell us about how likely data set* D *follows a uniform distribution in the data space?*" If D were uniformly distributed, then $\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n} x_i$ would be close to each other, and thus H would be about 0.5. However, if D were highly skewed, then $\sum_{i=1}^{n} y_i$ would be substantially smaller than $\sum_{i=1}^{n} x_i$ in expectation, and thus H would be close to 0.

Our null hypothesis is the *homogeneous hypothesis*—that D is uniformly distributed and thus contains no meaningful clusters. The *nonhomogeneous hypothesis* (i.e., that D is not uniformly distributed and thus contains clusters) is the alternative hypothesis. We can conduct the Hopkins Statistic test iteratively, using 0.5 as the threshold to reject the alternative hypothesis. That is, if $H > 0.5$, then it is unlikely that D has statistically significant clusters.

10.6.2 Determining the Number of Clusters

Determining the "right" number of clusters in a data set is important, not only because some clustering algorithms like k-means require such a parameter, but also because the appropriate number of clusters controls the proper granularity of cluster analysis. It can be regarded as finding a good balance between *compressibility* and *accuracy* in cluster analysis. Consider two extreme cases. What if you were to treat the entire data set as a cluster? This would maximize the compression of the data, but such a cluster analysis has no value. On the other hand, treating each object in a data set as a cluster gives the finest clustering resolution (i.e., most accurate due to the zero distance between an object and the corresponding cluster center). In some methods like k-means, this even achieves the best cost. However, having one object per cluster does not enable any data summarization.

Determining the number of clusters is far from easy, often because the "right" number is ambiguous. Figuring out what the right number of clusters should be often depends on the distribution's shape and scale in the data set, as well as the clustering resolution required by the user. There are many possible ways to estimate the number of clusters. Here, we briefly introduce a few simple yet popular and effective methods.

A simple method is to set the number of clusters to about $\sqrt{\frac{n}{2}}$ for a data set of n points. In expectation, each cluster has $\sqrt{2n}$ points.

The **elbow method** is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster. This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other. However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction. Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Technically, given a number, $k > 0$, we can form k clusters on the data set in question using a clustering algorithm like k-means, and calculate the sum of within-cluster variances, $var(k)$. We can then plot the curve of var with respect to k. The first (or most significant) turning point of the curve suggests the "right" number.

More advanced methods can determine the number of clusters using information criteria or information theoretic approaches. Please refer to the bibliographic notes for further information (Section 10.9).

The "right" number of clusters in a data set can also be determined by **cross-validation**, a technique often used in classification (Chapter 8). First, divide the given data set, D, into m parts. Next, use $m-1$ parts to build a clustering model, and use the remaining part to test the quality of the clustering. For example, for each point in the test set, we can find the closest centroid. Consequently, we can use the sum of the squared distances between all points in the test set and the closest centroids to measure how well the clustering model fits the test set. For any integer $k > 0$, we repeat this process m times to derive clusterings of k clusters by using each part in turn as the test set. The average of the quality measure is taken as the overall quality measure. We can then compare the overall quality measure with respect to different values of k, and find the number of clusters that best fits the data.

10.6.3 Measuring Clustering Quality

Suppose you have assessed the clustering tendency of a given data set. You may have also tried to predetermine the number of clusters in the set. You can now apply one or multiple clustering methods to obtain clusterings of the data set. "*How good is the clustering generated by a method, and how can we compare the clusterings generated by different methods?*"

We have a few methods to choose from for measuring the quality of a clustering. In general, these methods can be categorized into two groups according to whether ground truth is available. Here, *ground truth* is the ideal clustering that is often built using human experts.

If ground truth is available, it can be used by **extrinsic methods**, which compare the clustering against the group truth and measure. If the ground truth is unavailable, we can use **intrinsic methods**, which evaluate the goodness of a clustering by considering how well the clusters are separated. Ground truth can be considered as supervision in the form of "cluster labels." Hence, extrinsic methods are also known as *supervised methods*, while intrinsic methods are *unsupervised methods*.

