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

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

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

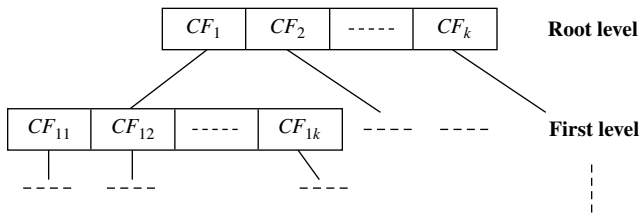

Figure 10.9 CF-tree structure.

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

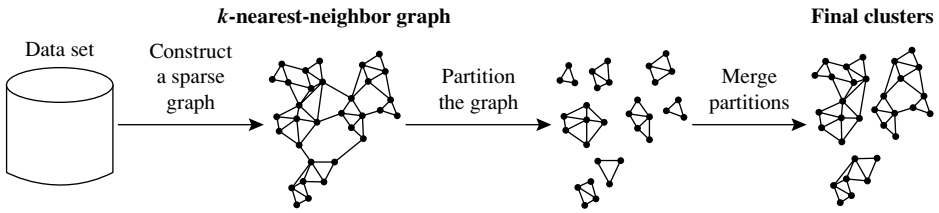

Figure 10.10 Chameleon: hierarchical clustering based on k-nearest neighbors and dynamic modeling. *Source:* Based on Karypis, Han, and Kumar [KHK99].

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

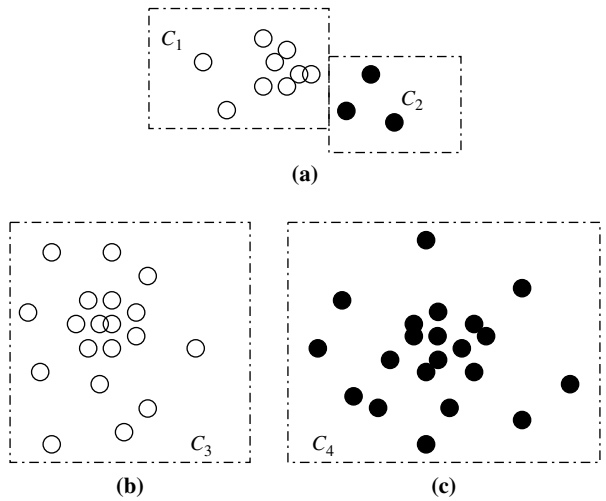

Figure 10.11 Merging clusters in probabilistic hierarchical clustering: (a) Merging clusters C_1 and C_2 leads to an increase in overall cluster quality, but merging clusters (b) C_3 and (c) C_4 does not.

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

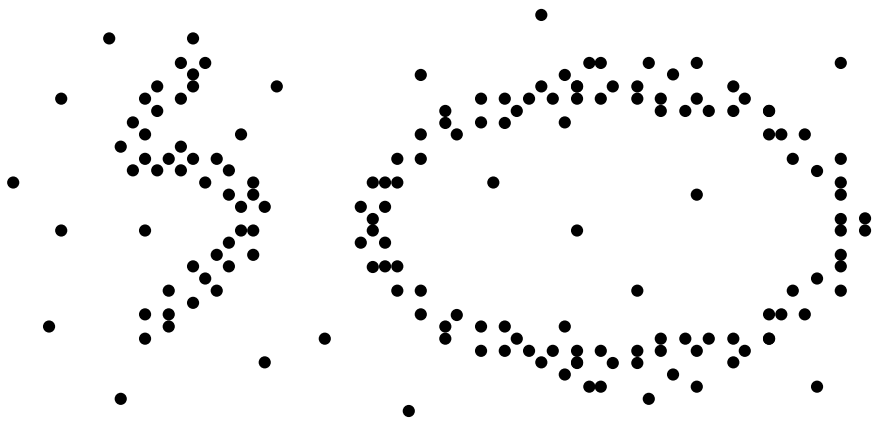

Figure 10.13 Clusters of arbitrary shape.

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.

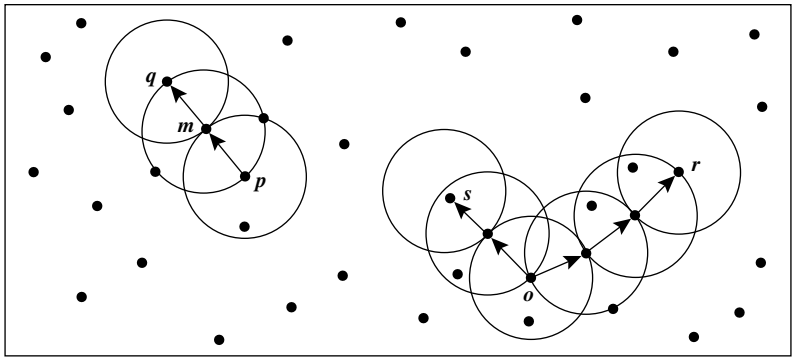

Figure 10.14 Density-reachability and density-connectivity in density-based clustering. *Source:* Based on Ester, Kriegel, Sander, and Xu [EKSX96].

"*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.

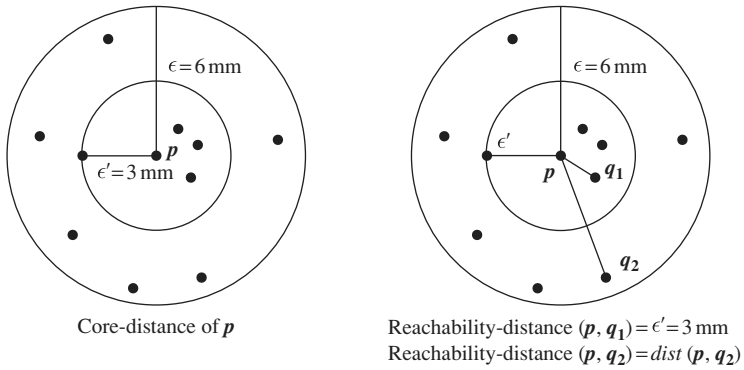

Figure 10.16 OPTICS terminology. *Source:* Based on Ankerst, Breunig, Kriegel, and Sander [ABKS99].

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.

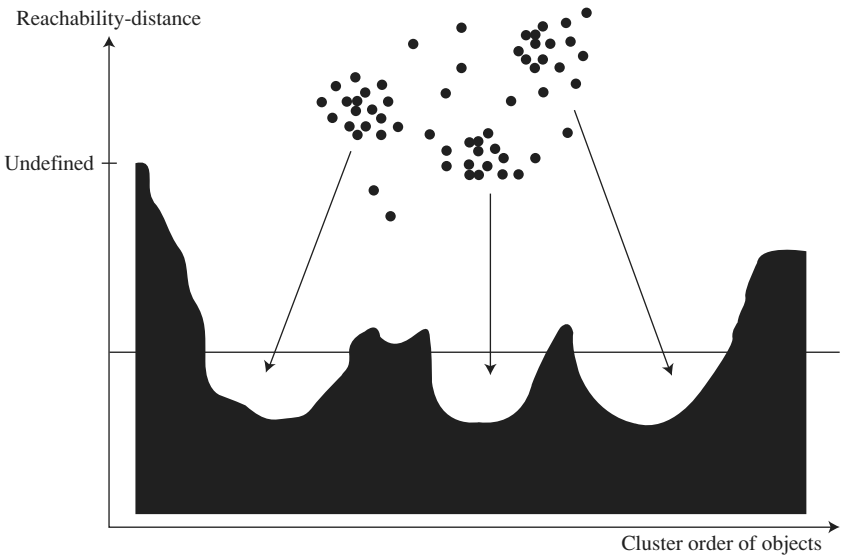

Figure 10.17 Cluster ordering in OPTICS. *Source:* Adapted from Ankerst, Breunig, Kriegel, and Sander [ABKS99].

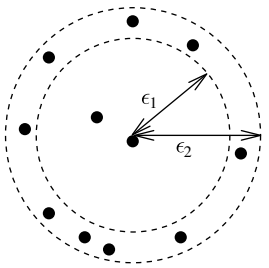

Figure 10.18 The subtlety in density estimation in DBSCAN and OPTICS: Increasing the neighborhood radius slightly from ϵ_1 to ϵ_2 results in a much higher density.

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

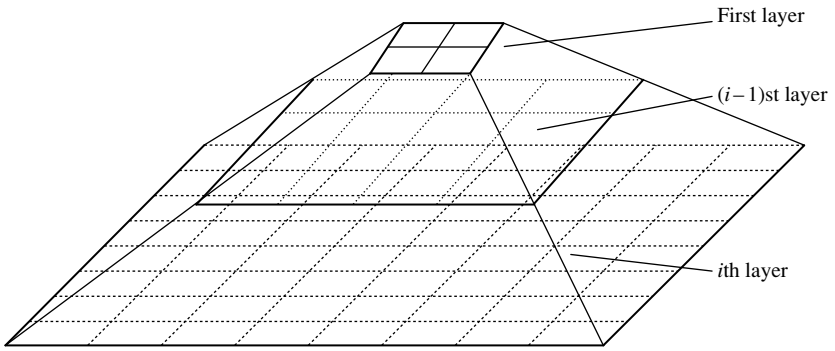

Figure 10.19 Hierarchical structure for STING clustering.

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,

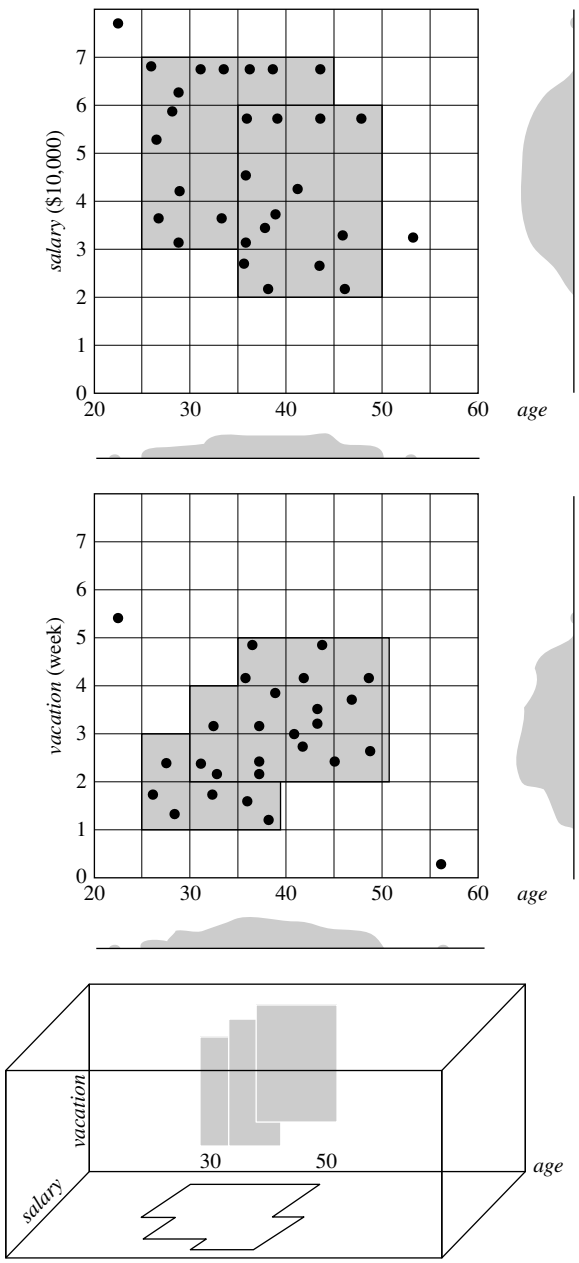

Figure 10.20 Dense units found with respect to *age* for the dimensions *salary* and *vacation* are intersected to provide a candidate search space for dense units of higher dimensionality.

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

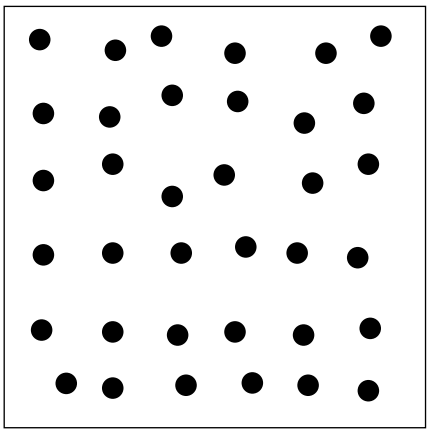

Figure 10.21 A data set that is uniformly distributed in the data space.

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., $b(\boldsymbol{o}) < a(\boldsymbol{o})$), 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

Advanced Cluster Analysis

You learned the fundamentals of cluster analysis in Chapter 10. In this chapter, we discuss advanced topics of cluster analysis. Specifically, we investigate four major perspectives:

- **Probabilistic model-based clustering**: Section 11.1 introduces a general framework and a method for deriving clusters where each object is assigned a probability of belonging to a cluster. Probabilistic model-based clustering is widely used in many data mining applications such as text mining.
- **Clustering high-dimensional data**: When the dimensionality is high, conventional distance measures can be dominated by noise. Section 11.2 introduces fundamental methods for cluster analysis on high-dimensional data.
- **Clustering graph and network data**: Graph and network data are increasingly popular in applications such as online social networks, the World Wide Web, and digital libraries. In Section 11.3, you will study the key issues in clustering graph and network data, including similarity measurement and clustering methods.
- **Clustering with constraints**: In our discussion so far, we do not assume any constraints in clustering. In some applications, however, various constraints may exist. These constraints may rise from background knowledge or spatial distribution of the objects. You will learn how to conduct cluster analysis with different kinds of constraints in Section 11.4.

By the end of this chapter, you will have a good grasp of the issues and techniques regarding advanced cluster analysis.

11.1 Probabilistic Model-Based Clustering

In all the cluster analysis methods we have discussed so far, each data object can be assigned to only one of a number of clusters. This cluster assignment rule is required in some applications such as assigning customers to marketing managers. However,

in other applications, this rigid requirement may not be desirable. In this section, we demonstrate the need for fuzzy or flexible cluster assignment in some applications, and introduce a general method to compute probabilistic clusters and assignments.

"*In what situations may a data object belong to more than one cluster?*" Consider Example 11.1.

Example 11.1 **Clustering product reviews.** *AllElectronics* has an online store, where customers not only purchase online, but also create reviews of products. Not every product receives reviews; instead, some products may have many reviews, while many others have none or only a few. Moreover, a review may involve multiple products. Thus, as the review editor of *AllElectronics*, your task is to cluster the reviews.

Ideally, a cluster is about a *topic*, for example, a group of products, services, or issues that are highly related. Assigning a review to one cluster exclusively would not work well for your task. Suppose there is a cluster for "cameras and camcorders" and another for "computers." What if a review talks about the compatibility between a camcorder and a computer? The review relates to both clusters; however, it does not exclusively belong to either cluster.

You would like to use a clustering method that allows a review to belong to more than one cluster if the review indeed involves more than one topic. To reflect the strength that a review belongs to a cluster, you want the assignment of a review to a cluster to carry a weight representing the partial membership. ■

The scenario where an object may belong to multiple clusters occurs often in many applications. This is illustrated in Example 11.2.

Example 11.2 **Clustering to study user search intent**. The *AllElectronics* online store records all customer browsing and purchasing behavior in a log. An important data mining task is to use the log data to categorize and understand *user search intent*. For example, consider a user *session* (a short period in which a user interacts with the online store). Is the user searching for a product, making comparisons among different products, or looking for customer support information? Clustering analysis helps here because it is difficult to predefine user behavior patterns thoroughly. A cluster that contains similar user browsing trajectories may represent similar user behavior.

However, not every session belongs to only one cluster. For example, suppose user sessions involving the purchase of digital cameras form one cluster, and user sessions that compare laptop computers form another cluster. What if a user in one session makes an order for a digital camera, and at the same time compares several laptop computers? Such a session should belong to both clusters to some extent. ■

In this section, we systematically study the theme of clustering that allows an object to belong to more than one cluster. We start with the notion of fuzzy clusters in Section 11.1.1. We then generalize the concept to probabilistic model-based clusters in Section 11.1.2. In Section 11.1.3, we introduce the expectation-maximization algorithm, a general framework for mining such clusters.

11.1.1 Fuzzy Clusters

Given a set of objects, $X = \{x_1, \ldots, x_n\}$, a **fuzzy set** S is a subset of X that allows each object in X to have a membership degree between 0 and 1. Formally, a fuzzy set, S, can be modeled as a function, $F_S{:}X \rightarrow [0, 1]$.

Example 11.3 **Fuzzy set**. The more digital camera units that are sold, the more popular the camera is. In *AllElectronics*, we can use the following formula to compute the degree of popularity of a digital camera, o, given the sales of o:

$$pop(o) = \begin{cases} 1 & \text{if 1000 or more units of } o \text{ are sold} \\ \frac{i}{1000} & \text{if } i \ (i < 1000) \text{ units of } o \text{ are sold.} \end{cases} \tag{11.1}$$

Function $pop()$ defines a fuzzy set of popular digital cameras. For example, suppose the sales of digital cameras at *AllElectronics* are as shown in Table 11.1. The fuzzy set of popular digital cameras is $\{A(0.05), B(1), C(0.86), D(0.27)\}$, where the degrees of membership are written in parentheses. ■

We can apply the fuzzy set idea on clusters. That is, given a set of objects, a cluster is a fuzzy set of objects. Such a cluster is called a fuzzy cluster. Consequently, a clustering contains multiple *fuzzy clusters*.

Formally, given a set of objects, $o_1, \ldots, o_n$, a **fuzzy clustering** of k **fuzzy clusters**, $C_1, \ldots, C_k$, can be represented using a **partition matrix**, $M = [w_{ij}]$ $(1 \leq i \leq n, 1 \leq j \leq k)$, where w_{ij} is the membership degree of o_i in fuzzy cluster C_j. The partition matrix should satisfy the following three requirements:

- For each object, o_i, and cluster, C_j, $0 \leq w_{ij} \leq 1$. This requirement enforces that a fuzzy cluster is a fuzzy set.
- For each object, o_i, $\sum_{j=1}^{k} w_{ij} = 1$. This requirement ensures that every object participates in the clustering equivalently.

Table 11.1 Set of Digital Cameras and Their Sales at *AllElectronics*

Camera	*Sales (units)*
A	50
B	1320
C	860
D	270

- For each cluster, C_j, $0 < \sum_{i=1}^{n} w_{ij} < n$. This requirement ensures that for every cluster, there is at least one object for which the membership value is nonzero.

Example 11.4 **Fuzzy clusters**. Suppose the *AllElectronics* online store has six reviews. The keywords contained in these reviews are listed in Table 11.2.

We can group the reviews into two fuzzy clusters, C_1 and C_2. C_1 is for "digital camera" and "lens," and C_2 is for "computer." The partition matrix is

$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \frac{2}{3} & \frac{1}{3} \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

Here, we use the keywords "digital camera" and "lens" as the features of cluster C_1, and "computer" as the feature of cluster C_2. For review, R_i, and cluster, C_j ($1 \le i \le 6, 1 \le j \le 2$), w_{ij} is defined as

$$w_{ij} = \frac{|R_i \cap C_j|}{|R_i \cap (C_1 \cup C_2)|} = \frac{|R_i \cap C_j|}{|R_i \cap \{digital\ camera, lens, computer\}|}.$$

In this fuzzy clustering, review R_4 belongs to clusters C_1 and C_2 with membership degrees $\frac{2}{3}$ and $\frac{1}{3}$, respectively. ■

"How can we evaluate how well a fuzzy clustering describes a data set?" Consider a set of objects, $o_1, \ldots, o_n$, and a fuzzy clustering $\mathcal{C}$ of k clusters, $C_1, \ldots, C_k$. Let $M = [w_{ij}]$ ($1 \le i \le n, 1 \le j \le k$) be the partition matrix. Let $c_1, \ldots, c_k$ be the *centers* of clusters $C_1, \ldots, C_k$, respectively. Here, a center can be defined either as the mean or the medoid, or in other ways specific to the application.

