# 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<br>– Distance-based<br>– May use mean or medoid (etc.) to represent cluster center<br>– Effective for small- to medium-size data sets |
| Hierarchical methods | – Clustering is a hierarchical decomposition (i.e., multiple levels)<br>– Cannot correct erroneous merges or splits<br>– May incorporate other techniques like microclustering or consider object "linkages" |
| Density-based methods | – Can find arbitrarily shaped clusters<br>– Clusters are dense regions of objects in space that are separated by low-density regions<br>– Cluster density: Each point must have a minimum number of points within its "neighborhood"<br>– May filter out outliers |
| Grid-based methods | – Use a multiresolution grid data structure<br>– 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:**

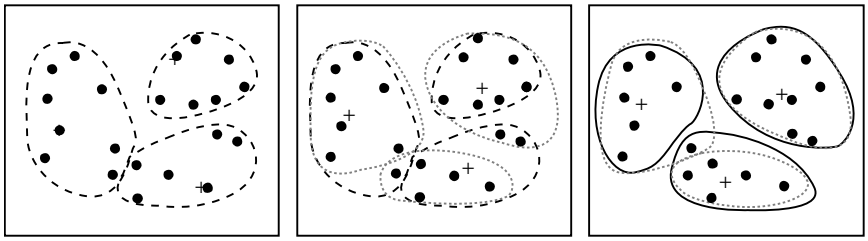
- $k$: the number of clusters,
- $D$: a data set containing $n$ objects.

**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 $\boldsymbol{o_i}$, $i \neq j$?

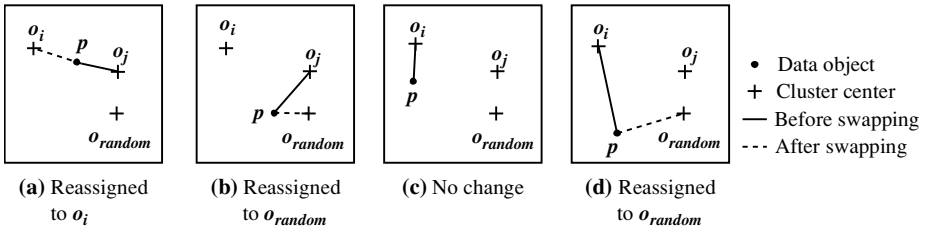



**Figure 10.4** Four cases of the cost function for $k$-medoids clustering.

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 $O(k(n-k)^2)$. 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