Let's have a look at simple methods from each category.

Extrinsic Methods

When the ground truth is available, we can compare it with a clustering to assess the clustering. Thus, the core task in extrinsic methods is to assign a score, $Q(\mathcal{C}, \mathcal{C}_g)$, to a clustering, $\mathcal{C}$, given the ground truth, $\mathcal{C}_g$. Whether an extrinsic method is effective largely depends on the measure, Q, it uses.

In general, a measure Q on clustering quality is effective if it satisfies the following four essential criteria:

- **Cluster homogeneity.** This requires that the more pure the clusters in a clustering are, the better the clustering. Suppose that ground truth says that the objects in a data set, D, can belong to categories $L_1, \ldots, L_n$. Consider clustering, $\mathcal{C}_1$, wherein a cluster $C \in \mathcal{C}_1$ contains objects from two categories L_i, L_j $(1 \leq i < j \leq n)$. Also

consider clustering $\mathcal{C}_2$, which is identical to $\mathcal{C}_1$ except that C_2 is split into two clusters containing the objects in L_i and L_j, respectively. A clustering quality measure, Q, respecting cluster homogeneity should give a higher score to $\mathcal{C}_2$ than $\mathcal{C}_1$, that is, $Q(\mathcal{C}_2,\mathcal{C}_g) > Q(\mathcal{C}_1,\mathcal{C}_g)$.

- **Cluster completeness.** This is the counterpart of cluster homogeneity. Cluster completeness requires that for a clustering, if any two objects belong to the same category according to ground truth, then they should be assigned to the same cluster. Cluster completeness requires that a clustering should assign objects belonging to the same category (according to ground truth) to the same cluster. Consider clustering $\mathcal{C}_1$, which contains clusters C_1 and C_2, of which the members belong to the same category according to ground truth. Let clustering $\mathcal{C}_2$ be identical to $\mathcal{C}_1$ except that C_1 and C_2 are merged into one cluster in $\mathcal{C}_2$. Then, a clustering quality measure, Q, respecting cluster completeness should give a higher score to $\mathcal{C}_2$, that is, $Q(\mathcal{C}_2,\mathcal{C}_g) > Q(\mathcal{C}_1,\mathcal{C}_g)$.
- **Rag bag.** In many practical scenarios, there is often a "rag bag" category containing objects that cannot be merged with other objects. Such a category is often called "miscellaneous," "other," and so on. The rag bag criterion states that putting a heterogeneous object into a pure cluster should be penalized more than putting it into a rag bag. Consider a clustering $\mathcal{C}_1$ and a cluster $C \in \mathcal{C}_1$ such that all objects in C except for one, denoted by $\boldsymbol{o}$, belong to the same category according to ground truth. Consider a clustering $\mathcal{C}_2$ identical to $\mathcal{C}_1$ except that $\boldsymbol{o}$ is assigned to a cluster $C' \neq C$ in $\mathcal{C}_2$ such that C' contains objects from various categories according to ground truth, and thus is noisy. In other words, C' in $\mathcal{C}_2$ is a rag bag. Then, a clustering quality measure Q respecting the rag bag criterion should give a higher score to $\mathcal{C}_2$, that is, $Q(\mathcal{C}_2,\mathcal{C}_g) > Q(\mathcal{C}_1,\mathcal{C}_g)$.
- **Small cluster preservation.** If a small category is split into small pieces in a clustering, those small pieces may likely become noise and thus the small category cannot be discovered from the clustering. The small cluster preservation criterion states that splitting a small category into pieces is more harmful than splitting a large category into pieces. Consider an extreme case. Let D be a data set of $n+2$ objects such that, according to ground truth, n objects, denoted by $\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}$, belong to one category and the other two objects, denoted by $\boldsymbol{o_{n+1}}, \boldsymbol{o_{n+2}}$, belong to another category. Suppose clustering $\mathcal{C}_1$ has three clusters, $C_1 = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}\}$, $C_2 = \{\boldsymbol{o_{n+1}}\}$, and $C_3 = \{\boldsymbol{o_{n+2}}\}$. Let clustering $\mathcal{C}_2$ have three clusters, too, namely $C_1 = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_{n-1}}\}$, $C_2 = \{\boldsymbol{o_n}\}$, and $C_3 = \{\boldsymbol{o_{n+1}}, \boldsymbol{o_{n+2}}\}$. In other words, C_1 splits the small category and C_2 splits the big category. A clustering quality measure Q preserving small clusters should give a higher score to $\mathcal{C}_2$, that is, $Q(\mathcal{C}_2,\mathcal{C}_g) > Q(\mathcal{C}_1,\mathcal{C}_g)$.