As discussed in Chapter 10, the distance or similarity between an object and the center of the cluster to which the object is assigned can be used to measure how well the

Table 11.2 Set of Reviews and the Keywords Used

Review_ID	*Keywords*
R_1	digital camera, lens
R_2	digital camera
R_3	lens
R_4	digital camera, lens, computer
R_5	computer, CPU
R_6	computer, computer game

object belongs to the cluster. This idea can be extended to fuzzy clustering. For any object, o_i, and cluster, C_j, if $w_{ij} > 0$, then $dist(o_i, c_j)$ measures how well o_i is represented by c_j, and thus belongs to cluster C_j. Because an object can participate in more than one cluster, the sum of distances to the corresponding cluster centers weighted by the degrees of membership captures how well the object fits the clustering.

Formally, for an object o_i, the **sum of the squared error** (SSE) is given by

$$\text{SSE}(o_i) = \sum_{j=1}^{k} w_{ij}^{p} dist(o_i, c_j)^2, \tag{11.2}$$

where the parameter $p (p \geq 1)$ controls the influence of the degrees of membership. The larger the value of p, the larger the influence of the degrees of membership. Orthogonally, the SSE for a cluster, C_j, is

$$\text{SSE}(C_j) = \sum_{i=1}^{n} w_{ij}^{p} dist(o_i, c_j)^2. \tag{11.3}$$

Finally, the SSE of the clustering is defined as

$$\text{SSE}(\mathcal{C}) = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij}^{p} dist(o_i, c_j)^2. \tag{11.4}$$

The SSE can be used to measure how well a fuzzy clustering fits a data set.

Fuzzy clustering is also called *soft clustering* because it allows an object to belong to more than one cluster. It is easy to see that traditional (rigid) clustering, which enforces each object to belong to only one cluster exclusively, is a special case of fuzzy clustering. We defer the discussion of how to compute fuzzy clustering to Section 11.1.3.

11.1.2 Probabilistic Model-Based Clusters

"*Fuzzy clusters (Section 11.1.1) provide the flexibility of allowing an object to participate in multiple clusters. Is there a general framework to specify clusterings where objects may participate in multiple clusters in a probabilistic way?*" In this section, we introduce the general notion of probabilistic model-based clusters to answer this question.

As discussed in Chapter 10, we conduct cluster analysis on a data set because we assume that the objects in the data set in fact belong to different inherent categories. Recall that clustering tendency analysis (Section 10.6.1) can be used to examine whether a data set contains objects that may lead to meaningful clusters. Here, the inherent categories hidden in the data are *latent*, which means they cannot be directly observed. Instead, we have to infer them using the data observed. For example, the topics hidden in a set of reviews in the *AllElectronics* online store are latent because one cannot read the topics directly. However, the topics can be inferred from the reviews because each review is about one or multiple topics.

Therefore, the goal of cluster analysis is to find hidden categories. A data set that is the subject of cluster analysis can be regarded as a sample of the possible instances of the hidden categories, but without any category labels. The clusters derived from cluster analysis are inferred using the data set, and are designed to approach the hidden categories.

Statistically, we can assume that a hidden category is a distribution over the data space, which can be mathematically represented using a probability density function (or distribution function). We call such a hidden category a *probabilistic cluster*. For a probabilistic cluster, C, its probability density function, f, and a point, o, in the data space, $f(o)$ is the relative likelihood that an instance of C appears at o.

Example 11.5 **Probabilistic clusters**. Suppose the digital cameras sold by *AllElectronics* can be divided into two categories: C_1, a consumer line (e.g., point-and-shoot cameras), and C_2, a professional line (e.g., single-lens reflex cameras). Their respective probability density functions, f_1 and f_2, are shown in Figure 11.1 with respect to the attribute *price*.

For a price value of, say, \$1000, $f_1(1000)$ is the relative likelihood that the price of a consumer-line camera is \$1000. Similarly, $f_2(1000)$ is the relative likelihood that the price of a professional-line camera is \$1000.

The probability density functions, f_1 and f_2, cannot be observed directly. Instead, *AllElectronics* can only infer these distributions by analyzing the prices of the digital cameras it sells. Moreover, a camera often does not come with a well-determined category (e.g., "consumer line" or "professional line"). Instead, such categories are typically based on user background knowledge and can vary. For example, a camera in the *prosumer* segment may be regarded at the high end of the consumer line by some customers, and the low end of the professional line by others.

As an analyst at *AllElectronics*, you can consider each category as a probabilistic cluster, and conduct cluster analysis on the price of cameras to approach these categories. ■

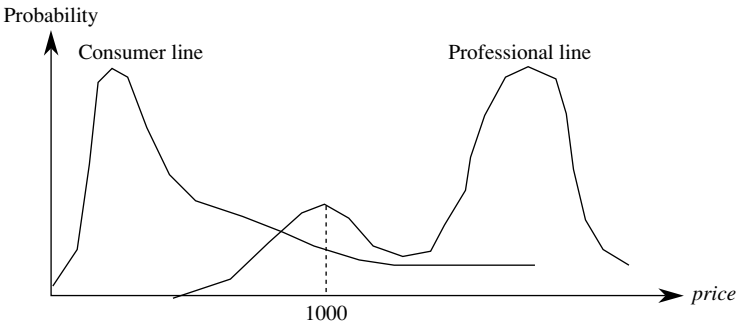

Figure 11.1 The probability density functions of two probabilistic clusters.

Suppose we want to find k probabilistic clusters, $C_1, \ldots, C_k$, through cluster analysis. For a data set, D, of n objects, we can regard D as a finite sample of the possible instances of the clusters. Conceptually, we can assume that D is formed as follows. Each cluster, C_j $(1 \leq j \leq k)$, is associated with a probability, ω_j, that some instance is sampled from the cluster. It is often assumed that $\omega_1, \ldots, \omega_k$ are given as part of the problem setting, and that $\sum_{j=1}^{k} \omega_j = 1$, which ensures that all objects are generated by the k clusters. Here, parameter ω_j captures background knowledge about the relative population of cluster C_j.

We then run the following two steps to generate an object in D. The steps are executed n times in total to generate n objects, $o_1, \ldots, o_n$, in D.

1. Choose a cluster, C_j, according to probabilities $\omega_1, \ldots, \omega_k$.

2. Choose an instance of C_j according to its probability density function, f_j.

The data generation process here is the basic assumption in mixture models. Formally, a **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters. Conceptually, each observed object is generated independently by two steps: first choosing a probabilistic cluster according to the probabilities of the clusters, and then choosing a sample according to the probability density function of the chosen cluster.

Given data set, D, and k, the number of clusters required, the task of *probabilistic model-based cluster analysis* is to infer a set of k probabilistic clusters that is most likely to generate D using this data generation process. An important question remaining is how we can measure the likelihood that a set of k probabilistic clusters and their probabilities will generate an observed data set.

Consider a set, $\boldsymbol{C}$, of k probabilistic clusters, $C_1, \ldots, C_k$, with probability density functions $f_1, \ldots, f_k$, respectively, and their probabilities, $\omega_1, \ldots, \omega_k$. For an object, o, the probability that o is generated by cluster C_j $(1 \leq j \leq k)$ is given by $P(o|C_j) = \omega_j f_j(o)$. Therefore, the probability that o is generated by the set $\boldsymbol{C}$ of clusters is

$$P(o|\boldsymbol{C}) = \sum_{j=1}^{k} \omega_j f_j(o). \tag{11.5}$$

Since the objects are assumed to have been generated independently, for a data set, $D = \{o_1, \ldots, o_n\}$, of n objects, we have

$$P(D|\boldsymbol{C}) = \prod_{i=1}^{n} P(o_i|\boldsymbol{C}) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j f_j(o_i). \tag{11.6}$$

Now, it is clear that the task of probabilistic model-based cluster analysis on a data set, D, is to find a set $\boldsymbol{C}$ of k probabilistic clusters such that $P(D|\boldsymbol{C})$ is maximized. Maximizing $P(D|\boldsymbol{C})$ is often intractable because, in general, the probability density function

of a cluster can take an arbitrarily complicated form. To make probabilistic model-based clusters computationally feasible, we often compromise by assuming that the probability density functions are parameterized distributions.

Formally, let $o_1, \ldots, o_n$ be the n observed objects, and $\Theta_1, \ldots, \Theta_k$ be the parameters of the k distributions, denoted by $\mathbf{O} = \{o_1, \ldots, o_n\}$ and $\Theta = \{\Theta_1, \ldots, \Theta_k\}$, respectively. Then, for any object, $o_i \in \mathbf{O}$ $(1 \leq i \leq n)$, Eq. (11.5) can be rewritten as

$$P(o_i|\Theta) = \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j), \tag{11.7}$$

where $P_j(o_i|\Theta_j)$ is the probability that o_i is generated from the jth distribution using parameter Θ_j. Consequently, Eq. (11.6) can be rewritten as

$$P(\mathbf{O}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \omega_j P_j(o_i|\Theta_j). \tag{11.8}$$

Using the parameterized probability distribution models, the task of probabilistic model-based cluster analysis is to infer a set of parameters, Θ, that maximizes Eq. (11.8).

Example 11.6 Univariate Gaussian mixture model. Let's use univariate Gaussian distributions as an example. That is, we assume that the probability density function of each cluster follows a 1-D Gaussian distribution. Suppose there are k clusters. The two parameters for the probability density function of each cluster are center, μ_j, and standard deviation, σ_j $(1 \leq j \leq k)$. We denote the parameters as $\Theta_j = (\mu_j, \sigma_j)$ and $\Theta = \{\Theta_1, \ldots, \Theta_k\}$. Let the data set be $\mathbf{O} = \{o_1, \ldots, o_n\}$, where o_i $(1 \leq i \leq n)$ is a real number. For any point, $o_i \in \mathbf{O}$, we have

$$P(o_i|\Theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i - \mu_j)^2}{2\sigma^2}}. \tag{11.9}$$

Assuming that each cluster has the same probability, that is $\omega_1 = \omega_2 = \cdots = \omega_k = \frac{1}{k}$, and plugging Eq. (11.9) into Eq. (11.7), we have

$$P(o_i|\Theta) = \frac{1}{k} \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i - \mu_j)^2}{2\sigma^2}}. \tag{11.10}$$

Applying Eq. (11.8), we have

$$P(\mathbf{O}|\Theta) = \frac{1}{k} \prod_{i=1}^{n} \sum_{j=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o_i - \mu_j)^2}{2\sigma^2}}. \tag{11.11}$$

The task of probabilistic model-based cluster analysis using a univariate Gaussian mixture model is to infer Θ such that Eq. (11.11) is maximized. ■

11.1.3 Expectation-Maximization Algorithm

"*How can we compute fuzzy clusterings and probabilistic model-based clusterings?*" In this section, we introduce a principled approach. Let's start with a review of the k-means clustering problem and the k-means algorithm studied in Chapter 10.

It can easily be shown that k-means clustering is a special case of fuzzy clustering (Exercise 11.1). The k-means algorithm iterates until the clustering cannot be improved. Each iteration consists of two steps:

The expectation step (E-step): Given the current cluster centers, each object is assigned to the cluster with a center that is closest to the object. Here, an object is expected to belong to the closest cluster.

The maximization step (M-step): Given the cluster assignment, for each cluster, the algorithm adjusts the center so that the sum of the distances from the objects assigned to this cluster and the new center is minimized. That is, the similarity of objects assigned to a cluster is maximized.

We can generalize this two-step method to tackle fuzzy clustering and probabilistic model-based clustering. In general, an **expectation-maximization (EM) algorithm** is a framework that approaches maximum likelihood or maximum a posteriori estimates of parameters in statistical models. In the context of fuzzy or probabilistic model-based clustering, an EM algorithm starts with an initial set of parameters and iterates until the clustering cannot be improved, that is, until the clustering converges or the change is sufficiently small (less than a preset threshold). Each iteration also consists of two steps:

- The **expectation step** assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters.
- The **maximization step** finds the new clustering or parameters that maximize the SSE in fuzzy clustering (Eq. 11.4) or the expected likelihood in probabilistic model-based clustering.

Example 11.7 **Fuzzy clustering using the EM algorithm.** Consider the six points in Figure 11.2, where the coordinates of the points are also shown. Let's compute two fuzzy clusters using the EM algorithm.

We randomly select two points, say $c_1 = a$ and $c_2 = b$, as the initial centers of the two clusters. The first iteration conducts the expectation step and the maximization step as follows.

In the **E-step**, for each point we calculate its membership degree in each cluster. For any point, o, we assign o to c_1 and c_2 with membership weights

$$\frac{\frac{1}{dist(o,c_1)^2}}{\frac{1}{dist(o,c_1)^2}+\frac{1}{dist(o,c_2)^2}} = \frac{dist(o,c_2)^2}{dist(o,c_1)^2+dist(o,c_2)^2} \text{ and } \frac{dist(o,c_1)^2}{dist(o,c_1)^2+dist(o,c_2)^2},$$

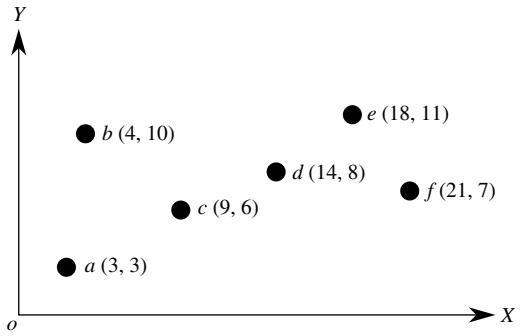

Figure 11.2 Data set for fuzzy clustering.

Table 11.3 Intermediate Results from the First Three Iterations of Example 11.7's EM Algorithm

Iteration	*E-Step*	*M-Step*
1	$M^T = \begin{bmatrix} 1 & 0 & 0.48 & 0.42 & 0.41 & 0.47 \\ 0 & 1 & 0.52 & 0.58 & 0.59 & 0.53 \end{bmatrix}$	$c_1 = (8.47, 5.12)$ $c_2 = (10.42, 8.99)$
2	$M^T = \begin{bmatrix} 0.73 & 0.49 & 0.91 & 0.26 & 0.33 & 0.42 \\ 0.27 & 0.51 & 0.09 & 0.74 & 0.67 & 0.58 \end{bmatrix}$	$c_1 = (8.51, 6.11)$ $c_2 = (14.42, 8.69)$
3	$M^T = \begin{bmatrix} 0.80 & 0.76 & 0.99 & 0.02 & 0.14 & 0.23 \\ 0.20 & 0.24 & 0.01 & 0.98 & 0.86 & 0.77 \end{bmatrix}$	$c_1 = (6.40, 6.24)$ $c_2 = (16.55, 8.64)$

respectively, where $dist(,)$ is the Euclidean distance. The rationale is that, if o is close to c_1 and $dist(o, c_1)$ is small, the membership degree of o with respect to c_1 should be high. We also normalize the membership degrees so that the sum of degrees for an object is equal to 1.

For point a, we have $w_{a,c_1} = 1$ and $w_{a,c_2} = 0$. That is, a exclusively belongs to c_1. For point b, we have $w_{b,c_1} = 0$ and $w_{b,c_2} = 1$. For point c, we have $w_{c,c_1} = \frac{41}{45+41} = 0.48$ and $w_{c,c_2} = \frac{45}{45+41} = 0.52$. The degrees of membership of the other points are shown in the partition matrix in Table 11.3. ■

In the **M-step**, we recalculate the centroids according to the partition matrix, minimizing the SSE given in Eq. (11.4). The new centroid should be adjusted to

$$c_j = \frac{\sum_{\text{each point } o} w_{o,c_j}^2 o}{\sum_{\text{each point } o} w_{o,c_j}^2}, \tag{11.12}$$

where $j = 1, 2$.

In this example,

$$c_1 = \left(\frac{1^2 \times 3 + 0^2 \times 4 + 0.48^2 \times 9 + 0.42^2 \times 14 + 0.41^2 \times 18 + 0.47^2 \times 21}{1^2 + 0^2 + 0.48^2 + 0.42^2 + 0.41^2 + 0.47^2}, \right.$$
$$\left. \frac{1^2 \times 3 + 0^2 \times 10 + 0.48^2 \times 6 + 0.42^2 \times 8 + 0.41^2 \times 11 + 0.47^2 \times 7}{1^2 + 0^2 + 0.48^2 + 0.42^2 + 0.41^2 + 0.47^2} \right)$$
$$= (8.47, 5.12)$$

and

$$c_2 = \left(\frac{0^2 \times 3 + 1^2 \times 4 + 0.52^2 \times 9 + 0.58^2 \times 14 + 0.59^2 \times 18 + 0.53^2 \times 21}{0^2 + 1^2 + 0.52^2 + 0.58^2 + 0.59^2 + 0.53^2}, \right.$$
$$\left. \frac{0^2 \times 3 + 1^2 \times 10 + 0.52^2 \times 6 + 0.58^2 \times 8 + 0.59^2 \times 11 + 0.53^2 \times 7}{0^2 + 1^2 + 0.52^2 + 0.58^2 + 0.59^2 + 0.53^2} \right)$$
$$= (10.42, 8.99).$$

We repeat the iterations, where each iteration contains an E-step and an M-step. Table 11.3 shows the results from the first three iterations. The algorithm stops when the cluster centers converge or the change is small enough.

"*How can we apply the EM algorithm to compute probabilistic model-based clustering?*" Let's use a univariate Gaussian mixture model (Example 11.6) to illustrate.

Example 11.8 **Using the EM algorithm for mixture models.** Given a set of objects, $\mathbf{O} = \{o_1, \ldots, o_n\}$, we want to mine a set of parameters, $\Theta = \{\Theta_1, \ldots, \Theta_k\}$, such that $P(\mathbf{O}|\Theta)$ in Eq. (11.11) is maximized, where $\Theta_j = (\mu_j, \sigma_j)$ are the mean and standard deviation, respectively, of the jth univariate Gaussian distribution, $(1 \leq j \leq k)$.

We can apply the EM algorithm. We assign random values to parameters Θ as the initial values. We then iteratively conduct the E-step and the M-step as follows until the parameters converge or the change is sufficiently small.

In the **E-step**, for each object, $o_i \in \mathbf{O}$ $(1 \leq i \leq n)$, we calculate the probability that o_i belongs to each distribution, that is,

$$P(\Theta_j|o_i, \Theta) = \frac{P(o_i|\Theta_j)}{\sum_{l=1}^{k} P(o_i|\Theta_l)}. \tag{11.13}$$

In the **M-step**, we adjust the parameters Θ so that the expected likelihood $P(\mathbf{O}|\Theta)$ in Eq. (11.11) is maximized. This can be achieved by setting

$$\mu_j = \frac{1}{k} \sum_{i=1}^{n} o_i \frac{P(\Theta_j|o_i, \Theta)}{\sum_{l=1}^{n} P(\Theta_j|o_l, \Theta)} = \frac{1}{k} \frac{\sum_{i=1}^{n} o_i P(\Theta_j|o_i, \Theta)}{\sum_{i=1}^{n} P(\Theta_j|o_i, \Theta)} \tag{11.14}$$

and

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{n} P(\Theta_j|o_i,\Theta)(o_i - u_j)^2}{\sum_{i=1}^{n} P(\Theta_j|o_i,\Theta)}}. \tag{11.15}$$

■

In many applications, probabilistic model-based clustering has been shown to be effective because it is more general than partitioning methods and fuzzy clustering methods. A distinct advantage is that appropriate statistical models can be used to capture latent clusters. The EM algorithm is commonly used to handle many learning problems in data mining and statistics due to its simplicity. Note that, in general, the EM algorithm may not converge to the optimal solution. It may instead converge to a local maximum. Many heuristics have been explored to avoid this. For example, we could run the EM process multiple times using different random initial values. Furthermore, the EM algorithm can be very costly if the number of distributions is large or the data set contains very few observed data points.