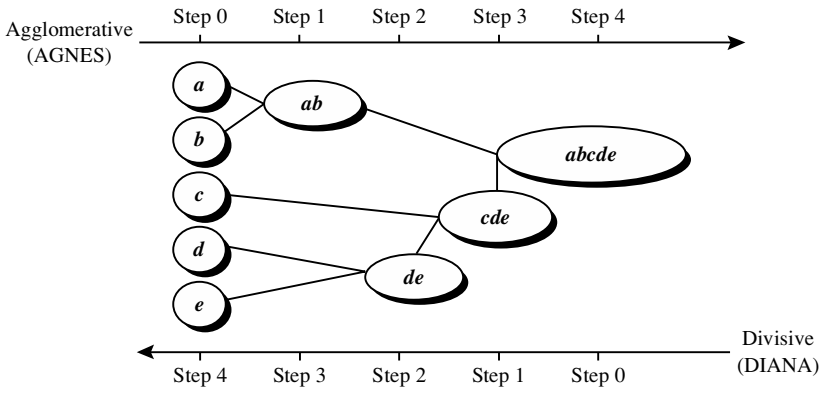



**Figure 10.6** Agglomerative and divisive hierarchical clustering on data objects $\{a, b, c, d, e\}$.

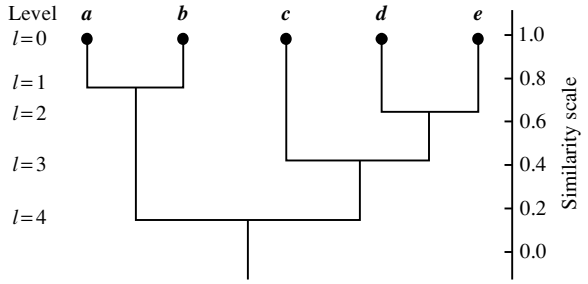



**Figure 10.7** Dendrogram representation for hierarchical clustering of data objects $\{a, b, c, d, e\}$.

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

## 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 $\epsilon$**-neighborhood** of an object $\boldsymbol{o}$ is the space within a radius $\epsilon$ centered at $\boldsymbol{o}$.

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

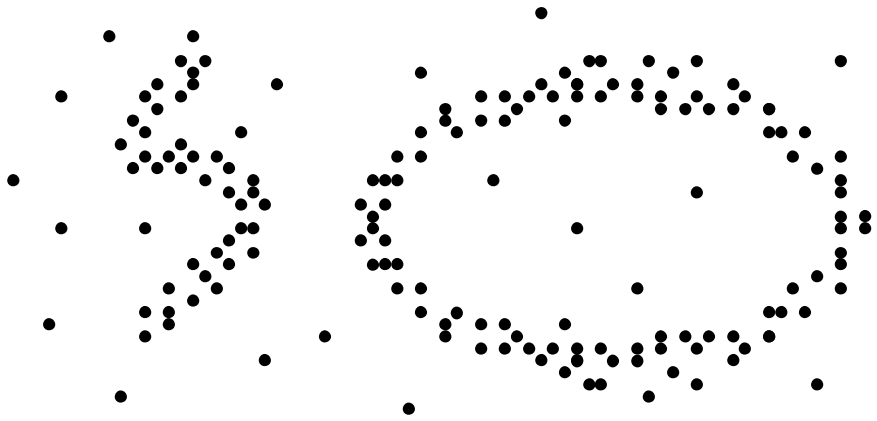


**Figure 10.13** Clusters of arbitrary shape.

parameter, *MinPts*, which specifies the density threshold of dense regions. An object is a **core object** if the $\epsilon$-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, $\epsilon$ and *MinPts*. The clustering task is therein reduced to using core objects and their neighborhoods to form dense regions, where the dense regions are clusters. 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 $\epsilon$ and *MinPts*) if $\boldsymbol{p}$ is within the $\epsilon$-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 $\epsilon$-neighborhood of $\boldsymbol{q}$. Using the directly density-reachable relation, a core object can "bring" all objects from its $\epsilon$-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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 $\epsilon$-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.

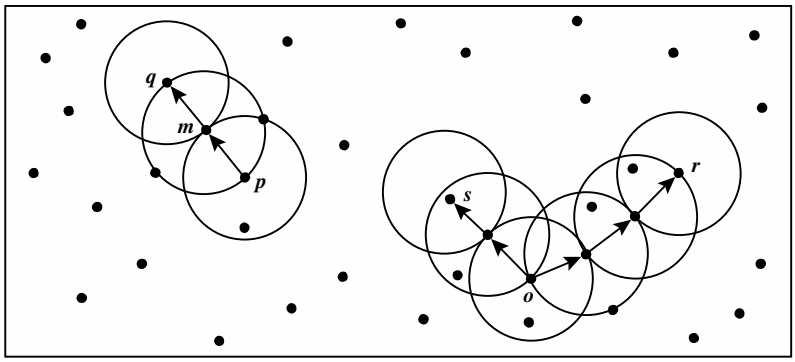



**Figure 10.14** Density-reachability and density-connectivity in density-based clustering. *Source:* Based on Ester, Kriegel, Sander, and Xu [EKSX96].