Many clustering quality measures satisfy some of these four criteria. Here, we introduce the *BCubed precision* and *recall* metrics, which satisfy all four criteria.

BCubed evaluates the precision and recall for every object in a clustering on a given data set according to ground truth. The precision of an object indicates how many other objects in the same cluster belong to the same category as the object. The recall

of an object reflects how many objects of the same category are assigned to the same cluster.

Formally, let $D = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}\}$ be a set of objects, and $\mathcal{C}$ be a clustering on D. Let $L(\boldsymbol{o_i})$ $(1 \leq i \leq n)$ be the category of $\boldsymbol{o_i}$ given by ground truth, and $C(o_i)$ be the *cluster_ID* of $\boldsymbol{o_i}$ in $\mathcal{C}$. Then, for two objects, $\boldsymbol{o_i}$ and $\boldsymbol{o_j}$, $(1 \leq i, j, \leq n, i \neq j)$, the *correctness* of the relation between $\boldsymbol{o_i}$ and $\boldsymbol{o_j}$ in clustering $\mathcal{C}$ is given by

$$\text{Correctness}(\boldsymbol{o_i}, \boldsymbol{o_j}) = \begin{cases} 1 & \text{if } L(\boldsymbol{o_i}) = L(\boldsymbol{o_j}) \Leftrightarrow C(\boldsymbol{o_i}) = C(\boldsymbol{o_j}) \\ 0 & \text{otherwise.} \end{cases} \tag{10.28}$$

BCubed precision is defined as

$$\text{Precision BCubed} = \frac{\sum_{i=1}^{n} \frac{\sum_{\boldsymbol{o_j}: i \neq j, C(\boldsymbol{o_i}) = C(\boldsymbol{o_j})} \text{Correctness}(\boldsymbol{o_i}, \boldsymbol{o_j})}{\|\{\boldsymbol{o_j} | i \neq j, C(\boldsymbol{o_i}) = C(\boldsymbol{o_j})\}\|}}{n}. \tag{10.29}$$

BCubed recall is defined as

$$\text{Recall BCubed} = \frac{\sum_{i=1}^{n} \frac{\sum_{\boldsymbol{o_j}: i \neq j, L(\boldsymbol{o_i}) = L(\boldsymbol{o_j})} \text{Correctness}(\boldsymbol{o_i}, \boldsymbol{o_j})}{\|\{\boldsymbol{o_j} | i \neq j, L(\boldsymbol{o_i}) = L(\boldsymbol{o_j})\}\|}}{n}. \tag{10.30}$$

Intrinsic Methods

When the ground truth of a data set is not available, we have to use an intrinsic method to assess the clustering quality. In general, intrinsic methods evaluate a clustering by examining how well the clusters are separated and how compact the clusters are. Many intrinsic methods have the advantage of a similarity metric between objects in the data set.