11.2 Clustering High-Dimensional Data

The clustering methods we have studied so far work well when the dimensionality is not high, that is, having less than 10 attributes. There are, however, important applications of high dimensionality. "*How can we conduct cluster analysis on high-dimensional data?*"

In this section, we study approaches to clustering high-dimensional data. Section 11.2.1 starts with an overview of the major challenges and the approaches used. Methods for high-dimensional data clustering can be divided into two categories: subspace clustering methods (Section 11.2.2) and dimensionality reduction methods (Section 11.2.3).

11.2.1 Clustering High-Dimensional Data: Problems, Challenges, and Major Methodologies

Before we present any specific methods for clustering high-dimensional data, let's first demonstrate the needs of cluster analysis on high-dimensional data using examples. We examine the challenges that call for new methods. We then categorize the major methods according to whether they search for clusters in subspaces of the original space, or whether they create a new lower-dimensionality space and search for clusters there.

In some applications, a data object may be described by 10 or more attributes. Such objects are referred to as a high-dimensional data space.

Example 11.9 **High-dimensional data and clustering.** *AllElectronics* keeps track of the products purchased by every customer. As a customer-relationship manager, you want to cluster customers into groups according to what they purchased from *AllElectronics*.

Table 11.4 Customer Purchase Data

Customer	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
Ada	1	0	0	0	0	0	0	0	0	0
Bob	0	0	0	0	0	0	0	0	0	1
Cathy	1	0	0	0	1	0	0	0	0	1

The customer purchase data are of very high dimensionality. *AllElectronics* carries tens of thousands of products. Therefore, a customer's purchase profile, which is a vector of the products carried by the company, has tens of thousands of dimensions.

"Are the traditional distance measures, which are frequently used in low-dimensional cluster analysis, also effective on high-dimensional data?" Consider the customers in Table 11.4, where 10 products, $P_1, \ldots, P_{10}$, are used in demonstration. If a customer purchases a product, a 1 is set at the corresponding bit; otherwise, a 0 appears. Let's calculate the Euclidean distances (Eq. 2.16) among Ada, Bob, and Cathy. It is easy to see that

$$dist(\text{Ada}, \text{Bob}) = dist(\text{Bob}, \text{Cathy}) = dist(\text{Ada}, \text{Cathy}) = \sqrt{2}.$$

According to Euclidean distance, the three customers are equivalently similar (or dissimilar) to each other. However, a close look tells us that Ada should be more similar to Cathy than to Bob because Ada and Cathy share one common purchased item, P_1. ■

As shown in Example 11.9, the traditional distance measures can be ineffective on high-dimensional data. Such distance measures may be dominated by the noise in many dimensions. Therefore, clusters in the full, high-dimensional space can be unreliable, and finding such clusters may not be meaningful.

"*Then what kinds of clusters are meaningful on high-dimensional data?*" For cluster analysis of high-dimensional data, we still want to group similar objects together. However, the data space is often too big and too messy. An additional challenge is that we need to find not only clusters, but, for each cluster, a set of attributes that manifest the cluster. In other words, a cluster on high-dimensional data often is defined using a small set of attributes instead of the full data space. Essentially, clustering high-dimensional data should return groups of objects as clusters (as conventional cluster analysis does), *in addition to*, for each cluster, the set of attributes that characterize the cluster. For example, in Table 11.4, to characterize the similarity between Ada and Cathy, P_1 may be returned as the attribute because Ada and Cathy both purchased P_1.

Clustering high-dimensional data is the search for clusters and the space in which they exist. Thus, there are two major kinds of methods:

- *Subspace clustering approaches* search for clusters existing in subspaces of the given high-dimensional data space, where a subspace is defined using a subset of attributes in the full space. Subspace clustering approaches are discussed in Section 11.2.2.

- *Dimensionality reduction approaches* try to construct a much lower-dimensional space and search for clusters in such a space. Often, a method may construct new dimensions by combining some dimensions from the original data. Dimensionality reduction methods are the topic of Section 11.2.4.

In general, clustering high-dimensional data raises several new challenges in addition to those of conventional clustering:

- A major issue is how to create appropriate models for clusters in high-dimensional data. Unlike conventional clusters in low-dimensional spaces, clusters hidden in high-dimensional data are often significantly smaller. For example, when clustering customer-purchase data, we would not expect many users to have similar purchase patterns. Searching for such small but meaningful clusters is like finding needles in a haystack. As shown before, the conventional distance measures can be ineffective. Instead, we often have to consider various more sophisticated techniques that can model correlations and consistency among objects in subspaces.
- There are typically an exponential number of possible subspaces or dimensionality reduction options, and thus the optimal solutions are often computationally prohibitive. For example, if the original data space has 1000 dimensions, and we want to find clusters of dimensionality 10, then there are $\binom{1000}{10} = 2.63 \times 10^{23}$ possible subspaces.

11.2.2 Subspace Clustering Methods

"*How can we find subspace clusters from high-dimensional data?*" Many methods have been proposed. They generally can be categorized into three major groups: *subspace search methods*, *correlation-based clustering methods*, and *biclustering methods*.

Subspace Search Methods

A subspace search method searches various subspaces for clusters. Here, a cluster is a subset of objects that are similar to each other in a subspace. The similarity is often captured by conventional measures such as distance or density. For example, the CLIQUE algorithm introduced in Section 10.5.2 is a subspace clustering method. It enumerates subspaces and the clusters in those subspaces in a dimensionality-increasing order, and applies antimonotonicity to prune subspaces in which no cluster may exist.

A major challenge that subspace search methods face is how to search a series of subspaces effectively and efficiently. Generally there are two kinds of strategies:

- *Bottom-up approaches* start from low-dimensional subspaces and search higher-dimensional subspaces only when there may be clusters in those higher-dimensional

subspaces. Various pruning techniques are explored to reduce the number of higher-dimensional subspaces that need to be searched. CLIQUE is an example of a bottom-up approach.

- *Top-down approaches* start from the full space and search smaller and smaller subspaces recursively. Top-down approaches are effective only if the *locality assumption* holds, which require that the subspace of a cluster can be determined by the local neighborhood.

Example 11.10 **PROCLUS, a top-down subspace approach.** PROCLUS is a k-medoid-like method that first generates k potential cluster centers for a high-dimensional data set using a sample of the data set. It then refines the subspace clusters iteratively. In each iteration, for each of the current k-medoids, PROCLUS considers the local neighborhood of the medoid in the whole data set, and identifies a subspace for the cluster by minimizing the standard deviation of the distances of the points in the neighborhood to the medoid on each dimension. Once all the subspaces for the medoids are determined, each point in the data set is assigned to the closest medoid according to the corresponding subspace. Clusters and possible outliers are identified. In the next iteration, new medoids replace existing ones if doing so improves the clustering quality. ■

Correlation-Based Clustering Methods

While subspace search methods search for clusters with a similarity that is measured using conventional metrics like distance or density, *correlation-based approaches* can further discover clusters that are defined by advanced correlation models.

Example 11.11 **A correlation-based approach using PCA.** As an example, a *PCA-based approach* first applies PCA (Principal Components Analysis; see Chapter 3) to derive a set of new, uncorrelated dimensions, and then mine clusters in the new space or its subspaces. In addition to PCA, other space transformations may be used, such as the Hough transform or fractal dimensions. ■

For additional details on subspace search methods and correlation-based clustering methods, please refer to the bibliographic notes (Section 11.7).

Biclustering Methods

In some applications, we want to cluster both objects and attributes simultaneously. The resulting clusters are known as *biclusters* and meet four requirements: (1) only a small set of objects participate in a cluster; (2) a cluster only involves a small number of attributes; (3) an object may participate in multiple clusters, or does not participate in any cluster; and (4) an attribute may be involved in multiple clusters, or is not involved in any cluster. Section 11.2.3 discusses biclustering in detail.

11.2.3 Biclustering

In the cluster analysis discussed so far, we cluster objects according to their attribute values. Objects and attributes are not treated in the same way. However, in some applications, objects and attributes are defined in a symmetric way, where data analysis involves searching data matrices for submatrices that show unique patterns as clusters. This kind of clustering technique belongs to the category of biclustering.

This section first introduces two motivating application examples of biclustering—gene expression and recommender systems. You will then learn about the different types of biclusters. Last, we present biclustering methods.

Application Examples

Biclustering techniques were first proposed to address the needs for analyzing gene expression data. A *gene* is a unit of the passing-on of traits from a living organism to its offspring. Typically, a gene resides on a segment of DNA. Genes are critical for all living things because they specify all proteins and functional RNA chains. They hold the information to build and maintain a living organism's cells and pass genetic traits to offspring. Synthesis of a functional gene product, either RNA or protein, relies on the process of gene expression. A *genotype* is the genetic makeup of a cell, an organism, or an individual. *Phenotypes* are observable characteristics of an organism. *Gene expression* is the most fundamental level in genetics in that genotypes cause phenotypes.

Using *DNA chips* (also known as *DNA microarrays*) and other biological engineering techniques, we can measure the expression level of a large number (possibly all) of an organism's genes, in a number of different experimental conditions. Such conditions may correspond to different time points in an experiment or samples from different organs. Roughly speaking, the *gene expression data* or *DNA microarray data* are conceptually a gene-sample/condition matrix, where each row corresponds to one gene, and each column corresponds to one sample or condition. Each element in the matrix is a real number and records the expression level of a gene under a specific condition. Figure 11.3 shows an illustration.

From the clustering viewpoint, an interesting issue is that a gene expression data matrix can be analyzed in two dimensions—the gene dimension and the sample/condition dimension.

- When analyzing in the *gene dimension*, we treat each gene as an object and treat the samples/conditions as attributes. By mining in the gene dimension, we may find patterns shared by multiple genes, or cluster genes into groups. For example, we may find a group of genes that express themselves similarly, which is highly interesting in bioinformatics, such as in finding pathways.
- When analyzing in the *sample/condition dimension*, we treat each sample/condition as an object and treat the genes as attributes. In this way, we may find patterns of samples/conditions, or cluster samples/conditions into groups. For example, we may find the differences in gene expression by comparing a group of tumor samples and nontumor samples.

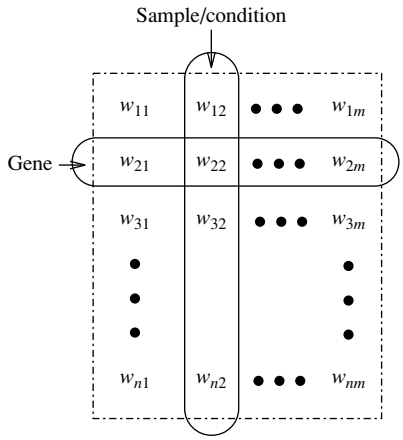

Figure 11.3 Microarrary data matrix.

Example 11.12 **Gene expression.** Gene expression matrices are popular in bioinformatics research and development. For example, an important task is to classify a new gene using the expression data of the gene and that of other genes in known classes. Symmetrically, we may classify a new sample (e.g., a new patient) using the expression data of the sample and that of samples in known classes (e.g., tumor and nontumor). Such tasks are invaluable in understanding the mechanisms of diseases and in clinical treatment. ■

As can be seen, many gene expression data mining problems are highly related to cluster analysis. However, a challenge here is that, instead of clustering in one dimension (e.g., gene or sample/condition), in many cases we need to cluster in two dimensions simultaneously (e.g., both gene and sample/condition). Moreover, unlike the clustering models we have discussed so far, a cluster in a gene expression data matrix is a *submatrix* and usually has the following characteristics:

- Only a small set of genes participate in the cluster.
- The cluster involves only a small subset of samples/conditions.
- A gene may participate in multiple clusters, or may not participate in any cluster.
- A sample/condition may be involved in multiple clusters, or may not be involved in any cluster.

To find clusters in gene-sample/condition matrices, we need new clustering techniques that meet the following requirements for *biclustering*:

- A cluster of genes is defined using only a subset of samples/conditions.
- A cluster of samples/conditions is defined using only a subset of genes.

- The clusters are neither *exclusive* (e.g., where one gene can participate in multiple clusters) nor *exhaustive* (e.g., where a gene may not participate in any cluster).

Biclustering is useful not only in bioinformatics, but also in other applications as well. Consider recommender systems as an example.

Example 11.13 Using biclustering for a recommender system. *AllElectronics* collects data from customers' evaluations of products and uses the data to recommend products to customers. The data can be modeled as a customer-product matrix, where each row represents a customer, and each column represents a product. Each element in the matrix represents a customer's evaluation of a product, which may be a score (e.g., like, like somewhat, not like) or purchase behavior (e.g., buy or not). Figure 11.4 illustrates the structure.

The customer-product matrix can be analyzed in two dimensions: the *customer* dimension and the *product* dimension. Treating each customer as an object and products as attributes, *AllElectronics* can find customer groups that have similar preferences or purchase patterns. Using products as objects and customers as attributes, *AllElectronics* can mine product groups that are similar in customer interest.

Moreover, *AllElectronics* can mine clusters in both customers and products simultaneously. Such a cluster contains a subset of customers and involves a subset of products. For example, *AllElectronics* is highly interested in finding a group of customers who all like the same group of products. Such a cluster is a submatrix in the customer-product matrix, where all elements have a high value. Using such a cluster, *AllElectronics* can make recommendations in two directions. First, the company can recommend products to new customers who are similar to the customers in the cluster. Second, the company can recommend to customers new products that are similar to those involved in the cluster. ■

As with biclusters in a gene expression data matrix, the biclusters in a customer-product matrix usually have the following characteristics:

- Only a small set of customers participate in a cluster.
- A cluster involves only a small subset of products.
- A customer can participate in multiple clusters, or may not participate in any cluster.

	Products			
Customers	w_{11}	w_{12}	$\cdots$	w_{1m}
	w_{21}	w_{22}	$\cdots$	w_{2m}
	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	w_{n1}	w_{n2}	$\cdots$	w_{nm}

Figure 11.4 Customer–product matrix.

- A product may be involved in multiple clusters, or may not be involved in any cluster.

Biclustering can be applied to customer-product matrices to mine clusters satisfying these requirements.

Types of Biclusters

"How can we model biclusters and mine them?" Let's start with some basic notation. For the sake of simplicity, we will use "genes" and "conditions" to refer to the two dimensions in our discussion. Our discussion can easily be extended to other applications. For example, we can simply replace "genes" and "conditions" by "customers" and "products" to tackle the customer-product biclustering problem.

Let $A = \{a_1, \ldots, a_n\}$ be a set of genes and $B = \{b_1, \ldots, b_m\}$ be a set of conditions. Let $E = [e_{ij}]$ be a gene expression data matrix, that is, a gene-condition matrix, where $1 \leq i \leq n$ and $1 \leq j \leq m$. A submatrix $I \times J$ is defined by a subset $I \subseteq A$ of genes and a subset $J \subseteq B$ of conditions. For example, in the matrix shown in Figure 11.5, $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$ is a submatrix.

A bicluster is a submatrix where genes and conditions follow consistent patterns. We can define different types of biclusters based on such patterns.

- As the simplest case, a submatrix $I \times J$ ($I \subseteq A, J \subseteq B$) is a **bicluster with constant values** if for any $i \in I$ and $j \in J$, $e_{ij} = c$, where c is a constant. For example, the submatrix $\{a_1, a_{33}, a_{86}\} \times \{b_6, b_{12}, b_{36}, b_{99}\}$ in Figure 11.5 is a bicluster with constant values.
- A bicluster is interesting if each row has a constant value, though different rows may have different values. A **bicluster with constant values on rows** is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \alpha_i$, where α_i is the adjustment for row i. For example, Figure 11.6 shows a bicluster with constant values on rows.

  Symmetrically, a **bicluster with constant values on columns** is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \beta_j$, where β_j is the adjustment for column j.

	$\cdots$	b_6	$\cdots$	b_{12}	$\cdots$	b_{36}	$\cdots$	b_{99} $\cdots$
a_1	$\cdots$	60	$\cdots$	60	$\cdots$	60	$\cdots$	60 $\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$ $\cdots$
a_{33}	$\cdots$	60	$\cdots$	60	$\cdots$	60	$\cdots$	60 $\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$ $\cdots$
a_{86}	$\cdots$	60	$\cdots$	60	$\cdots$	60	$\cdots$	60 $\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$ $\cdots$

Figure 11.5 Gene-condition matrix, a submatrix, and a bicluster.

- More generally, a bicluster is interesting if the rows change in a synchronized way with respect to the columns and vice versa. Mathematically, a **bicluster with coherent values** (also known as **a pattern-based cluster**) is a submatrix $I \times J$ such that for any $i \in I$ and $j \in J$, $e_{ij} = c + \alpha_i + \beta_j$, where α_i and β_j are the adjustment for row i and column j, respectively. For example, Figure 11.7 shows a bicluster with coherent values.

  It can be shown that $I \times J$ is a bicluster with coherent values if and only if for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, then $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$. Moreover, instead of using addition, we can define a bicluster with coherent values using multiplication, that is, $e_{ij} = c \cdot (\alpha_i \cdot \beta_j)$. Clearly, biclusters with constant values on rows or columns are special cases of biclusters with coherent values.

- In some applications, we may only be interested in the up- or down-regulated changes across genes or conditions without constraining the exact values. A **bicluster with coherent evolutions on rows** is a submatrix $I \times J$ such that for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, $(e_{i_1 j_1} - e_{i_1 j_2})(e_{i_2 j_1} - e_{i_2 j_2}) \geq 0$. For example, Figure 11.8 shows a bicluster with coherent evolutions on rows. Symmetrically, we can define biclusters with coherent evolutions on columns.

Next, we study how to mine biclusters.

10	10	10	10	10
20	20	20	20	20
50	50	50	50	50
0	0	0	0	0

Figure 11.6 Bicluster with constant values on rows.

10	50	30	70	20
20	60	40	80	30
50	90	70	110	60
0	40	20	60	10

Figure 11.7 Bicluster with coherent values.

10	50	30	70	20
20	100	50	1000	30
50	100	90	120	80
0	80	20	100	10

Figure 11.8 Bicluster with coherent evolutions on rows.

Biclustering Methods

The previous specification of the types of biclusters only considers ideal cases. In real data sets, such perfect biclusters rarely exist. When they do exist, they are usually very small. Instead, random noise can affect the readings of e_{ij} and thus prevent a bicluster in nature from appearing in a perfect shape.

There are two major types of methods for discovering biclusters in data that may come with noise. **Optimization-based methods** conduct an iterative search. At each iteration, the submatrix with the highest significance score is identified as a bicluster. The process terminates when a user-specified condition is met. Due to cost concerns in computation, greedy search is often employed to find local optimal biclusters. **Enumeration methods** use a tolerance threshold to specify the degree of noise allowed in the biclusters to be mined, and then tries to enumerate all submatrices of biclusters that satisfy the requirements. We use the δ-Cluster and MaPle algorithms as examples to illustrate these ideas.