*"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 $\epsilon$-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 $\epsilon$-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 $\epsilon$-neighborhood. If the $\epsilon$-neighborhood of $\boldsymbol{p'}$ has at least *MinPts* objects, those objects in the $\epsilon$-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, $\epsilon$ 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 $\epsilon$ (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

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

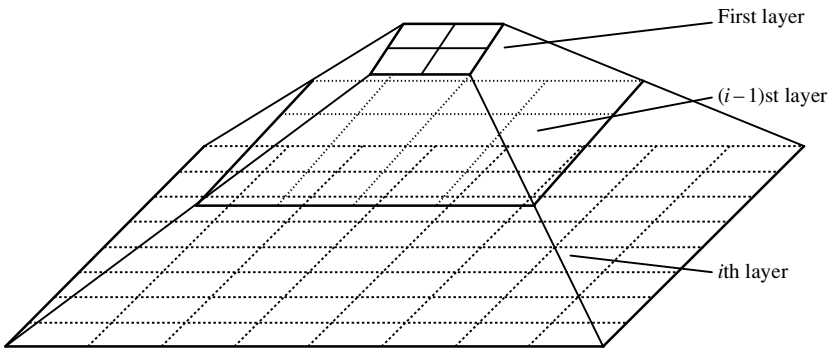



**Figure 10.19** Hierarchical structure for STING clustering.

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

"*How is this statistical information useful for query answering?*" The statistical parameters can be used in a top-down, grid-based manner as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated probability range) reflecting the 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,

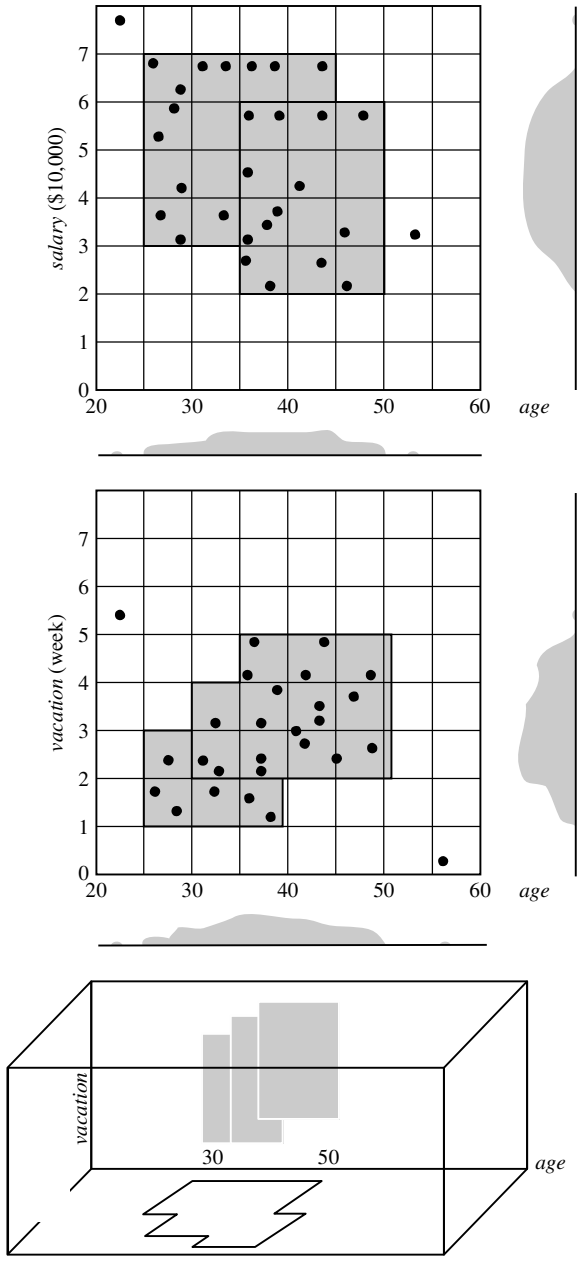



**Figure 10.20** Dense units found with respect to *age* for the dimensions *salary* and *vacation* are intersected to provide a candidate search space for dense units of higher dimensionality.

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