The **silhouette coefficient** is such a measure. For a data set, D, of n objects, suppose D is partitioned into k clusters, $C_1, \ldots, C_k$. For each object $\boldsymbol{o} \in D$, we calculate $a(\boldsymbol{o})$ as the average distance between $\boldsymbol{o}$ and all other objects in the cluster to which $\boldsymbol{o}$ belongs. Similarly, $b(\boldsymbol{o})$ is the minimum average distance from $\boldsymbol{o}$ to all clusters to which $\boldsymbol{o}$ does not belong. Formally, suppose $\boldsymbol{o} \in C_i$ $(1 \leq i \leq k)$; then

$$a(\boldsymbol{o}) = \frac{\sum_{\boldsymbol{o'} \in C_i, \boldsymbol{o} \neq \boldsymbol{o'}} dist(\boldsymbol{o}, \boldsymbol{o'})}{|C_i| - 1} \tag{10.31}$$

and

$$b(\boldsymbol{o}) = \min_{C_j:1\le j\le k, j\neq i} \left\{ \frac{\sum_{\boldsymbol{o}'\in C_j} dist(\boldsymbol{o},\boldsymbol{o}')}{|C_j|} \right\}. \tag{10.32}$$

The **silhouette coefficient** of $\boldsymbol{o}$ is then defined as

$$s(\boldsymbol{o}) = \frac{b(\boldsymbol{o}) - a(\boldsymbol{o})}{\max\{a(\boldsymbol{o}), b(\boldsymbol{o})\}}. \tag{10.33}$$

The value of the silhouette coefficient is between -1 and 1. The value of $\boldsymbol{a(o)}$ reflects the compactness of the cluster to which $\boldsymbol{o}$ belongs. The smaller the value, the more compact the cluster. The value of $b(\boldsymbol{o})$ captures the degree to which $\boldsymbol{o}$ is separated from other clusters. The larger $b(\boldsymbol{o})$ is, the more separated $\boldsymbol{o}$ is from other clusters. Therefore, when the silhouette coefficient value of $\boldsymbol{o}$ approaches 1, the cluster containing $\boldsymbol{o}$ is compact and $\boldsymbol{o}$ is far away from other clusters, which is the preferable case. However, when the silhouette coefficient value is negative (i.e., $b(\boldsymbol{o}) < \mathrm{a}(\boldsymbol{o})$), this means that, in expectation, $\boldsymbol{o}$ is closer to the objects in another cluster than to the objects in the same cluster as $\boldsymbol{o}$. In many cases, this is a bad situation and should be avoided.

To measure a cluster's fitness within a clustering, we can compute the average silhouette coefficient value of all objects in the cluster. To measure the quality of a clustering, we can use the average silhouette coefficient value of all objects in the data set. The silhouette coefficient and other intrinsic measures can also be used in the elbow method to heuristically derive the number of clusters in a data set by replacing the sum of within-cluster variances.

10.7 Summary

- A **cluster** is a collection of data objects that are *similar* to one another within the same cluster and are *dissimilar* to the objects in other clusters. The process of grouping a set of physical or abstract objects into classes of *similar* objects is called **clustering**.
- Cluster analysis has extensive **applications**, including business intelligence, image pattern recognition, Web search, biology, and security. Cluster analysis can be used as a standalone data mining tool to gain insight into the data distribution, or as a preprocessing step for other data mining algorithms operating on the detected clusters.
- Clustering is a dynamic field of research in data mining. It is related to **unsupervised learning** in machine learning.
- Clustering is a challenging field. Typical **requirements** of it include scalability, the ability to deal with different types of data and attributes, the discovery of clusters in arbitrary shape, minimal requirements for domain knowledge to determine input parameters, the ability to deal with noisy data, incremental clustering and

insensitivity to input order, the capability of clustering high-dimensionality data, constraint-based clustering, as well as interpretability and usability.