Optimization Using the δ-Cluster Algorithm

For a submatrix, $I \times J$, the mean of the ith row is

$$e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{ij}. \tag{11.16}$$

Symmetrically, the mean of the jth column is

$$e_{Ij} = \frac{1}{|I|} \sum_{i \in I} e_{ij}. \tag{11.17}$$

The mean of all elements in the submatrix is

$$e_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} e_{ij} = \frac{1}{|I|} \sum_{i \in I} e_{iJ} = \frac{1}{|J|} \sum_{j \in J} e_{Ij}. \tag{11.18}$$

The quality of the submatrix as a bicluster can be measured by the *mean-squared residue* value as

$$H(I \times J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.19}$$

Submatrix $I \times J$ is a δ-**bicluster** if $H(I \times J) \leq \delta$, where $\delta \geq 0$ is a threshold. When $\delta = 0$, $I \times J$ is a perfect bicluster with coherent values. By setting $\delta > 0$, a user can specify the tolerance of average noise per element against a perfect bicluster, because in Eq. (11.19) the residue on each element is

$$\text{residue}(e_{ij}) = e_{ij} - e_{iJ} - e_{Ij} + e_{IJ}. \tag{11.20}$$

A *maximal δ-bicluster* is a δ-bicluster $I \times J$ such that there does not exist another δ-bicluster $I' \times J'$, and $I \subseteq I'$, $J \subseteq J'$, and at least one inequality holds. Finding the

maximal δ-bicluster of the largest size is computationally costly. Therefore, we can use a heuristic greedy search method to obtain a local optimal cluster. The algorithm works in two phases.

- In the *deletion phase*, we start from the whole matrix. While the mean-squared residue of the matrix is over δ, we iteratively remove rows and columns. At each iteration, for each row i, we compute the *mean-squared residue* as

$$d(i) = \frac{1}{|J|}\sum_{j\in J}(e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.21}$$

  Moreover, for each column j, we compute the *mean-squared residue* as

$$d(j) = \frac{1}{|I|}\sum_{i\in I}(e_{ij} - e_{iJ} - e_{Ij} + e_{IJ})^2. \tag{11.22}$$

  We remove the row or column of the largest mean-squared residue. At the end of this phase, we obtain a submatrix $I \times J$ that is a δ-bicluster. However, the submatrix may not be maximal.

- In the *addition phase*, we iteratively expand the δ-bicluster $I \times J$ obtained in the deletion phase as long as the δ-bicluster requirement is maintained. At each iteration, we consider rows and columns that are not involved in the current bicluster $I \times J$ by calculating their mean-squared residues. A row or column of the smallest mean-squared residue is added into the current δ-bicluster.

This greedy algorithm can find one δ-bicluster only. To find multiple biclusters that do not have heavy overlaps, we can run the algorithm multiple times. After each execution where a δ-bicluster is output, we can replace the elements in the output bicluster by random numbers. Although the greedy algorithm may find neither the optimal biclusters nor all biclusters, it is very fast even on large matrices.

Enumerating All Biclusters Using MaPle

As mentioned, a submatrix $I \times J$ is a bicluster with coherent values if and only if for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$. For any 2×2 submatrix of $I \times J$, we can define a *p-score* as

$$p\text{-score}\begin{pmatrix} e_{i_1 j_1} & e_{i_1 j_2} \\ e_{i_2 j_1} & e_{i_2 j_2} \end{pmatrix} = |(e_{i_1 j_1} - e_{i_2 j_1}) - (e_{i_1 j_2} - e_{i_2 j_2})|. \tag{11.23}$$

A submatrix $I \times J$ is a δ-**pCluster** (for *p*attern-based *cluster*) if the p-score of every 2×2 submatrix of $I \times J$ is at most δ, where $\delta \geq 0$ is a threshold specifying a user's tolerance of noise against a perfect bicluster. Here, the p-score controls the noise on every element in a bicluster, while the mean-squared residue captures the average noise.

An interesting property of δ-pCluster is that if $I \times J$ is a δ-pCluster, then every $x \times y$ $(x, y \geq 2)$ submatrix of $I \times J$ is also a δ-pCluster. This monotonicity enables

us to obtain a succinct representation of nonredundant δ-pClusters. A δ-pCluster is maximal if no more rows or columns can be added into the cluster while maintaining the δ-pCluster property. To avoid redundancy, instead of finding all δ-pClusters, we only need to compute all maximal δ-pClusters.

MaPle is an algorithm that enumerates all maximal δ-pClusters. It systematically enumerates every combination of conditions using a set enumeration tree and a depth-first search. This enumeration framework is the same as the pattern-growth methods for frequent pattern mining (Chapter 6). Consider gene expression data. For each condition combination, J, MaPle finds the maximal subsets of genes, I, such that $I \times J$ is a δ-pCluster. If $I \times J$ is not a submatrix of another δ-pCluster, then $I \times J$ is a maximal δ-pCluster.

There may be a huge number of condition combinations. MaPle prunes many unfruitful combinations using the monotonicity of δ-pClusters. For a condition combination, J, if there does not exist a set of genes, I, such that $I \times J$ is a δ-pCluster, then we do not need to consider any superset of J. Moreover, we should consider $I \times J$ as a candidate of a δ-pCluster only if for every $(|J| - 1)$-subset J' of J, $I \times J'$ is a δ-pCluster. MaPle also employs several pruning techniques to speed up the search while retaining the completeness of returning all maximal δ-pClusters. For example, when examining a current δ-pCluster, $I \times J$, MaPle collects all the genes and conditions that may be added to expand the cluster. If these candidate genes and conditions together with I and J form a submatrix of a δ-pCluster that has already been found, then the search of $I \times J$ and any superset of J can be pruned. Interested readers may refer to the bibliographic notes for additional information on the MaPle algorithm (Section 11.7).

An interesting observation here is that the search for maximal δ-pClusters in MaPle is somewhat similar to mining frequent closed itemsets. Consequently, MaPle borrows the depth-first search framework and ideas from the pruning techniques of pattern-growth methods for frequent pattern mining. This is an example where frequent pattern mining and cluster analysis may share similar techniques and ideas.

An advantage of MaPle and the other algorithms that enumerate all biclusters is that they guarantee the completeness of the results and do not miss any overlapping biclusters. However, a challenge for such enumeration algorithms is that they may become very time consuming if a matrix becomes very large, such as a customer-purchase matrix of hundreds of thousands of customers and millions of products.

11.2.4 Dimensionality Reduction Methods and Spectral Clustering

Subspace clustering methods try to find clusters in subspaces of the original data space. In some situations, it is more effective to construct a new space instead of using subspaces of the original data. This is the motivation behind dimensionality reduction methods for clustering high-dimensional data.

Example 11.14 Clustering in a derived space. Consider the three clusters of points in Figure 11.9. It is not possible to cluster these points in any subspace of the original space, $X \times Y$, because

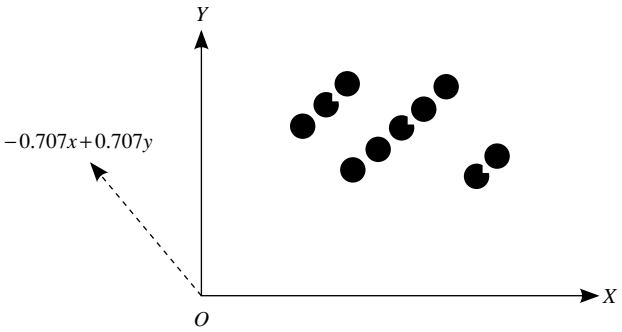

Figure 11.9 Clustering in a derived space may be more effective.

all three clusters would end up being projected onto overlapping areas in the x and y axes. What if, instead, we construct a new dimension, $-\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y$ (shown as a dashed line in the figure)? By projecting the points onto this new dimension, the three clusters become apparent. ■

Although Example 11.14 involves only two dimensions, the idea of constructing a new space (so that any clustering structure that is hidden in the data becomes well manifested) can be extended to high-dimensional data. Preferably, the newly constructed space should have low dimensionality.

There are many dimensionality reduction methods. A straightforward approach is to apply feature selection and extraction methods to the data set such as those discussed in Chapter 3. However, such methods may not be able to detect the clustering structure. Therefore, methods that combine feature extraction and clustering are preferred. In this section, we introduce *spectral clustering*, a group of methods that are effective in high-dimensional data applications.

Figure 11.10 shows the general framework for spectral clustering approaches. The Ng-Jordan-Weiss algorithm is a spectral clustering method. Let's have a look at each step of the framework. In doing so, we also note special conditions that apply to the Ng-Jordan-Weiss algorithm as an example.

Given a set of objects, $o_1, \ldots, o_n$, the distance between each pair of objects, $dist(o_i, o_j)$ $(1 \leq i, j \leq n)$, and the desired number k of clusters, a spectral clustering approach works as follows.

1. Using the distance measure, calculate an *affinity matrix*, W, such that

$$W_{ij} = e^{-\frac{dist(o_i,o_j)}{\sigma^2}},$$

where σ is a scaling parameter that controls how fast the affinity W_{ij} decreases as $dist(o_i, o_j)$ increases. In the Ng-Jordan-Weiss algorithm, W_{ii} is set to 0.

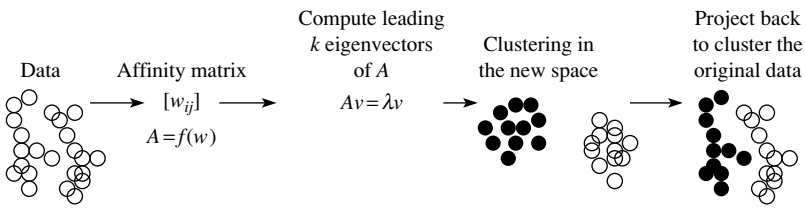

Figure 11.10 The framework of spectral clustering approaches. *Source:* Adapted from Slide 8 at *http://videolectures.net/micued08_azran_mcl/*.

2. Using the affinity matrix W, derive a matrix $A = f(W)$. The way in which this is done can vary. The Ng-Jordan-Weiss algorithm defines a matrix, D, as a diagonal matrix such that D_{ii} is the sum of the ith row of W, that is,

$$D_{ii} = \sum_{j=1}^{n} W_{ij}. \tag{11.24}$$

A is then set to

$$A = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}. \tag{11.25}$$

3. Find the k leading eigenvectors of A. Recall that the *eigenvectors* of a square matrix are the nonzero vectors that remain proportional to the original vector after being multiplied by the matrix. Mathematically, a vector $\boldsymbol{v}$ is an eigenvector of matrix A if $A\boldsymbol{v} = \lambda \boldsymbol{v}$, where λ is called the corresponding *eigenvalue*. This step derives k new dimensions from A, which are based on the affinity matrix W. Typically, k should be much smaller than the dimensionality of the original data.

The Ng-Jordan-Weiss algorithm computes the k eigenvectors with the largest eigenvalues $x_1, \ldots, x_k$ of A.

4. Using the k leading eigenvectors, project the original data into the new space defined by the k leading eigenvectors, and run a clustering algorithm such as k-means to find k clusters.

The Ng-Jordan-Weiss algorithm stacks the k largest eigenvectors in columns to form a matrix $X = [x_1 x_2 \cdots x_k] \in \mathbb{R}^{n \times k}$. The algorithm forms a matrix Y by renormalizing each row in X to have unit length, that is,

$$Y_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{k} X_{ij}^2}}. \tag{11.26}$$

The algorithm then treats each row in Y as a point in the k-dimensional space $\mathbb{R}^k$, and runs k-means (or any other algorithm serving the partitioning purpose) to cluster the points into k clusters.

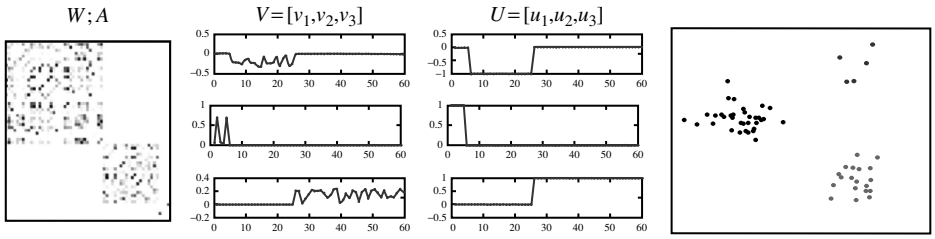

Figure 11.11 The new dimensions and the clustering results of the Ng-Jordan-Weiss algorithm. *Source:* Adapted from Slide 9 at *http://videolectures.net/micued08_azran_mcl/*.

5. Assign the original data points to clusters according to how the transformed points are assigned in the clusters obtained in step 4.

In the Ng-Jordan-Weiss algorithm, the original object o_i is assigned to the jth cluster if and only if matrix Y's row i is assigned to the jth cluster as a result of step 4.

In spectral clustering methods, the dimensionality of the new space is set to the desired number of clusters. This setting expects that each new dimension should be able to manifest a cluster.

Example 11.15 The Ng-Jordan-Weiss algorithm. Consider the set of points in Figure 11.11. The data set, the affinity matrix, the three largest eigenvectors, and the normalized vectors are shown. Note that with the three new dimensions (formed by the three largest eigenvectors), the clusters are easily detected. ■

Spectral clustering is effective in high-dimensional applications such as image processing. Theoretically, it works well when certain conditions apply. Scalability, however, is a challenge. Computing eigenvectors on a large matrix is costly. Spectral clustering can be combined with other clustering methods, such as biclustering. Additional information on other dimensionality reduction clustering methods, such as kernel PCA, can be found in the bibliographic notes (Section 11.7).

11.3 Clustering Graph and Network Data

Cluster analysis on graph and network data extracts valuable knowledge and information. Such data are increasingly popular in many applications. We discuss applications and challenges of clustering graph and network data in Section 11.3.1. Similarity measures for this form of clustering are given in Section 11.3.2. You will learn about graph clustering methods in Section 11.3.3.

In general, the terms *graph* and *network* can be used interchangeably. In the rest of this section, we mainly use the term *graph*.

11.3.1 Applications and Challenges

As a customer relationship manager at *AllElectronics*, you notice that a lot of data relating to customers and their purchase behavior can be preferably modeled using graphs.

Example 11.16 **Bipartite graph.** The customer purchase behavior at *AllElectronics* can be represented in a *bipartite graph*. In a bipartite graph, vertices can be divided into two disjoint sets so that each edge connects a vertex in one set to a vertex in the other set. For the *AllElectronics* customer purchase data, one set of vertices represents customers, with one customer per vertex. The other set represents products, with one product per vertex. An edge connects a customer to a product, representing the purchase of the product by the customer. Figure 11.12 shows an illustration.

"*What kind of knowledge can we obtain by a cluster analysis of the customer-product bipartite graph?*" By clustering the customers such that those customers buying similar sets of products are placed into one group, a customer relationship manager can make product recommendations. For example, suppose Ada belongs to a customer cluster in which most of the customers purchased a digital camera in the last 12 months, but Ada has yet to purchase one. As manager, you decide to recommend a digital camera to her.

Alternatively, we can cluster products such that those products purchased by similar sets of customers are grouped together. This clustering information can also be used for product recommendations. For example, if a digital camera and a high-speed flash memory card belong to the same product cluster, then when a customer purchases a digital camera, we can recommend the high-speed flash memory card. ■

Bipartite graphs are widely used in many applications. Consider another example.

Example 11.17 **Web search engines.** In web search engines, search logs are archived to record user queries and the corresponding *click-through information*. (The click-through information tells us on which pages, given as a result of a search, the user clicked.) The query and click-through information can be represented using a bipartite graph, where the two sets

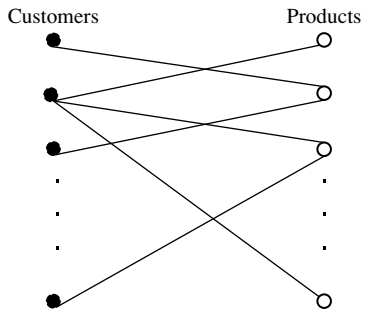

Figure 11.12 Bipartite graph representing customer-purchase data.

of vertices correspond to queries and web pages, respectively. An edge links a query to a web page if a user clicks the web page when asking the query. Valuable information can be obtained by cluster analyses on the query–web page bipartite graph. For instance, we may identify queries posed in different languages, but that mean the same thing, if the click-through information for each query is similar.

As another example, all the web pages on the Web form a directed graph, also known as the *web graph*, where each web page is a vertex, and each hyperlink is an edge pointing from a source page to a destination page. Cluster analysis on the web graph can disclose communities, find hubs and authoritative web pages, and detect web spams. ■

In addition to bipartite graphs, cluster analysis can also be applied to other types of graphs, including general graphs, as elaborated Example 11.18.

Example 11.18 **Social network.** A *social network* is a social structure. It can be represented as a graph, where the vertices are individuals or organizations, and the links are interdependencies between the vertices, representing friendship, common interests, or collaborative activities. *AllElectronics*' customers form a social network, where each customer is a vertex, and an edge links two customers if they know each other.

As customer relationship manager, you are interested in finding useful information that can be derived from *AllElectronics*' social network through cluster analysis. You obtain clusters from the network, where customers in a cluster know each other or have friends in common. Customers within a cluster may influence one another regarding purchase decision making. Moreover, communication channels can be designed to inform the "heads" of clusters (i.e., the "best" connected people in the clusters), so that promotional information can be spread out quickly. Thus, you may use customer clustering to promote sales at *AllElectronics*.

As another example, the authors of scientific publications form a social network, where the authors are vertices and two authors are connected by an edge if they coauthored a publication. The network is, in general, a weighted graph because an edge between two authors can carry a weight representing the strength of the collaboration such as how many publications the two authors (as the end vertices) coauthored. Clustering the coauthor network provides insight as to communities of authors and patterns of collaboration. ■

"Are there any challenges specific to cluster analysis on graph and network data?" In most of the clustering methods discussed so far, objects are represented using a set of attributes. A unique feature of graph and network data is that only objects (as vertices) and relationships between them (as edges) are given. No dimensions or attributes are explicitly defined. To conduct cluster analysis on graph and network data, there are two major new challenges.

- *"How can we measure the similarity between two objects on a graph accordingly?"* Typically, we cannot use conventional distance measures, such as Euclidean distance. Instead, we need to develop new measures to quantify the similarity. Such

measures often are not metric, and thus raise new challenges regarding the development of efficient clustering methods. Similarity measures for graphs are discussed in Section 11.3.2.

- *"How can we design clustering models and methods that are effective on graph and network data?"* Graph and network data are often complicated, carrying topological structures that are more sophisticated than traditional cluster analysis applications. Many graph data sets are large, such as the web graph containing at least tens of billions of web pages in the publicly indexable Web. Graphs can also be sparse where, on average, a vertex is connected to only a small number of other vertices in the graph. To discover accurate and useful knowledge hidden deep in the data, a good clustering method has to accommodate these factors. Clustering methods for graph and network data are introduced in Section 11.3.3.

11.3.2 Similarity Measures

"How can we measure the similarity or distance between two vertices in a graph?" In our discussion, we examine two types of measures: *geodesic distance* and *distance based on random walk*.

Geodesic Distance

A simple measure of the distance between two vertices in a graph is the shortest path between the vertices. Formally, the **geodesic distance** between two vertices is the length in terms of the number of edges of the shortest path between the vertices. For two vertices that are not connected in a graph, the geodesic distance is defined as infinite.

Using geodesic distance, we can define several other useful measurements for graph analysis and clustering. Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, we define the following:

- For a vertext $v \in V$, the **eccentricity** of v, denoted $eccen(v)$, is the largest geodesic distance between v and any other vertex $u \in V - \{v\}$. The eccentricity of v captures how far away v is from its remotest vertex in the graph.
- The **radius** of graph G is the minimum eccentricity of all vertices. That is,

$$r = \min_{v \in V} eccen(v). \tag{11.27}$$

  The radius captures the distance between the "most central point" and the "farthest border" of the graph.
- The **diameter** of graph G is the maximum eccentricity of all vertices. That is,

$$d = \max_{v \in V} eccen(v). \tag{11.28}$$

  The diameter represents the largest distance between any pair of vertices.
- A **peripheral vertex** is a vertex that achieves the diameter.

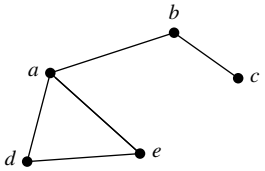

Figure 11.13 A graph, G, where vertices c, d, and e are peripheral.

Example 11.19 **Measurements based on geodesic distance.** Consider graph G in Figure 11.13. The eccentricity of a is 2, that is, $eccen(a) = 2$, $eccen(b) = 2$, and $eccen(c) = eccen(d) = eccen(e) = 3$. Thus, the radius of G is 2, and the diameter is 3. Note that it is not necessary that $d = 2 \times r$. Vertices c, d, and e are peripheral vertices. ■

SimRank: Similarity Based on Random Walk and Structural Context

For some applications, geodesic distance may be inappropriate in measuring the similarity between vertices in a graph. Here we introduce SimRank, a similarity measure based on random walk and on the structural context of the graph. In mathematics, a *random walk* is a trajectory that consists of taking successive random steps.

Example 11.20 **Similarity between people in a social network.** Let's consider measuring the similarity between two vertices in the *AllElectronics* customer social network of Example 11.18. Here, similarity can be explained as the closeness between two participants in the network, that is, how close two people are in terms of the relationship represented by the social network.

"*How well can the geodesic distance measure similarity and closeness in such a network?*" Suppose Ada and Bob are two customers in the network, and the network is undirected. The geodesic distance (i.e., the length of the shortest path between Ada and Bob) is the shortest path that a message can be passed from Ada to Bob and vice versa. However, this information is not useful for *AllElectronics*' customer relationship management because the company typically does not want to send a specific message from one customer to another. Therefore, geodesic distance does not suit the application.

"*What does similarity mean in a social network?*" We consider two ways to define similarity:

- Two customers are considered similar to one another if they have similar neighbors in the social network. This heuristic is intuitive because, in practice, two people receiving recommendations from a good number of common friends often make similar decisions. This kind of similarity is based on the local structure (i.e., the *neighborhoods*) of the vertices, and thus is called *structural context–based similarity*.

- Suppose *AllElectronics* sends promotional information to both Ada and Bob in the social network. Ada and Bob may randomly forward such information to their friends (or *neighbors*) in the network. The closeness between Ada and Bob can then be measured by the likelihood that other customers simultaneously receive the promotional information that was originally sent to Ada and Bob. This kind of similarity is based on the random walk reachability over the network, and thus is referred to as *similarity based on random walk*. ■

Let's have a closer look at what is meant by similarity based on structural context, and similarity based on random walk.

The intuition behind similarity based on structural context is that two vertices in a graph are similar if they are connected to similar vertices. To measure such similarity, we need to define the notion of individual neighborhood. In a directed graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, for a vertex $v \in V$, the *individual in-neighborhood* of v is defined as

$$I(v) = \{u | (u, v) \in E\}. \tag{11.29}$$

Symmetrically, we define the *individual out-neighborhood* of v as

$$O(v) = \{w | (v, w) \in E\}. \tag{11.30}$$

Following the intuition illustrated in Example 11.20, we define SimRank, a structural-context similarity, with a value that is between 0 and 1 for any pair of vertices. For any vertex, $v \in V$, the similarity between the vertex and itself is $s(v, v) = 1$ because the neighborhoods are identical. For vertices $u, v \in V$ such that $u \neq v$, we can define

$$s(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s(x, y), \tag{11.31}$$

where C is a constant between 0 and 1. A vertex may not have any in-neighbors. Thus, we define Eq. (11.31) to be 0 when either $I(u)$ or $I(v)$ is $\emptyset$. Parameter C specifies the rate of decay as similarity is propagated across edges.

"*How can we compute SimRank?*" A straightforward method iteratively evaluates Eq. (11.31) until a fixed point is reached. Let $s_i(u, v)$ be the SimRank score calculated at the ith round. To begin, we set

$$s_0(u, v) = \begin{cases} 0 & \text{if } u \neq v \\ 1 & \text{if } u = v. \end{cases} \tag{11.32}$$

We use Eq. (11.31) to compute s_{i+1} from s_i as

$$s_{i+1}(u, v) = \frac{C}{|I(u)||I(v)|} \sum_{x \in I(u)} \sum_{y \in I(v)} s_i(x, y). \tag{11.33}$$

It can be shown that $\lim_{i \to \infty} s_i(u, v) = s(u, v)$. Additional methods for approximating SimRank are given in the bibliographic notes (Section 11.7).

Now, let's consider similarity based on random walk. A directed graph is *strongly connected* if, for any two nodes u and v, there is a path from u to v and another path from v to u. In a strongly connected graph, $G = (V, E)$, for any two vertices, $u, v \in V$, we can define the *expected distance* from u to v as

$$d(u, v) = \sum_{t: u \rightsquigarrow v} P[t] l(t), \tag{11.34}$$

where $u \rightsquigarrow v$ is a path starting from u and ending at v that may contain cycles but does not reach v until the end. For a *traveling tour*, $t = w_1 \to w_2 \to \cdots \to w_k$, its length is $l(t) = k - 1$. The probability of the tour is defined as

$$P[t] = \begin{cases} \prod_{i=1}^{k-1} \frac{1}{|O(w_i)|} & \text{if } l(t) > 0 \\ 0 & \text{if } l(t) = 0. \end{cases} \tag{11.35}$$

To measure the probability that a vertex w receives a message that originated simultaneously from u and v, we extend the expected distance to the notion of *expected meeting distance*, that is,

$$m(u, v) = \sum_{t: (u,v) \rightsquigarrow (x,x)} P[t] l(t), \tag{11.36}$$

where $(u, v) \rightsquigarrow (x, x)$ is a pair of tours $u \rightsquigarrow x$ and $v \rightsquigarrow x$ of the same length. Using a constant C between 0 and 1, we define the *expected meeting probability* as

$$p(u, v) = \sum_{t: (u,v) \rightsquigarrow (x,x)} P[t] C^{l(t)}, \tag{11.37}$$

which is a similarity measure based on random walk. Here, the parameter C specifies the probability of continuing the walk at each step of the trajectory.

It has been shown that $s(u, v) = p(u, v)$ for any two vertices, u and v. That is, SimRank is based on both structural context and random walk.

11.3.3 Graph Clustering Methods

Let's consider how to conduct clustering on a graph. We first describe the intuition behind graph clustering. We then discuss two general categories of graph clustering methods.

To find clusters in a graph, imagine cutting the graph into pieces, each piece being a cluster, such that the vertices within a cluster are well connected and the vertices in different clusters are connected in a much weaker way. Formally, for a graph, $G = (V, E)$,

a **cut**, $C = (S, T)$, is a partitioning of the set of vertices V in G, that is, $V = S \cup T$ and $S \cap T = \emptyset$. The *cut set* of a cut is the set of edges, $\{(u, v) \in E | u \in S, v \in T\}$. The *size* of the cut is the number of edges in the cut set. For weighted graphs, the size of a cut is the sum of the weights of the edges in the cut set.

"What kinds of cuts are good for deriving clusters in graphs?" In graph theory and some network applications, a minimum cut is of importance. A cut is *minimum* if the cut's size is not greater than any other cut's size. There are polynomial time algorithms to compute minimum cuts of graphs. Can we use these algorithms in graph clustering?

Example 11.21 **Cuts and clusters.** Consider graph G in Figure 11.14. The graph has two clusters: $\{a, b, c, d, e, f\}$ and $\{g, h, i, j, k\}$, and one outlier vertex, l.

Consider cut $C_1 = (\{a, b, c, d, e, f, g, h, i, j, k\}, \{l\})$. Only one edge, namely, (e, l), crosses the two partitions created by C_1. Therefore, the cut set of C_1 is $\{(e, l)\}$ and the size of C_1 is 1. (Note that the size of any cut in a connected graph cannot be smaller than 1.) As a minimum cut, C_1 does not lead to a good clustering because it only separates the outlier vertex, l, from the rest of the graph.

Cut $C_2 = (\{a, b, c, d, e, f, l\}, \{g, h, i, j, k\})$ leads to a much better clustering than C_1. The edges in the cut set of C_2 are those connecting the two "natural clusters" in the graph. Specifically, for edges (d, h) and (e, k) that are in the cut set, most of the edges connecting d, h, e, and k belong to one cluster. ■

Example 11.21 indicates that using a minimum cut is unlikely to lead to a good clustering. We are better off choosing a cut where, for each vertex u that is involved in an edge in the cut set, most of the edges connecting to u belong to one cluster. Formally, let $deg(u)$ be the degree of u, that is, the number of edges connecting to u. The *sparsity* of a cut $C = (S, T)$ is defined as

$$\Phi = \frac{\text{cut size}}{\min\{|S|, |T|\}}. \tag{11.38}$$

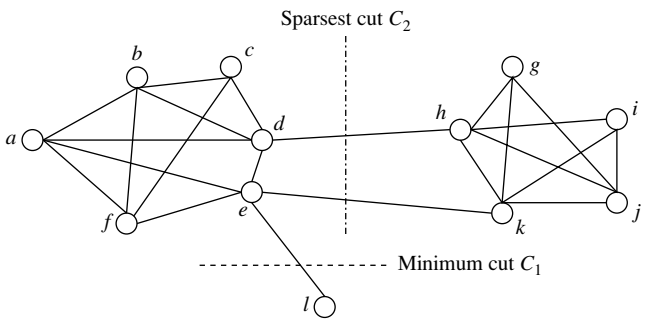

Figure 11.14 A graph G and two cuts.

A cut is *sparsest* if its sparsity is not greater than the sparsity of any other cut. There may be more than one sparsest cut.

In Example 11.21 and Figure 11.14, C_2 is a sparsest cut. Using sparsity as the objective function, a sparsest cut tries to minimize the number of edges crossing the partitions and balance the partitions in size.

Consider a clustering on a graph $G = (V, E)$ that partitions the graph into k clusters. The **modularity** of a clustering assesses the quality of the clustering and is defined as

$$Q = \sum_{i=1}^{k} \left(\frac{l_i}{|E|} - \left(\frac{d_i}{2|E|} \right)^2 \right), \tag{11.39}$$

where l_i is the number of edges between vertices in the ith cluster, and d_i is the sum of the degrees of the vertices in the ith cluster. The modularity of a clustering of a graph is the difference between the fraction of all edges that fall into individual clusters and the fraction that would do so if the graph vertices were randomly connected. The optimal clustering of graphs maximizes the modularity.

Theoretically, many graph clustering problems can be regarded as finding good cuts, such as the sparsest cuts, on the graph. In practice, however, a number of challenges exist:

- **High computational cost:** Many graph cut problems are computationally expensive. The sparsest cut problem, for example, is NP-hard. Therefore, finding the optimal solutions on large graphs is often impossible. A good trade-off between efficiency/scalability and quality has to be achieved.
- **Sophisticated graphs:** Graphs can be more sophisticated than the ones described here, involving weights and/or cycles.
- **High dimensionality:** A graph can have many vertices. In a similarity matrix, a vertex is represented as a vector (a row in the matrix) with a dimensionality that is the number of vertices in the graph. Therefore, graph clustering methods must handle high dimensionality.
- **Sparsity:** A large graph is often sparse, meaning each vertex on average connects to only a small number of other vertices. A similarity matrix from a large sparse graph can also be sparse.

There are two kinds of methods for clustering graph data, which address these challenges. One uses clustering methods for high-dimensional data, while the other is designed specifically for clustering graphs.

The first group of methods is based on generic clustering methods for high-dimensional data. They extract a similarity matrix from a graph using a similarity measure such as those discussed in Section 11.3.2. A generic clustering method can then be applied on the similarity matrix to discover clusters. Clustering methods for

high-dimensional data are typically employed. For example, in many scenarios, once a similarity matrix is obtained, spectral clustering methods (Section 11.2.4) can be applied. Spectral clustering can approximate optimal graph cut solutions. For additional information, please refer to the bibliographic notes (Section 11.7).

The second group of methods is specific to graphs. They search the graph to find well-connected components as clusters. Let's look at a method called **SCAN** (Structural Clustering Algorithm for Networks) as an example.

Given an undirected graph, $G = (V, E)$, for a vertex, $u \in V$, the neighborhood of u is $\Gamma(u) = \{v | (u, v) \in E\} \cup \{u\}$. Using the idea of structural-context similarity, SCAN measures the similarity between two vertices, $u, v \in V$, by the normalized common neighborhood size, that is,

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)||\Gamma(v)|}}. \tag{11.40}$$

The larger the value computed, the more similar the two vertices. SCAN uses a similarity threshold ε to define the cluster membership. For a vertex, $u \in V$, the *ε-neighborhood* of u is defined as $N_\varepsilon(u) = \{v \in \Gamma(u) | \sigma(u, v) \geq \varepsilon\}$. The ε-neighborhood of u contains all neighbors of u with a structural-context similarity to u that is at least ε.

In SCAN, a *core vertex* is a vertex inside of a cluster. That is, $u \in V$ is a core vertex if $|N_\varepsilon(u)| \geq \mu$, where μ is a popularity threshold. SCAN grows clusters from core vertices. If a vertex v is in the ε-neighborhood of a core u, then v is assigned to the same cluster as u. This process of growing clusters continues until no cluster can be further grown. The process is similar to the density-based clustering method, DBSCAN (Chapter 10).

Formally, a vertex v can be *directly reached* from a core u if $v \in N_\varepsilon(u)$. Transitively, a vertex v can be *reached* from a core u if there exist vertices $w_1, \ldots, w_n$ such that w_1 can be reached from u, w_i can be reached from w_{i-1} for $1 < i \leq n$, and v can be reached from w_n. Moreover, two vertices, $u, v \in V$, which may or may not be cores, are said to be *connected* if there exists a core w such that both u and v can be reached from w. All vertices in a cluster are connected. A cluster is a maximum set of vertices such that every pair in the set is connected.

Some vertices may not belong to any cluster. Such a vertex u is a *hub* if the neighborhood $\Gamma(u)$ of u contains vertices from more than one cluster. If a vertex does not belong to any cluster, and is not a hub, it is an *outlier*.

The SCAN algorithm is shown in Figure 11.15. The search framework closely resembles the cluster-finding process in DBSCAN. SCAN finds a cut of the graph, where each cluster is a set of vertices that are connected based on the transitive similarity in a structural context.

An advantage of SCAN is that its time complexity is linear with respect to the number of edges. In very large and sparse graphs, the number of edges is in the same scale of the number of vertices. Therefore, SCAN is expected to have good scalability on clustering large graphs.

```
Algorithm: SCAN for clusters on graph data.
Input: a graph G = (V, E), a similarity threshold ε, and a
  population threshold μ
Output: a set of clusters
Method: set all vertices in V unlabeled
  for all unlabeled vertex u do
    if u is a core then
      generate a new cluster-id c
      insert all v ∈ N_ε(u) into a queue Q
      while Q ≠ do
        w ← the first vertex in Q
        R ← the set of vertices that can be directly reached from w
        for all s ∈ R do
          if s is not unlabeled or labeled as nonmember then
            assign the current cluster-id c to s
          endif
          if s is unlabeled then
            insert s into queue Q
          endif
        endfor
        remove w from Q
      end while
    else
      label u as nonmember
    endif
  endfor
  for all vertex u labeled nonmember do
    if ∃x, y ∈ Γ(u) : x and y have different cluster-ids then
      label u as hub
    else
      label u as outlier
    endif
  endfor
```

Figure 11.15 SCAN algorithm for cluster analysis on graph data.

11.4 Clustering with Constraints

Users often have background knowledge that they want to integrate into cluster analysis. There may also be application-specific requirements. Such information can be modeled as clustering constraints. We approach the topic of clustering with constraints in two steps. Section 11.4.1 categorizes the types of constraints for clustering graph data. Methods for clustering with constraints are introduced in Section 11.4.2.

11.4.1 Categorization of Constraints

This section studies how to categorize the constraints used in cluster analysis. Specifically, we can categorize constraints according to the subjects on which they are set, or on how strongly the constraints are to be enforced.

As discussed in Chapter 10, cluster analysis involves three essential aspects: objects as instances of clusters, clusters as groups of objects, and the similarity among objects. Therefore, the first method we discuss categorizes constraints according to what they are applied to. We thus have three types: *constraints on instances*, *constraints on clusters*, and *constraints on similarity measurement*.

Constraints on instances: A *constraint on instances* specifies how a pair or a set of instances should be grouped in the cluster analysis. Two common types of constraints from this category include:

- **Must-link constraints.** If a must-link constraint is specified on two objects x and y, then x and y should be grouped into one cluster in the output of the cluster analysis. These must-link constraints are transitive. That is, if must-link(x, y) and must-link(y, z), then must-link(x, z).
- **Cannot-link constraints.** Cannot-link constraints are the opposite of must-link constraints. If a cannot-link constraint is specified on two objects, x and y, then in the output of the cluster analysis, x and y should belong to different clusters. Cannot-link constraints can be entailed. That is, if cannot-link(x, y), must-link(x, x'), and must-link(y, y'), then cannot-link(x', y').

A constraint on instances can be defined using specific instances. Alternatively, it can also be defined using instance variables or attributes of instances. For example, a constraint,

$$Constraint(x, y) : \text{must-link}(x, y) \text{ if } dist(x, y) \leq \epsilon,$$

uses the distance between objects to specify a must-link constraint.

Constraints on clusters: A *constraint on clusters* specifies a requirement on the clusters, possibly using attributes of the clusters. For example, a constraint may specify the minimum number of objects in a cluster, the maximum diameter of a cluster, or the shape of a cluster (e.g., a convex). The number of clusters specified for partitioning clustering methods can be regarded as a constraint on clusters.

Constraints on similarity measurement: Often, a similarity measure, such as Euclidean distance, is used to measure the similarity between objects in a cluster analysis. In some applications, exceptions apply. A *constraint on similarity measurement* specifies a requirement that the similarity calculation must respect. For example, to cluster people as moving objects in a plaza, while Euclidean distance is used to give

the walking distance between two points, a constraint on similarity measurement is that the trajectory implementing the shortest distance cannot cross a wall.

There can be more than one way to express a constraint, depending on the category. For example, we can specify a constraint on clusters as

$$Constraint_1\text{: the diameter of a cluster cannot be larger than } d.$$

The requirement can also be expressed using a constraint on instances as

$$Constraint_1'\text{: cannot-link}(x, y) \text{ if } dist(x, y) > d. \tag{11.41}$$

Example 11.22 **Constraints on instances, clusters, and similarity measurement.** *AllElectronics* clusters its customers so that each group of customers can be assigned to a customer relationship manager. Suppose we want to specify that all customers at the same address are to be placed in the same group, which would allow more comprehensive service to families. This can be expressed using a must-link constraint on instances:

$$Constraint_{family}(x, y) : \text{must-link}(x, y) \text{ if } x.address = y.address.$$

AllElectronics has eight customer relationship managers. To ensure that they each have a similar workload, we place a constraint on clusters such that there should be eight clusters, and each cluster should have at least 10% of the customers and no more than 15% of the customers. We can calculate the spatial distance between two customers using the driving distance between the two. However, if two customers live in different countries, we have to use the flight distance instead. This is a constraint on similarity measurement. ■

Another way to categorize clustering constraints considers how firmly the constraints have to be respected. A constraint is **hard** if a clustering that violates the constraint is unacceptable. A constraint is **soft** if a clustering that violates the constraint is not preferable but acceptable when no better solution can be found. Soft constraints are also called *preferences*.