- Many clustering algorithms have been developed. These can be categorized from several **orthogonal aspects** such as those regarding partitioning criteria, separation of clusters, similarity measures used, and clustering space. This chapter discusses major fundamental clustering methods of the following categories: *partitioning methods, hierarchical methods, density-based methods*, and *grid-based methods*. Some algorithms may belong to more than one category.
- A **partitioning method** first creates an initial set of k partitions, where parameter k is the number of partitions to construct. It then uses an *iterative relocation technique* that attempts to improve the partitioning by moving objects from one group to another. Typical partitioning methods include k-means, k-medoids, and CLARANS.
- A **hierarchical method** creates a hierarchical decomposition of the given set of data objects. The method can be classified as being either *agglomerative* (*bottom-up*) or *divisive* (*top-down*), based on how the hierarchical decomposition is formed. To compensate for the rigidity of *merge* or *split*, the quality of hierarchical agglomeration can be improved by analyzing object linkages at each hierarchical partitioning (e.g., in Chameleon), or by first performing *microclustering* (that is, grouping objects into "microclusters") and then operating on the microclusters with other clustering techniques such as iterative relocation (as in BIRCH).
- A **density-based method** clusters objects based on the notion of density. It grows clusters either according to the density of neighborhood objects (e.g., in DBSCAN) or according to a density function (e.g., in DENCLUE). OPTICS is a density-based method that generates an augmented ordering of the data's clustering structure.
- A **grid-based method** first quantizes the object space into a finite number of cells that form a grid structure, and then performs clustering on the grid structure. STING is a typical example of a grid-based method based on statistical information stored in grid cells. CLIQUE is a grid-based and subspace clustering algorithm.
- **Clustering evaluation** assesses the feasibility of clustering analysis on a data set and the quality of the results generated by a clustering method. The tasks include assessing clustering tendency, determining the number of clusters, and measuring clustering quality.

10.8 Exercises

10.1 Briefly describe and give examples of each of the following approaches to clustering: *partitioning* methods, *hierarchical* methods, *density-based* methods, and *grid-based* methods.

10.2 Suppose that the data mining task is to cluster points (with (x, y) representing location) into three clusters, where the points are

$$A_1(2,10), A_2(2,5), A_3(8,4), B_1(5,8), B_2(7,5), B_3(6,4), C_1(1,2), C_2(4,9).$$

The distance function is Euclidean distance. Suppose initially we assign A_1, B_1, and C_1 as the center of each cluster, respectively. Use the *k-means* algorithm to show *only*

(a) The three cluster centers after the first round of execution.

(b) The final three clusters.

10.3 Use an example to show why the *k*-means algorithm may not find the global optimum, that is, optimizing the within-cluster variation.

10.4 For the *k*-means algorithm, it is interesting to note that by choosing the initial cluster centers carefully, we may be able to not only speed up the algorithm's convergence, but also guarantee the quality of the final clustering. The ***k*-means++** algorithm is a variant of *k*-means, which chooses the initial centers as follows. First, it selects one center uniformly at random from the objects in the data set. Iteratively, for each object $\boldsymbol{p}$ other than the chosen center, it chooses an object as the new center. This object is chosen at random with probability proportional to $dist(\boldsymbol{p})^2$, where $dist(\boldsymbol{p})$ is the distance from $\boldsymbol{p}$ to the closest center that has already been chosen. The iteration continues until k centers are selected.

Explain why this method will not only speed up the convergence of the *k*-means algorithm, but also guarantee the quality of the final clustering results.

10.5 Provide the pseudocode of the object reassignment step of the PAM algorithm.

10.6 Both *k-means* and *k-medoids* algorithms can perform effective clustering.

(a) Illustrate the strength and weakness of *k-means* in comparison with *k-medoids*.

(b) Illustrate the strength and weakness of these schemes in comparison with a hierarchical clustering scheme (e.g., AGNES).

10.7 Prove that in DBSCAN, the density-connectedness is an equivalence relation.

10.8 Prove that in DBSCAN, for a fixed *MinPts* value and two neighborhood thresholds, $\epsilon_1 < \epsilon_2$, a cluster C with respect to ϵ_1 and *MinPts* must be a subset of a cluster C' with respect to ϵ_2 and *MinPts*.

10.9 Provide the pseudocode of the OPTICS algorithm.

10.10 Why is it that BIRCH encounters difficulties in finding clusters of arbitrary shape but OPTICS does not? Propose modifications to BIRCH to help it find clusters of arbitrary shape.

10.11 Provide the pseudocode of the step in CLIQUE that finds dense cells in all subspaces.

10.12 Present conditions under which density-based clustering is more suitable than partitioning-based clustering and hierarchical clustering. Give application examples to support your argument.