Example 11.23 **Hard and soft constraints.** For *AllElectronics*, $Constraint_{family}$ in Example 11.22 is a hard constraint because splitting a family into different clusters could prevent the company from providing comprehensive services to the family, leading to poor customer satisfaction. The constraint on the number of clusters (which corresponds to the number of customer relationship managers in the company) is also hard. Example 11.22 also has a constraint to balance the size of clusters. While satisfying this constraint is strongly preferred, the company is flexible in that it is willing to assign a senior and more capable customer relationship manager to oversee a larger cluster. Therefore, the constraint is soft. ■

Ideally, for a specific data set and a set of constraints, all clusterings satisfy the constraints. However, it is possible that there may be no clustering of the data set that

satisfies all the constraints. Trivially, if two constraints in the set conflict, then no clustering can satisfy them at the same time.

Example 11.24 Conflicting constraints. Consider these constraints:

$$\text{must-link}(x, y) \text{ if } dist(x, y) < 5$$
$$\text{cannot-link}(x, y) \text{ if } dist(x, y) > 3.$$

If a data set has two objects, x, y, such that $dist(x, y) = 4$, then no clustering can satisfy both constraints simultaneously.

Consider these two constraints:

$$\text{must-link}(x, y) \text{ if } dist(x, y) < 5$$
$$\text{must-link}(x, y) \text{ if } dist(x, y) < 3.$$

The second constraint is redundant given the first. Moreover, for a data set where the distance between any two objects is at least 5, every possible clustering of the objects satisfies the constraints. ■

"How can we measure the quality and the usefulness of a set of constraints?" In general, we consider either their informativeness, or their coherence. The **informativeness** is the amount of information carried by the constraints that is beyond the clustering model. Given a data set, D, a clustering method, $\mathcal{A}$, and a set of constraints, $\mathcal{C}$, the informativeness of $\mathcal{C}$ with respect to $\mathcal{A}$ on D can be measured by the fraction of constraints in $\mathcal{C}$ that are unsatisfied by the clustering computed by $\mathcal{A}$ on D. The higher the informativeness, the more specific the requirements and background knowledge that the constraints carry. The **coherence** of a set of constraints is the degree of agreement among the constraints themselves, which can be measured by the redundancy among the constraints.

11.4.2 Methods for Clustering with Constraints

Although we can categorize clustering constraints, applications may have very different constraints of specific forms. Consequently, various techniques are needed to handle specific constraints. In this section, we discuss the general principles of handling hard and soft constraints.

Handling Hard Constraints

A general strategy for handling hard constraints is to strictly respect the constraints in the cluster assignment process. To illustrate this idea, we will use partitioning clustering as an example.

Given a data set and a set of constraints on instances (i.e., must-link or cannot-link constraints), how can we extend the k-means method to satisfy such constraints? The **COP-k-means algorithm** works as follows:

1. **Generate superinstances for must-link constraints**. Compute the transitive closure of the must-link constraints. Here, all must-link constraints are treated as an equivalence relation. The closure gives one or multiple subsets of objects where all objects in a subset must be assigned to one cluster. To represent such a subset, we replace all those objects in the subset by the mean. The superinstance also carries a weight, which is the number of objects it represents.

   After this step, the must-link constraints are always satisfied.

2. **Conduct modified k-means clustering.** Recall that, in k-means, an object is assigned to the closest center. What if a nearest-center assignment violates a cannot-link constraint? To respect cannot-link constraints, we modify the center assignment process in k-means to a *nearest feasible center assignment*. That is, when the objects are assigned to centers in sequence, at each step we make sure the assignments so far do not violate any cannot-link constraints. An object is assigned to the nearest center so that the assignment respects all cannot-link constraints.

Because COP-k-means ensures that no constraints are violated at every step, it does not require any backtracking. It is a greedy algorithm for generating a clustering that satisfies all constraints, provided that no conflicts exist among the constraints.

Handling Soft Constraints

Clustering with soft constraints is an optimization problem. When a clustering violates a soft constraint, a penalty is imposed on the clustering. Therefore, the optimization goal of the clustering contains two parts: optimizing the clustering quality and minimizing the constraint violation penalty. The overall objective function is a combination of the clustering quality score and the penalty score.

To illustrate, we again use partitioning clustering as an example. Given a data set and a set of soft constraints on instances, the **CVQE** (**Constrained Vector Quantization Error**) **algorithm** conducts k-means clustering while enforcing constraint violation penalties. The objective function used in CVQE is the sum of the distance used in k-means, adjusted by the constraint violation penalties, which are calculated as follows.

- **Penalty of a must-link violation.** If there is a must-link constraint on objects x and y, but they are assigned to two different centers, c_1 and c_2, respectively, then the constraint is violated. As a result, $dist(c_1, c_2)$, the distance between c_1 and c_2, is added to the objective function as the penalty.
- **Penalty of a cannot-link violation.** If there is a cannot-link constraint on objects x and y, but they are assigned to a common center, c, then the constraint is violated.

The distance, $dist(c, c')$, between c and c' is added to the objective function as the penalty.

Speeding up Constrained Clustering

Constraints, such as on similarity measurements, can lead to heavy costs in clustering. Consider the following **clustering with obstacles** problem: To cluster people as moving objects in a plaza, Euclidean distance is used to measure the walking distance between two points. However, a constraint on similarity measurement is that the trajectory implementing the shortest distance cannot cross a wall (Section 11.4.1). Because obstacles may occur between objects, the distance between two objects may have to be derived by geometric computations (e.g., involving triangulation). The computational cost is high if a large number of objects and obstacles are involved.

The clustering with obstacles problem can be represented using a graphical notation. First, a point, $\boldsymbol{p}$, is **visible** from another point, $\boldsymbol{q}$, in the region R if the straight line joining $\boldsymbol{p}$ and $\boldsymbol{q}$ does not intersect any obstacles. A **visibility graph** is the graph, $VG = (V, E)$, such that each vertex of the obstacles has a corresponding node in V and two nodes, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, in V are joined by an edge in E if and only if the corresponding vertices they represent are visible to each other. Let $VG' = (V', E')$ be a visibility graph created from VG by adding two additional points, $\boldsymbol{p}$ and $\boldsymbol{q}$, in V'. E' contains an edge joining two points in V' if the two points are mutually visible. The shortest path between two points, $\boldsymbol{p}$ and $\boldsymbol{q}$, will be a subpath of VG', as shown in Figure 11.16(a). We see that it begins with an edge from $\boldsymbol{p}$ to either $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, or $\boldsymbol{v_3}$, goes through a path in VG, and then ends with an edge from either $\boldsymbol{v_4}$ or $\boldsymbol{v_5}$ to $\boldsymbol{q}$.

To reduce the cost of distance computation between any two pairs of objects or points, several preprocessing and optimization techniques can be used. One method groups points that are close together into microclusters. This can be done by first triangulating the region R into triangles, and then grouping nearby points in the same triangle into microclusters, using a method similar to BIRCH or DBSCAN, as shown in Figure 11.16(b). By processing microclusters rather than individual points, the overall computation is reduced. After that, precomputation can be performed to build two

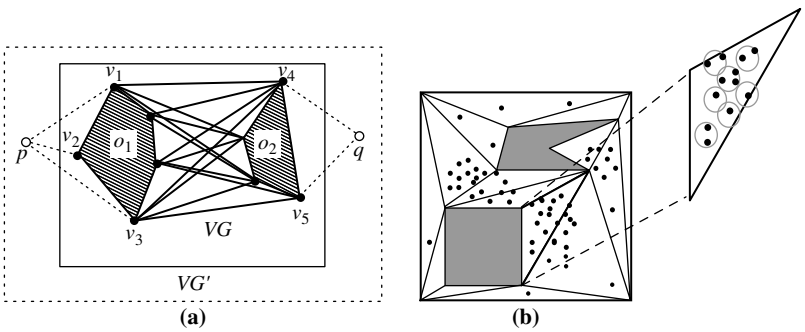

Figure 11.16 Clustering with obstacle objects (o_1 and o_2): (a) a visibility graph and (b) triangulation of regions with microclusters. *Source:* Adapted from Tung, Hou, and Han [THH01].

kinds of join indices based on the computation of the shortest paths: (1) *VV indices*, for any pair of obstacle vertices, and (2) *MV indices*, for any pair of microcluster and obstacle vertex. Use of the indices helps further optimize the overall performance.

Using such precomputation and optimization strategies, the distance between any two points (at the granularity level of a microcluster) can be computed efficiently. Thus, the clustering process can be performed in a manner similar to a typical efficient k-medoids algorithm, such as CLARANS, and achieve good clustering quality for large data sets.

11.5 Summary

- In conventional cluster analysis, an object is assigned to one cluster exclusively. However, in some applications, there is a need to assign an object to one or more clusters in a fuzzy or probabilistic way. **Fuzzy clustering** and **probabilistic model-based clustering** allow an object to belong to one or more clusters. A **partition matrix** records the membership degree of objects belonging to clusters.
- **Probabilistic model-based clustering** assumes that a cluster is a parameterized distribution. Using the data to be clustered as the observed samples, we can estimate the parameters of the clusters.
- A **mixture model** assumes that a set of observed objects is a mixture of instances from multiple probabilistic clusters. Conceptually, each observed object is generated independently by first choosing a probabilistic cluster according to the probabilities of the clusters, and then choosing a sample according to the probability density function of the chosen cluster.
- An **expectation-maximization algorithm** is a framework for approaching maximum likelihood or maximum a posteriori estimates of parameters in statistical models. Expectation-maximization algorithms can be used to compute fuzzy clustering and probabilistic model-based clustering.
- **High-dimensional data** pose several challenges for cluster analysis, including how to model high-dimensional clusters and how to search for such clusters.
- There are two major categories of clustering methods for high-dimensional data: subspace clustering methods and dimensionality reduction methods. **Subspace clustering methods** search for clusters in subspaces of the original space. Examples include **subspace search methods**, **correlation-based clustering methods**, and **biclustering methods**. **Dimensionality reduction methods** create a new space of lower dimensionality and search for clusters there.
- **Biclustering methods** cluster objects and attributes simultaneously. Types of biclusters include biclusters with **constant values**, **constant values on rows/columns**, **coherent values**, and **coherent evolutions on rows/columns**. Two major types of biclustering methods are **optimization-based methods** and **enumeration methods**.

- **Spectral clustering** is a **dimensionality reduction method**. The general idea is to construct new dimensions using an affinity matrix.
- **Clustering graph and network data** has many applications such as social network analysis. Challenges include how to measure the similarity between objects in a graph, and how to design clustering models and methods for graph and network data.
- **Geodesic distance** is the number of edges between two vertices on a graph. It can be used to measure similarity. Alternatively, similarity in graphs, such as social networks, can be measured using structural context and random walk. **SimRank** is a similarity measure that is based on both structural context and random walk.
- Graph clustering can be modeled as computing **graph cuts**. A **sparsest cut** may lead to a good clustering, while **modularity** can be used to measure the clustering quality.
- **SCAN** is a graph clustering algorithm that searches graphs to identify well-connected components as clusters.
- **Constraints** can be used to express application-specific requirements or background knowledge for cluster analysis. Constraints for clustering can be categorized as constraints on **instances**, on **clusters**, or on **similarity measurement**. Constraints on instances include **must-link** and **cannot-link** constraints. A constraint can be **hard** or **soft**.
- **Hard constraints for clustering** can be enforced by strictly respecting the constraints in the cluster assignment process. **Clustering with soft constraints** can be considered an optimization problem. Heuristics can be used to speed up constrained clustering.

11.6 Exercises

11.1 Traditional clustering methods are rigid in that they require each object to belong exclusively to only one cluster. Explain why this is a special case of fuzzy clustering. You may use k-means as an example.

11.2 *AllElectronics* carries 1000 products, $P_1, \ldots, P_{1000}$. Consider customers Ada, Bob, and Cathy such that Ada and Bob purchase three products in common, P_1, P_2, and P_3. For the other 997 products, Ada and Bob independently purchase seven of them randomly. Cathy purchases 10 products, randomly selected from the 1000 products. In Euclidean distance, what is the probability that $dist(\text{Ada}, \text{Bob}) > dist(\text{Ada}, \text{Cathy})$? What if Jaccard similarity (Chapter 2) is used? What can you learn from this example?

11.3 Show that $I \times J$ is a bicluster with coherent values if and only if, for any $i_1, i_2 \in I$ and $j_1, j_2 \in J$, $e_{i_1 j_1} - e_{i_2 j_1} = e_{i_1 j_2} - e_{i_2 j_2}$.

11.4 Compare the MaPle algorithm (Section 11.2.3) with the frequent closed itemset mining algorithm, CLOSET (Pei, Han, and Mao [PHM00]). What are the major similarities and differences?

11.5 SimRank is a similarity measure for clustering graph and network data.

(a) Prove $\lim_{i \to \infty} s_i(u, v) = s(u, v)$ for SimRank computation.

(b) Show $s(u, v) = p(u, v)$ for SimRank.

11.6 In a large sparse graph where on average each node has a low degree, is the similarity matrix using SimRank still sparse? If so, in what sense? If not, why? Deliberate on your answer.

11.7 Compare the SCAN algorithm (Section 11.3.3) with DBSCAN (Section 10.4.1). What are their similarities and differences?

11.8 Consider partitioning clustering and the following constraint on clusters: The number of objects in each cluster must be between $\frac{n}{k}(1 - \delta)$ and $\frac{n}{k}(1 + \delta)$, where n is the total number of objects in the data set, k is the number of clusters desired, and δ in $[0, 1)$ is a parameter. Can you extend the k-means method to handle this constraint? Discuss situations where the constraint is hard and soft.

11.7 Bibliographic Notes

Höppner Klawonn, Kruse, and Runkler [HKKR99] provide a thorough discussion of fuzzy clustering. The fuzzy c-means algorithm (on which Example 11.7 is based) was proposed by Bezdek [Bez81]. Fraley and Raftery [FR02] give a comprehensive overview of model-based cluster analysis and probabilistic models. McLachlan and Basford [MB88] present a systematic introduction to mixture models and applications in cluster analysis.

Dempster, Laird, and Rubin [DLR77] are recognized as the first to introduce the EM algorithm and give it its name. However, the idea of the EM algorithm had been "proposed many times in special circumstances" before, as admitted in Dempster, Laird, and Rubin [DLR77]. Wu [Wu83] gives the correct analysis of the EM algorithm.

Mixture models and EM algorithms are used extensively in many data mining applications. Introductions to model-based clustering, mixture models, and EM algorithms can be found in recent textbooks on machine learning and statistical learning—for example, Bishop [Bis06], Marsland [Mar09], and Alpaydin [Alp11].

The increase of dimensionality has severe effects on distance functions, as indicated by Beyer et al. [BGRS99]. It also has had a dramatic impact on various techniques for classification, clustering, and semisupervised learning (Radovanović, Nanopoulos, and Ivanović [RNI09]).

Kriegel, Kröger, and Zimek [KKZ09] present a comprehensive survey on methods for clustering high-dimensional data. The CLIQUE algorithm was developed by Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98]. The PROCLUS algorithm was proposed by Aggawal, Procopiuc, Wolf, et al. [APW+99].

The technique of biclustering was initially proposed by Hartigan [Har72]. The term *biclustering* was coined by Mirkin [Mir98]. Cheng and Church [CC00] introduced

biclustering into gene expression data analysis. There are many studies on biclustering models and methods. The notion of δ-pCluster was introduced by Wang, Wang, Yang, and Yu [WWYY02]. For informative surveys, see Madeira and Oliveira [MO04] and Tanay, Sharan, and Shamir [TSS04]. In this chapter, we introduced the δ-cluster algorithm by Cheng and Church [CC00] and MaPle by Pei, Zhang, Cho, et al. [PZC+03] as examples of optimization-based methods and enumeration methods for biclustering, respectively.

Donath and Hoffman [DH73] and Fiedler [Fie73] pioneered spectral clustering. In this chapter, we use an algorithm proposed by Ng, Jordan, and Weiss [NJW01] as an example. For a thorough tutorial on spectral clustering, see Luxburg [Lux07].

Clustering graph and network data is an important and fast-growing topic. Schaeffer [Sch07] provides a survey. The SimRank measure of similarity was developed by Jeh and Widom [JW02a]. Xu et al. [XYFS07] proposed the SCAN algorithm. Arora, Rao, and Vazirani [ARV09] discuss the sparsest cuts and approximation algorithms.

Clustering with constraints has been extensively studied. Davidson, Wagstaff, and Basu [DWB06] proposed the measures of informativeness and coherence. The COP-*k*-means algorithm is given by Wagstaff et al. [WCRS01]. The CVQE algorithm was proposed by Davidson and Ravi [DR05]. Tung, Han, Lakshmanan, and Ng [THLN01] presented a framework for constraint-based clustering based on user-specified constraints. An efficient method for constraint-based spatial clustering in the existence of physical obstacle constraints was proposed by Tung, Hou, and Han [THH01].

This page intentionally left blank

12 Outlier Detection

Imagine that you are a transaction auditor in a credit card company. To protect your customers from credit card fraud, you pay special attention to card usages that are rather different from typical cases. For example, if a purchase amount is much bigger than usual for a card owner, and if the purchase occurs far from the owner's resident city, then the purchase is suspicious. You want to detect such transactions as soon as they occur and contact the card owner for verification. This is common practice in many credit card companies. *What data mining techniques can help detect suspicious transactions?*

Most credit card transactions are normal. However, if a credit card is stolen, its transaction pattern usually changes dramatically—the locations of purchases and the items purchased are often very different from those of the authentic card owner and other customers. An essential idea behind credit card fraud detection is to identify those transactions that are very different from the norm.

Outlier detection (also known as *anomaly detection*) is the process of finding data objects with behaviors that are very different from expectation. Such objects are called **outliers** or **anomalies**. Outlier detection is important in many applications in addition to fraud detection such as medical care, public safety and security, industry damage detection, image processing, sensor/video network surveillance, and intrusion detection.

Outlier detection and clustering analysis are two highly related tasks. Clustering finds the majority patterns in a data set and organizes the data accordingly, whereas outlier detection tries to capture those exceptional cases that deviate substantially from the majority patterns. Outlier detection and clustering analysis serve different purposes.

In this chapter, we study outlier detection techniques. Section 12.1 defines the different types of outliers. Section 12.2 presents an overview of outlier detection methods. In the rest of the chapter, you will learn about outlier detection methods in detail. These approaches, organized here by category, are statistical (Section 12.3), proximity-based (Section 12.4), clustering-based (Section 12.5), and classification-based (Section 12.6). In addition, you will learn about mining contextual and collective outliers (Section 12.7) and outlier detection in high-dimensional data (Section 12.8).

12.1 Outliers and Outlier Analysis

Let us first define what outliers are, categorize the different types of outliers, and then discuss the challenges in outlier detection at a general level.

12.1.1 What Are Outliers?

Assume that a given statistical process is used to generate a set of data objects. An **outlier** is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism. For ease of presentation within this chapter, we may refer to data objects that are not outliers as "normal" or expected data. Similarly, we may refer to outliers as "abnormal" data.

Example 12.1 **Outliers.** In Figure 12.1, most objects follow a roughly Gaussian distribution. However, the objects in region *R* are significantly different. It is unlikely that they follow the same distribution as the other objects in the data set. Thus, the objects in *R* are outliers in the data set. ■

Outliers are different from noisy data. As mentioned in Chapter 3, noise is a random error or variance in a measured variable. In general, noise is not interesting in data analysis, including outlier detection. For example, in credit card fraud detection, a customer's purchase behavior can be modeled as a random variable. A customer may generate some "noise transactions" that may seem like "random errors" or "variance," such as by buying a bigger lunch one day, or having one more cup of coffee than usual. Such transactions should not be treated as outliers; otherwise, the credit card company would incur heavy costs from verifying that many transactions. The company may also lose customers by bothering them with multiple false alarms. As in many other data analysis and data mining tasks, noise should be removed before outlier detection.

Outliers are interesting because they are suspected of not being generated by the same mechanisms as the rest of the data. Therefore, in outlier detection, it is important to

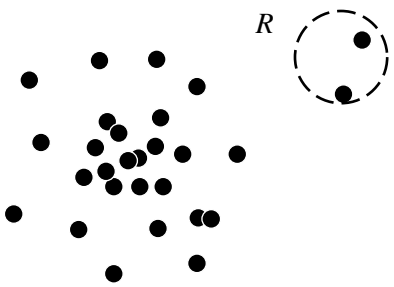

Figure 12.1 The objects in region *R* are outliers.

justify *why* the outliers detected are generated by some other mechanisms. This is often achieved by making various assumptions on the rest of the data and showing that the outliers detected violate those assumptions significantly.

Outlier detection is also related to *novelty detection* in evolving data sets. For example, by monitoring a social media web site where new content is incoming, novelty detection may identify new topics and trends in a timely manner. Novel topics may initially appear as outliers. To this extent, outlier detection and novelty detection share some similarity in modeling and detection methods. However, a critical difference between the two is that in novelty detection, once new topics are confirmed, they are usually incorporated into the model of normal behavior so that follow-up instances are not treated as outliers anymore.

12.1.2 Types of Outliers

In general, outliers can be classified into three categories, namely global outliers, contextual (or conditional) outliers, and collective outliers. Let's examine each of these categories.

Global Outliers

In a given data set, a data object is a **global outlier** if it deviates significantly from the rest of the data set. Global outliers are sometimes called *point anomalies*, and are the simplest type of outliers. Most outlier detection methods are aimed at finding global outliers.

Example 12.2 **Global outliers.** Consider the points in Figure 12.1 again. The points in region R significantly deviate from the rest of the data set, and hence are examples of global outliers. ■

To detect global outliers, a critical issue is to find an appropriate measurement of deviation with respect to the application in question. Various measurements are proposed, and, based on these, outlier detection methods are partitioned into different categories. We will come to this issue in detail later.

Global outlier detection is important in many applications. Consider intrusion detection in computer networks, for example. If the communication behavior of a computer is very different from the normal patterns (e.g., a large number of packages is broadcast in a short time), this behavior may be considered as a global outlier and the corresponding computer is a suspected victim of hacking. As another example, in trading transaction auditing systems, transactions that do not follow the regulations are considered as global outliers and should be held for further examination.

Contextual Outliers

"*The temperature today is* 28°*C. Is it exceptional (i.e., an outlier)?*" It depends, for example, on the time and location! If it is in winter in Toronto, yes, it is an outlier. If it is a summer day in Toronto, then it is normal. Unlike global outlier detection, in this case,

whether or not today's temperature value is an outlier depends on the context—the date, the location, and possibly some other factors.

In a given data set, a data object is a **contextual outlier** if it deviates significantly with respect to a specific context of the object. Contextual outliers are also known as *conditional outliers* because they are conditional on the selected context. Therefore, in contextual outlier detection, the context has to be specified as part of the problem definition. Generally, in contextual outlier detection, the attributes of the data objects in question are divided into two groups:

- **Contextual attributes**: The contextual attributes of a data object define the object's context. In the temperature example, the contextual attributes may be date and location.
- **Behavioral attributes**: These define the object's characteristics, and are used to evaluate whether the object is an outlier in the context to which it belongs. In the temperature example, the behavioral attributes may be the temperature, humidity, and pressure.

Unlike global outlier detection, in contextual outlier detection, whether a data object is an outlier depends on not only the behavioral attributes but also the contextual attributes. A configuration of behavioral attribute values may be considered an outlier in one context (e.g., 28°C is an outlier for a Toronto winter), but not an outlier in another context (e.g., 28°C is not an outlier for a Toronto summer).

Contextual outliers are a generalization of local outliers, a notion introduced in density-based outlier analysis approaches. An object in a data set is a **local outlier** if its density significantly deviates from the local area in which it occurs. We will discuss local outlier analysis in greater detail in Section 12.4.3.

Global outlier detection can be regarded as a special case of contextual outlier detection where the set of contextual attributes is empty. In other words, global outlier detection uses the whole data set as the context. Contextual outlier analysis provides flexibility to users in that one can examine outliers in different contexts, which can be highly desirable in many applications.

Example 12.3 **Contextual outliers.** In credit card fraud detection, in addition to global outliers, an analyst may consider outliers in different contexts. Consider customers who use more than 90% of their credit limit. If one such customer is viewed as belonging to a group of customers with low credit limits, then such behavior may not be considered an outlier. However, similar behavior of customers from a high-income group may be considered outliers if their balance often exceeds their credit limit. Such outliers may lead to business opportunities—raising credit limits for such customers can bring in new revenue. ■

The quality of contextual outlier detection in an application depends on the meaningfulness of the contextual attributes, in addition to the measurement of the deviation of an object to the majority in the space of behavioral attributes. More often than not, the contextual attributes should be determined by domain experts, which can be regarded as part of the input background knowledge. In many applications, neither obtaining sufficient information to determine contextual attributes nor collecting high-quality contextual attribute data is easy.

"*How can we formulate meaningful contexts in contextual outlier detection?*" A straightforward method simply uses group-bys of the contextual attributes as contexts. This may not be effective, however, because many group-bys may have insufficient data and/or noise. A more general method uses the proximity of data objects in the space of contextual attributes. We discuss this approach in detail in Section 12.4.

Collective Outliers

Suppose you are a supply-chain manager of *AllElectronics*. You handle thousands of orders and shipments every day. If the shipment of an order is delayed, it may not be considered an outlier because, statistically, delays occur from time to time. However, you have to pay attention if 100 orders are delayed on a single day. Those 100 orders as a whole form an outlier, although each of them may not be regarded as an outlier if considered individually. You may have to take a close look at those orders collectively to understand the shipment problem.

Given a data set, a subset of data objects forms a **collective outlier** if the objects as a whole deviate significantly from the entire data set. Importantly, the individual data objects may not be outliers.

Example 12.4 **Collective outliers.** In Figure 12.2, the black objects as a whole form a collective outlier because the density of those objects is much higher than the rest in the data set. However, every black object individually is not an outlier with respect to the whole data set.

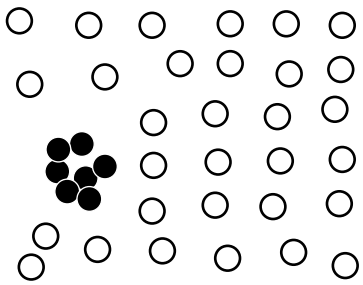

Figure 12.2 The black objects form a collective outlier. ■

Collective outlier detection has many important applications. For example, in intrusion detection, a denial-of-service package from one computer to another is considered normal, and not an outlier at all. However, if several computers keep sending denial-of-service packages to each other, they as a whole should be considered as a collective outlier. The computers involved may be suspected of being compromised by an attack. As another example, a stock transaction between two parties is considered normal. However, a large set of transactions of the same stock among a small party in a short period are collective outliers because they may be evidence of some people manipulating the market.

Unlike global or contextual outlier detection, in collective outlier detection we have to consider not only the behavior of individual objects, but also that of groups of objects. Therefore, to detect collective outliers, we need background knowledge of the relationship among data objects such as distance or similarity measurements between objects.

In summary, a data set can have multiple types of outliers. Moreover, an object may belong to more than one type of outlier. In business, different outliers may be used in various applications or for different purposes. Global outlier detection is the simplest. Context outlier detection requires background information to determine contextual attributes and contexts. Collective outlier detection requires background information to model the relationship among objects to find groups of outliers.

12.1.3 Challenges of Outlier Detection

Outlier detection is useful in many applications yet faces many challenges such as the following:

- **Modeling normal objects and outliers effectively.** Outlier detection quality highly depends on the modeling of normal (nonoutlier) objects and outliers. Often, building a comprehensive model for data normality is very challenging, if not impossible. This is partly because it is hard to enumerate all possible normal behaviors in an application.

  The border between data normality and abnormality (outliers) is often not clear cut. Instead, there can be a wide range of gray area. Consequently, while some outlier detection methods assign to each object in the input data set a label of either "normal" or "outlier," other methods assign to each object a score measuring the "outlier-ness" of the object.

- **Application-specific outlier detection.** Technically, choosing the similarity/distance measure and the relationship model to describe data objects is critical in outlier detection. Unfortunately, such choices are often application-dependent. Different applications may have very different requirements. For example, in clinic data analysis, a small deviation may be important enough to justify an outlier. In contrast, in marketing analysis, objects are often subject to larger fluctuations, and consequently a substantially larger deviation is needed to justify an outlier. Outlier detection's high

dependency on the application type makes it impossible to develop a universally applicable outlier detection method. Instead, individual outlier detection methods that are dedicated to specific applications must be developed.

- **Handling noise in outlier detection.** As mentioned earlier, outliers are different from noise. It is also well known that the quality of real data sets tends to be poor. Noise often unavoidably exists in data collected in many applications. Noise may be present as deviations in attribute values or even as missing values. Low data quality and the presence of noise bring a huge challenge to outlier detection. They can distort the data, blurring the distinction between normal objects and outliers. Moreover, noise and missing data may "hide" outliers and reduce the effectiveness of outlier detection—an outlier may appear "disguised" as a noise point, and an outlier detection method may mistakenly identify a noise point as an outlier.
- **Understandability.** In some application scenarios, a user may want to not only detect outliers, but also understand why the detected objects are outliers. To meet the understandability requirement, an outlier detection method has to provide some justification of the detection. For example, a statistical method can be used to justify the degree to which an object may be an outlier based on the likelihood that the object was generated by the same mechanism that generated the majority of the data. The smaller the likelihood, the more unlikely the object was generated by the same mechanism, and the more likely the object is an outlier.

The rest of this chapter discusses approaches to outlier detection.

12.2 Outlier Detection Methods

There are many outlier detection methods in the literature and in practice. Here, we present two orthogonal ways to categorize outlier detection methods. First, we categorize outlier detection methods according to whether the sample of data for analysis is given with domain expert–provided labels that can be used to build an outlier detection model. Second, we divide methods into groups according to their assumptions regarding normal objects versus outliers.

12.2.1 Supervised, Semi-Supervised, and Unsupervised Methods

If expert-labeled examples of normal and/or outlier objects can be obtained, they can be used to build outlier detection models. The methods used can be divided into supervised methods, semi-supervised methods, and unsupervised methods.

Supervised Methods

Supervised methods model data normality and abnormality. Domain experts examine and label a sample of the underlying data. Outlier detection can then be modeled as

a classification problem (Chapters 8 and 9). The task is to learn a classifier that can recognize outliers. The sample is used for training and testing. In some applications, the experts may label just the normal objects, and any other objects not matching the model of normal objects are reported as outliers. Other methods model the outliers and treat objects not matching the model of outliers as normal.

Although many classification methods can be applied, challenges to supervised outlier detection include the following:

- The two classes (i.e., normal objects versus outliers) are imbalanced. That is, the population of outliers is typically much smaller than that of normal objects. Therefore, methods for handling imbalanced classes (Section 8.6.5) may be used, such as oversampling (i.e., replicating) outliers to increase their distribution in the training set used to construct the classifier. Due to the small population of outliers in data, the sample data examined by domain experts and used in training may not even sufficiently represent the outlier distribution. The lack of outlier samples can limit the capability of classifiers built as such. To tackle these problems, some methods "make up" artificial outliers.
- In many outlier detection applications, catching as many outliers as possible (i.e., the sensitivity or recall of outlier detection) is far more important than not mislabeling normal objects as outliers. Consequently, when a classification method is used for supervised outlier detection, it has to be interpreted appropriately so as to consider the application interest on recall.

In summary, supervised methods of outlier detection must be careful in how they train and how they interpret classification rates due to the fact that outliers are rare in comparison to the other data samples.

Unsupervised Methods

In some application scenarios, objects labeled as "normal" or "outlier" are not available. Thus, an unsupervised learning method has to be used.

Unsupervised outlier detection methods make an implicit assumption: The normal objects are somewhat "clustered." In other words, an unsupervised outlier detection method expects that normal objects follow a pattern far more frequently than outliers. Normal objects do not have to fall into one group sharing high similarity. Instead, they can form multiple groups, where each group has distinct features. However, an outlier is expected to occur far away in feature space from any of those groups of normal objects.

This assumption may not be true all the time. For example, in Figure 12.2, the normal objects do not share any strong patterns. Instead, they are uniformly distributed. The collective outliers, however, share high similarity in a small area. Unsupervised methods cannot detect such outliers effectively. In some applications, normal objects are diversely distributed, and many such objects do not follow strong patterns. For instance, in some intrusion detection and computer virus detection problems, normal activities are very diverse and many do not fall into high-quality clusters. In such scenarios, unsupervised

methods may have a high false positive rate—they may mislabel many normal objects as outliers (intrusions or viruses in these applications), and let many actual outliers go undetected. Due to the high similarity between intrusions and viruses (i.e., they have to attack key resources in the target systems), modeling outliers using supervised methods may be far more effective.

Many clustering methods can be adapted to act as unsupervised outlier detection methods. The central idea is to find clusters first, and then the data objects not belonging to any cluster are detected as outliers. However, such methods suffer from two issues. First, a data object not belonging to any cluster may be noise instead of an outlier. Second, it is often costly to find clusters first and then find outliers. It is usually assumed that there are far fewer outliers than normal objects. Having to process a large population of nontarget data entries (i.e., the normal objects) before one can touch the real meat (i.e., the outliers) can be unappealing. The latest unsupervised outlier detection methods develop various smart ideas to tackle outliers directly without explicitly and completely finding clusters. You will learn more about these techniques in Sections 12.4 and 12.5 on proximity-based and clustering-based methods, respectively.

Semi-Supervised Methods

In many applications, although obtaining some labeled examples is feasible, the number of such labeled examples is often small. We may encounter cases where only a small set of the normal and/or outlier objects are labeled, but most of the data are unlabeled. Semi-supervised outlier detection methods were developed to tackle such scenarios.

Semi-supervised outlier detection methods can be regarded as applications of semi-supervised learning methods (Section 9.7.2). For example, when some labeled normal objects are available, we can use them, together with unlabeled objects that are close by, to train a model for normal objects. The model of normal objects then can be used to detect outliers—those objects not fitting the model of normal objects are classified as outliers.

If only some labeled outliers are available, semi-supervised outlier detection is trickier. A small number of labeled outliers are unlikely to represent all the possible outliers. Therefore, building a model for outliers based on only a few labeled outliers is unlikely to be effective. To improve the quality of outlier detection, we can get help from models for normal objects learned from unsupervised methods.

For additional information on semi-supervised methods, interested readers are referred to the bibliographic notes at the end of this chapter (Section 12.11).

12.2.2 Statistical Methods, Proximity-Based Methods, and Clustering-Based Methods

As discussed in Section 12.1, outlier detection methods make assumptions about outliers versus the rest of the data. According to the assumptions made, we can categorize outlier detection methods into three types: statistical methods, proximity-based methods, and clustering-based methods.

Statistical Methods

Statistical methods (also known as **model-based methods**) make assumptions of data normality. They assume that normal data objects are generated by a statistical (stochastic) model, and that data not following the model are outliers.

Example 12.5 **Detecting outliers using a statistical (Gaussian) model.** In Figure 12.1, the data points except for those in region R fit a Gaussian distribution g_D, where for a location $\boldsymbol{x}$ in the data space, $g_D(\boldsymbol{x})$ gives the probability density at $\boldsymbol{x}$. Thus, the Gaussian distribution g_D can be used to model the normal data, that is, most of the data points in the data set. For each object $\boldsymbol{y}$ in region, R, we can estimate $g_D(\boldsymbol{y})$, the probability that this point fits the Gaussian distribution. Because $g_D(\boldsymbol{y})$ is very low, $\boldsymbol{y}$ is unlikely generated by the Gaussian model, and thus is an outlier. ■

The effectiveness of statistical methods highly depends on whether the assumptions made for the statistical model hold true for the given data. There are many kinds of statistical models. For example, the statistic models used in the methods may be parametric or nonparametric. Statistical methods for outlier detection are discussed in detail in Section 12.3.

Proximity-Based Methods

Proximity-based methods assume that an object is an outlier if the nearest neighbors of the object are far away in feature space, that is, the proximity of the object to its neighbors significantly deviates from the proximity of most of the other objects to their neighbors in the same data set.

Example 12.6 **Detecting outliers using proximity.** Consider the objects in Figure 12.1 again. If we model the proximity of an object using its three nearest neighbors, then the objects in region R are substantially different from other objects in the data set. For the two objects in R, their second and third nearest neighbors are dramatically more remote than those of any other objects. Therefore, we can label the objects in R as outliers based on proximity. ■

The effectiveness of proximity-based methods relies heavily on the proximity (or distance) measure used. In some applications, such measures cannot be easily obtained. Moreover, proximity-based methods often have difficulty in detecting a group of outliers if the outliers are close to one another.

There are two major types of proximity-based outlier detection, namely *distance-based* and *density-based* outlier detection. Proximity-based outlier detection is discussed in Section 12.4.

Clustering-Based Methods

Clustering-based methods assume that the normal data objects belong to large and dense clusters, whereas outliers belong to small or sparse clusters, or do not belong to any clusters.

Example 12.7 **Detecting outliers using clustering.** In Figure 12.1, there are two clusters. Cluster C_1 contains all the points in the data set except for those in region R. Cluster C_2 is tiny, containing just two points in R. Cluster C_1 is large in comparison to C_2. Therefore, a clustering-based method asserts that the two objects in R are outliers. ■

There are many clustering methods, as discussed in Chapters 10 and 11. Therefore, there are many clustering-based outlier detection methods as well. Clustering is an expensive data mining operation. A straightforward adaptation of a clustering method for outlier detection can be very costly, and thus does not scale up well for large data sets. Clustering-based outlier detection methods are discussed in detail in Section 12.5.

12.3 Statistical Approaches

As with statistical methods for clustering, statistical methods for outlier detection make assumptions about data normality. They assume that the normal objects in a data set are generated by a stochastic process (a generative model). Consequently, normal objects occur in regions of high probability for the stochastic model, and objects in the regions of low probability are outliers.

The general idea behind statistical methods for outlier detection is to learn a generative model fitting the given data set, and then identify those objects in low-probability regions of the model as outliers. However, there are many different ways to learn generative models. In general, statistical methods for outlier detection can be divided into two major categories: *parametric methods* and *nonparametric methods*, according to how the models are specified and learned.

A **parametric method** assumes that the normal data objects are generated by a parametric distribution with parameter Θ. The *probability density function* of the parametric distribution $f(\boldsymbol{x}, \Theta)$ gives the probability that object $\boldsymbol{x}$ is generated by the distribution. The smaller this value, the more likely $\boldsymbol{x}$ is an outlier.