10.13 Give an example of how specific clustering methods can be *integrated*, for example, where one clustering algorithm is used as a preprocessing step for another. In addition, provide reasoning as to why the integration of two methods may sometimes lead to improved clustering quality and efficiency.

10.14 Clustering is recognized as an important data mining task with broad applications. Give one application example for each of the following cases:

(a) An application that uses clustering as a major data mining function.

(b) An application that uses clustering as a preprocessing tool for data preparation for other data mining tasks.

10.15 Data cubes and multidimensional databases contain nominal, ordinal, and numeric data in hierarchical or aggregate forms. Based on what you have learned about the clustering methods, design a clustering method that finds clusters in large data cubes effectively and efficiently.

10.16 Describe each of the following clustering algorithms in terms of the following criteria: (1) shapes of clusters that can be determined; (2) input parameters that must be specified; and (3) limitations.

(a) *k*-means

(b) *k*-medoids

(c) CLARA

(d) BIRCH

(e) CHAMELEON

(f) DBSCAN

10.17 Human eyes are fast and effective at judging the quality of clustering methods for 2-D data. Can you design a data visualization method that may help humans visualize data clusters and judge the clustering quality for 3-D data? What about for even higher-dimensional data?

10.18 Suppose that you are to allocate a number of automatic teller machines (ATMs) in a given region so as to satisfy a number of constraints. Households or workplaces may be clustered so that typically one ATM is assigned per cluster. The clustering, however, may be constrained by two factors: (1) obstacle objects (i.e., there are bridges, rivers, and highways that can affect ATM accessibility), and (2) additional user-specified constraints such as that each ATM should serve at least 10,000 households. How can a clustering algorithm such as *k*-means be modified for quality clustering under *both* constraints?

10.19 For *constraint-based clustering*, aside from having the minimum number of customers in each cluster (for ATM allocation) as a constraint, there can be many other kinds of

constraints. For example, a constraint could be in the form of the maximum number of customers per cluster, average income of customers per cluster, maximum distance between every two clusters, and so on. Categorize the kinds of constraints that can be imposed on the clusters produced and discuss how to perform clustering efficiently under such kinds of constraints.

10.20 Design a *privacy-preserving clustering* method so that a data owner would be able to ask a third party to mine the data for quality clustering without worrying about the potential inappropriate disclosure of certain private or sensitive information stored in the data.

10.21 Show that BCubed metrics satisfy the four essential requirements for extrinsic clustering evaluation methods.

10.9 Bibliographic Notes

Clustering has been extensively studied for over 40 years and across many disciplines due to its broad applications. Most books on pattern classification and machine learning contain chapters on cluster analysis or unsupervised learning. Several textbooks are dedicated to the methods of cluster analysis, including Hartigan [Har75]; Jain and Dubes [JD88]; Kaufman and Rousseeuw [KR90]; and Arabie, Hubert, and De Sorte [AHS96]. There are also many survey articles on different aspects of clustering methods. Recent ones include Jain, Murty, and Flynn [JMF99]; Parsons, Haque, and Liu [PHL04]; and Jain [Jai10].

For partitioning methods, the *k*-means algorithm was first introduced by Lloyd [Llo57], and then by MacQueen [Mac67]. Arthur and Vassilvitskii [AV07] presented the *k*-means++ algorithm. A filtering algorithm, which uses a spatial hierarchical data index to speed up the computation of cluster means, is given in Kanungo, Mount, Netanyahu, et al. [KMN+02].

The *k*-medoids algorithms of PAM and CLARA were proposed by Kaufman and Rousseeuw [KR90]. The *k*-modes (for clustering nominal data) and *k*-prototypes (for clustering hybrid data) algorithms were proposed by Huang [Hua98]. The *k*-modes clustering algorithm was also proposed independently by Chaturvedi, Green, and Carroll [CGC94, CGC01]. The CLARANS algorithm was proposed by Ng and Han [NH94]. Ester, Kriegel, and Xu [EKX95] proposed techniques for further improvement of the performance of CLARANS using efficient spatial access methods such as R*-tree and focusing techniques. A *k*-means-based scalable clustering algorithm was proposed by Bradley, Fayyad, and Reina [BFR98].

An early survey of agglomerative hierarchical clustering algorithms was conducted by Day and Edelsbrunner [DE84]. Agglomerative hierarchical clustering, such as AGNES, and divisive hierarchical clustering, such as DIANA, were introduced by Kaufman and Rousseeuw [KR90]. An interesting direction for improving the clustering quality of hierarchical clustering methods is to integrate hierarchical clustering with distance-based iterative relocation or other nonhierarchical clustering methods. For example, BIRCH, by Zhang, Ramakrishnan, and Livny [ZRL96], first performs hierarchical clustering with

a CF-tree before applying other techniques. Hierarchical clustering can also be performed by sophisticated linkage analysis, transformation, or nearest-neighbor analysis, such as CURE by Guha, Rastogi, and Shim [GRS98]; ROCK (for clustering nominal attributes) by Guha, Rastogi, and Shim [GRS99]; and Chameleon by Karypis, Han, and Kumar [KHK99].

A probabilistic hierarchical clustering framework following normal linkage algorithms and using probabilistic models to define cluster similarity was developed by Friedman [Fri03] and Heller and Ghahramani [HG05].

For density-based clustering methods, DBSCAN was proposed by Ester, Kriegel, Sander, and Xu [EKSX96]. Ankerst, Breunig, Kriegel, and Sander [ABKS99] developed OPTICS, a cluster-ordering method that facilitates density-based clustering without worrying about parameter specification. The DENCLUE algorithm, based on a set of density distribution functions, was proposed by Hinneburg and Keim [HK98]. Hinneburg and Gabriel [HG07] developed DENCLUE 2.0, which includes a new hill-climbing procedure for Gaussian kernels that adjusts the step size automatically.

STING, a grid-based multiresolution approach that collects statistical information in grid cells, was proposed by Wang, Yang, and Muntz [WYM97]. WaveCluster, developed by Sheikholeslami, Chatterjee, and Zhang [SCZ98], is a multiresolution clustering approach that transforms the original feature space by wavelet transform.

Scalable methods for clustering nominal data were studied by Gibson, Kleinberg, and Raghavan [GKR98]; Guha, Rastogi, and Shim [GRS99]; and Ganti, Gehrke, and Ramakrishnan [GGR99]. There are also many other clustering paradigms. For example, fuzzy clustering methods are discussed in Kaufman and Rousseeuw [KR90], Bezdek [Bez81], and Bezdek and Pal [BP92].

For high-dimensional clustering, an Apriori-based dimension-growth subspace clustering algorithm called CLIQUE was proposed by Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98]. It integrates density-based and grid-based clustering methods.

Recent studies have proceeded to clustering stream data Babcock, Badu, Datar, et al. [BBD+02]. A *k*-median-based data stream clustering algorithm was proposed by Guha, Mishra, Motwani, and O'Callaghan [GMMO00] and by O'Callaghan et al. [OMM+02]. A method for clustering evolving data streams was proposed by Aggarwal, Han, Wang, and Yu [AHWY03]. A framework for projected clustering of high-dimensional data streams was proposed by Aggarwal, Han, Wang, and Yu [AHWY04a].

Clustering evaluation is discussed in a few monographs and survey articles such as Jain and Dubes [JD88] and Halkidi, Batistakis, and Vazirgiannis [HBV01]. The extrinsic methods for clustering quality evaluation are extensively explored. Some recent studies include Meilă [Mei03, Mei05] and Amigó, Gonzalo, Artiles, and Verdejo [AGAV09]. The four essential criteria introduced in this chapter are formulated in Amigó, Gonzalo, Artiles, and Verdejo [AGAV09], while some individual criteria were also mentioned earlier, for example, in Meilă [Mei03] and Rosenberg and Hirschberg [RH07]. Bagga and Baldwin [BB98] introduced the BCubed metrics. The silhouette coefficient is described in Kaufman and Rousseeuw [KR90].

This page intentionally left blank