A **nonparametric method** does not assume an a priori statistical model. Instead, a nonparametric method tries to determine the model from the input data. Note that most nonparametric methods do not assume that the model is completely parameter-free. (Such an assumption would make learning the model from data almost mission impossible.) Instead, nonparametric methods often take the position that the number and nature of the parameters are flexible and not fixed in advance. Examples of nonparametric methods include histogram and kernel density estimation.

12.3.1 Parametric Methods

In this subsection, we introduce several simple yet practical parametric methods for outlier detection. We first discuss methods for univariate data based on normal distribution. We then discuss how to handle multivariate data using multiple parametric distributions.

Detection of Univariate Outliers Based on Normal Distribution

Data involving only one attribute or variable are called *univariate data*. For simplicity, we often choose to assume that data are generated from a normal distribution. We can then learn the parameters of the normal distribution from the input data, and identify the points with low probability as outliers.

Let's start with univariate data. We will try to detect outliers by assuming the data follow a normal distribution.

Example 12.8 **Univariate outlier detection using maximum likelihood.** Suppose a city's average temperature values in July in the last 10 years are, in value-ascending order, 24.0°C, 28.9°C, 28.9°C, 29.0°C, 29.1°C, 29.1°C, 29.2°C, 29.2°C, 29.3°C, and 29.4°C. Let's assume that the average temperature follows a normal distribution, which is determined by two parameters: the mean, μ, and the standard deviation, σ.

We can use the *maximum likelihood method* to estimate the parameters μ and σ. That is, we maximize the *log-likelihood function*

$$\ln \mathcal{L}(\mu, \sigma^2) = \sum_{i=1}^{n} \ln f(x_i|(\mu, \sigma^2)) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2, \quad (12.1)$$

where n is the total number of samples, which is 10 in this example.

Taking derivatives with respect to μ and σ^2 and solving the resulting system of first-order conditions leads to the following *maximum likelihood estimates*:

$$\hat{\mu} = \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (12.2)$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2. \quad (12.3)$$

In this example, we have

$$\hat{\mu} = \frac{24.0 + 28.9 + 28.9 + 29.0 + 29.1 + 29.1 + 29.2 + 29.2 + 29.3 + 29.4}{10} = 28.61$$

$$\begin{aligned}\hat{\sigma}^2 = ((&24.1 - 28.61)^2 + (28.9 - 28.61)^2 + (28.9 - 28.61)^2 + (29.0 - 28.61)^2 \\ &+ (29.1 - 28.61)^2 + (29.1 - 28.61)^2 + (29.2 - 28.61)^2 + (29.2 - 28.61)^2 \\ &+ (29.3 - 28.61)^2 + (29.4 - 28.61)^2)/10 \backsimeq 2.29.\end{aligned}$$

Accordingly, we have $\hat{\sigma} = \sqrt{2.29} = 1.51$.

The most deviating value, 24.0°C, is 4.61°C away from the estimated mean. We know that the $\mu \pm 3\sigma$ region contains 99.7% data under the assumption of normal

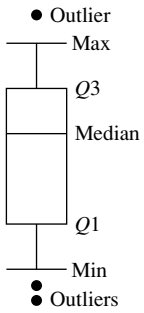

Figure 12.3 Using a boxplot to visualize outliers.

distribution. Because $\frac{4.61}{1.51} = 3.04 > 3$, the probability that the value 24.0°C is generated by the normal distribution is less than 0.15%, and thus can be identified as an outlier. ■

Example 12.8 elaborates a simple yet practical outlier detection method. It simply labels any object as an outlier if it is more than 3σ away from the mean of the estimated distribution, where σ is the standard deviation.

Such straightforward methods for statistical outlier detection can also be used in visualization. For example, the *boxplot method* (described in Chapter 2) plots the univariate input data using a five-number summary (Figure 12.3): the smallest nonoutlier value (Min), the lower quartile ($Q1$), the median ($Q2$), the upper quartile ($Q3$), and the largest nonoutlier value (Max). The *interquantile range* (IQR) is defined as $Q3 - Q1$. Any object that is more than $1.5 \times IQR$ smaller than $Q1$ or $1.5 \times IQR$ larger than $Q3$ is treated as an outlier because the region between $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$ contains 99.3% of the objects. The rationale is similar to using 3σ as the threshold for normal distribution.

Another simple statistical method for univariate outlier detection using normal distribution is the *Grubb's test* (also known as the *maximum normed residual test*). For each object x in a data set, we define a z-score as

$$z = \frac{|x - \bar{x}|}{s}, \tag{12.4}$$

where $\bar{x}$ is the mean, and s is the standard deviation of the input data. An object x is an outlier if

$$z \geq \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/(2N),N-2}}{N-2+t^2_{\alpha/(2N),N-2}}}, \tag{12.5}$$

where $t^2_{\alpha/(2N),N-2}$ is the value taken by a t-distribution at a significance level of $\alpha/(2N)$, and N is the number of objects in the data set.

Detection of Multivariate Outliers

Data involving two or more attributes or variables are *multivariate data*. Many univariate outlier detection methods can be extended to handle multivariate data. The central idea is to transform the multivariate outlier detection task into a univariate outlier detection problem. Here, we use two examples to illustrate this idea.

Example 12.9 Multivariate outlier detection using the Mahalanobis distance. For a multivariate data set, let $\bar{\boldsymbol{o}}$ be the mean vector. For an object, $\boldsymbol{o}$, in the data set, the Mahalanobis distance from $\boldsymbol{o}$ to $\bar{\boldsymbol{o}}$ is

$$MDist(\boldsymbol{o}, \bar{\boldsymbol{o}}) = (\boldsymbol{o} - \bar{\boldsymbol{o}})^T S^{-1} (\boldsymbol{o} - \bar{\boldsymbol{o}}), \tag{12.6}$$

where S is the covariance matrix.

$MDist(\boldsymbol{o}, \bar{\boldsymbol{o}})$ is a univariate variable, and thus Grubb's test can be applied to this measure. Therefore, we can transform the multivariate outlier detection tasks as follows:

1. Calculate the mean vector from the multivariate data set.
2. For each object $\boldsymbol{o}$, calculate $MDist(\boldsymbol{o}, \bar{\boldsymbol{o}})$, the Mahalanobis distance from $\boldsymbol{o}$ to $\bar{\boldsymbol{o}}$.
3. Detect outliers in the transformed univariate data set, $\{MDist(\boldsymbol{o}, \bar{\boldsymbol{o}}) | \boldsymbol{o} \in D\}$.
4. If $MDist(\boldsymbol{o}, \bar{\boldsymbol{o}})$ is determined to be an outlier, then $\boldsymbol{o}$ is regarded as an outlier as well.

■

Our second example uses the χ^2-statistic to measure the distance between an object to the mean of the input data set.

Example 12.10 Multivariate outlier detection using the χ^2-statistic. The χ^2-statistic can also be used to capture multivariate outliers under the assumption of normal distribution. For an object, $\boldsymbol{o}$, the χ^2-statistic is

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - E_i)^2}{E_i}, \tag{12.7}$$

where $\boldsymbol{o}_i$ is the value of $\boldsymbol{o}$ on the ith dimension, E_i is the mean of the i-dimension among all objects, and n is the dimensionality. If the χ^2-statistic is large, the object is an outlier. ■

Using a Mixture of Parametric Distributions

If we assume that the data were generated by a normal distribution, this works well in many situations. However, this assumption may be overly simplified when the actual data distribution is complex. In such cases, we instead assume that the data were generated by a mixture of parametric distributions.

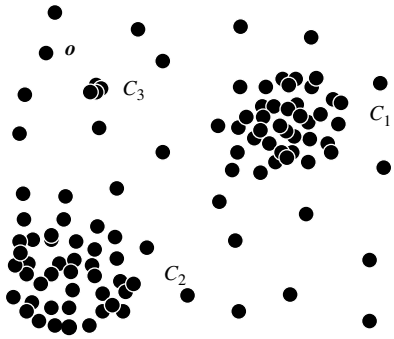

Figure 12.4 A complex data set.

Example 12.11 **Multivariate outlier detection using multiple parametric distributions.** Consider the data set in Figure 12.4. There are two big clusters, C_1 and C_2. To assume that the data are generated by a normal distribution would not work well here. The estimated mean is located between the two clusters and not inside any cluster. The objects between the two clusters cannot be detected as outliers since they are close to the mean. ■

To overcome this problem, we can instead assume that the normal data objects are generated by multiple normal distributions, two in this case. That is, we assume two normal distributions, $\Theta_1(\mu_1, \sigma_1)$ and $\Theta_2(\mu_2, \sigma_2)$. For any object, $\boldsymbol{o}$, in the data set, the probability that $\boldsymbol{o}$ is generated by the mixture of the two distributions is given by

$$Pr(\boldsymbol{o}|\Theta_1, \Theta_2) = f_{\Theta_1}(\boldsymbol{o}) + f_{\Theta_2}(\boldsymbol{o}),$$

where f_{Θ_1} and f_{Θ_2} are the probability density functions of Θ_1 and Θ_2, respectively. We can use the *expectation-maximization* (EM) algorithm (Chapter 11) to learn the parameters $\mu_1, \sigma_1, \mu_2, \sigma_2$ from the data, as we do in mixture models for clustering. Each cluster is represented by a learned normal distribution. An object, $\boldsymbol{o}$, is detected as an outlier if it does not belong to any cluster, that is, the probability is very low that it was generated by the combination of the two distributions.

Example 12.12 **Multivariate outlier detection using multiple clusters.** Most of the data objects shown in Figure 12.4 are in either C_1 or C_2. Other objects, representing noise, are uniformly distributed in the data space. A small cluster, C_3, is highly suspicious because it is not close to either of the two major clusters, C_1 and C_2. The objects in C_3 should therefore be detected as outliers.

Note that identifying the objects in C_3 as outliers is difficult, whether or not we assume that the given data follow a normal distribution or a mixture of multiple distributions. This is because the probability of the objects in C_3 will be higher than some of the noise objects, like $\boldsymbol{o}$ in Figure 12.4, due to a higher local density in C_3. ■

To tackle the problem demonstrated in Example 12.12, we can assume that the normal data objects are generated by a normal distribution, or a mixture of normal distributions, whereas the outliers are generated by another distribution. Heuristically, we can add constraints on the distribution that is generating outliers. For example, it is reasonable to assume that this distribution has a larger variance if the outliers are distributed in a larger area. Technically, we can assign $\sigma_{outlier} = k\sigma$, where k is a user-specified parameter and σ is the standard deviation of the normal distribution generating the normal data. Again, the EM algorithm can be used to learn the parameters.

12.3.2 Nonparametric Methods

In nonparametric methods for outlier detection, the model of "normal data" is learned from the input data, rather than assuming one a priori. Nonparametric methods often make fewer assumptions about the data, and thus can be applicable in more scenarios.

Example 12.13 Outlier detection using a histogram. *AllElectronics* records the purchase amount for every customer transaction. Figure 12.5 uses a histogram (refer to Chapters 2 and 3) to graph these amounts as percentages, given all transactions. For example, 60% of the transaction amounts are between \$0.00 and \$1000.

We can use the histogram as a nonparametric statistical model to capture outliers. For example, a transaction in the amount of \$7500 can be regarded as an outlier because only $1 - (60\% + 20\% + 10\% + 6.7\% + 3.1\%) = 0.2\%$ of transactions have an amount higher than \$5000. On the other hand, a transaction amount of \$385 can be treated as normal because it falls into the bin (or bucket) holding 60% of the transactions.

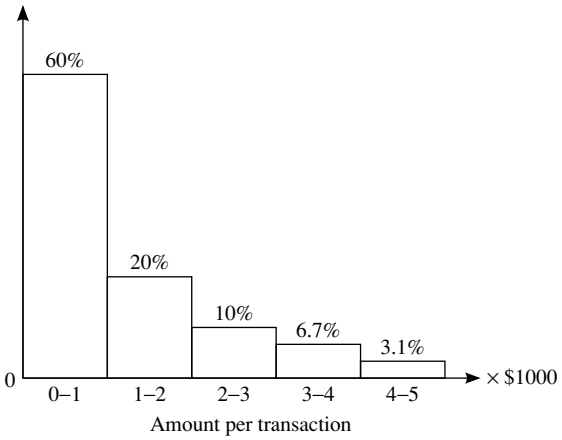

Figure 12.5 Histogram of purchase amounts in transactions.

■

As illustrated in the previous example, the histogram is a frequently used nonparametric statistical model that can be used to detect outliers. The procedure involves the following two steps.

Step 1: Histogram construction. In this step, we construct a histogram using the input data (training data). The histogram may be univariate as in Example 12.13, or multivariate if the input data are multidimensional.

Note that although nonparametric methods do not assume any a priori statistical model, they often do require user-specified parameters to learn models from data. For example, to construct a good histogram, a user has to specify the type of histogram (e.g., equal width or equal depth) and other parameters (e.g., the number of bins in the histogram or the size of each bin). Unlike parametric methods, these parameters do not specify types of data distribution (e.g., Gaussian).

Step 2: Outlier detection. To determine whether an object, $\boldsymbol{o}$, is an outlier, we can check it against the histogram. In the simplest approach, if the object falls in one of the histogram's bins, the object is regarded as normal. Otherwise, it is considered an outlier.

For a more sophisticated approach, we can use the histogram to assign an outlier score to the object. In Example 12.13, we can let an object's outlier score be the inverse of the volume of the bin in which the object falls. For example, the outlier score for a transaction amount of \$7500 is $\frac{1}{0.2\%} = 500$, and that for a transaction amount of \$385 is $\frac{1}{60\%} = 1.67$. The scores indicate that the transaction amount of \$7500 is much more likely to be an outlier than that of \$385.

A drawback to using histograms as a nonparametric model for outlier detection is that it is hard to choose an appropriate bin size. On the one hand, if the bin size is set too small, many normal objects may end up in empty or rare bins, and thus be misidentified as outliers. This leads to a high false positive rate and low precision. On the other hand, if the bin size is set too high, outlier objects may infiltrate into some frequent bins and thus be "disguised" as normal. This leads to a high false negative rate and low recall.

To overcome this problem, we can adopt kernel density estimation to estimate the probability density distribution of the data. We treat an observed object as an indicator of high probability density in the surrounding region. The probability density at a point depends on the distances from this point to the observed objects. We use a *kernel function* to model the influence of a sample point within its neighborhood. A kernel $K()$ is a non-negative real-valued integrable function that satisfies the following two conditions:

- $\int_{-\infty}^{+\infty} K(u)du = 1$.
- $K(-u) = K(u)$ for all values of u.

A frequently used kernel is a standard Gaussian function with mean 0 and variance 1:

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}. \tag{12.8}$$

Let $x_1, \ldots, x_n$ be an independent and identically distributed sample of a random variable f. The kernel density approximation of the probability density function is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{12.9}$$

where $K()$ is a kernel and h is the bandwidth serving as a smoothing parameter.

Once the probability density function of a data set is approximated through kernel density estimation, we can use the estimated density function $\hat{f}$ to detect outliers. For an object, o, $\hat{f}(o)$ gives the estimated probability that the object is generated by the stochastic process. If $\hat{f}(o)$ is high, then the object is likely normal. Otherwise, o is likely an outlier. This step is often similar to the corresponding step in parametric methods.

In summary, statistical methods for outlier detection learn models from data to distinguish normal data objects from outliers. An advantage of using statistical methods is that the outlier detection may be statistically justifiable. Of course, this is true only if the statistical assumption made about the underlying data meets the constraints in reality.

The data distribution of high-dimensional data is often complicated and hard to fully understand. Consequently, statistical methods for outlier detection on high-dimensional data remain a big challenge. Outlier detection for high-dimensional data is further addressed in Section 12.8.

The computational cost of statistical methods depends on the models. When simple parametric models are used (e.g., a Gaussian), fitting the parameters typically takes linear time. When more sophisticated models are used (e.g., mixture models, where the EM algorithm is used in learning), approximating the best parameter values often takes several iterations. Each iteration, however, is typically linear with respect to the data set's size. For kernel density estimation, the model learning cost can be up to quadratic. Once the model is learned, the outlier detection cost is often very small per object.

12.4 Proximity-Based Approaches

Given a set of objects in feature space, a distance measure can be used to quantify the similarity between objects. Intuitively, objects that are far from others can be regarded as outliers. Proximity-based approaches assume that the proximity of an outlier object to its nearest neighbors significantly deviates from the proximity of the object to most of the other objects in the data set.

There are two types of proximity-based outlier detection methods: distance-based and density-based methods. A *distance-based outlier detection method* consults the **neighborhood** of an object, which is defined by a given radius. An object is then considered an outlier if its neighborhood does not have enough other points. A *density-based outlier detection method* investigates the density of an object and that of its neighbors. Here, an object is identified as an outlier if its density is relatively much lower than that of its neighbors.

Let's start with distance-based outliers.

12.4.1 Distance-Based Outlier Detection and a Nested Loop Method

A representative method of proximity-based outlier detection uses the concept of **distance-based outliers**. For a set, D, of data objects to be analyzed, a user can specify a distance threshold, r, to define a reasonable neighborhood of an object. For each object, $\boldsymbol{o}$, we can examine the number of other objects in the r-neighborhood of $\boldsymbol{o}$. If most of the objects in D are far from $\boldsymbol{o}$, that is, not in the r-neighborhood of $\boldsymbol{o}$, then $\boldsymbol{o}$ can be regarded as an outlier.

Formally, let r $(r \geq 0)$ be a *distance threshold* and π $(0 < \pi \leq 1)$ be a fraction threshold. An object, $\boldsymbol{o}$, is a $DB(r,\pi)$-**outlier** if

$$\frac{\|\{\boldsymbol{o}' | dist(\boldsymbol{o},\boldsymbol{o}') \leq r\}\|}{\|D\|} \leq \pi, \tag{12.10}$$

where $dist(\cdot,\cdot)$ is a distance measure.

Equivalently, we can determine whether an object, $\boldsymbol{o}$, is a $DB(r,\pi)$-outlier by checking the distance between $\boldsymbol{o}$ and its k-nearest neighbor, $\boldsymbol{o}_k$, where $k = \lceil \pi \|D\| \rceil$. Object $\boldsymbol{o}$ is an outlier if $dist(\boldsymbol{o},\boldsymbol{o}_k) > r$, because in such a case, there are fewer than k objects except for $\boldsymbol{o}$ that are in the r-neighborhood of $\boldsymbol{o}$.

"*How can we compute $DB(r,\pi)$-outliers?*" A straightforward approach is to use nested loops to check the r-neighborhood for every object, as shown in Figure 12.6. For any object, $\boldsymbol{o}_i$ $(1 \leq i \leq n)$, we calculate the distance between $\boldsymbol{o}_i$ and the other object, and count the number of other objects in the r-neighborhood of $\boldsymbol{o}_i$. Once we find $\pi \cdot n$ other

Algorithm: Distance-based outlier detection.

Input:

- a set of objects $D = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n\}$, threshold r $(r > 0)$ and π $(0 < \pi \leq 1)$;

Output: $DB(r,\pi)$ outliers in D.

Method:

```
for i = 1 to n do
    count ← 0
    for j = 1 to n do
        if i ≠ j and dist(o_i, o_j) ≤ r then
            count ← count + 1
            if count ≥ π · n then
                exit {o_i cannot be a DB(r,π) outlier}
            endif
        endif
    endfor
    print o_i {o_i is a DB(r,π) outlier according to (Eq. 12.10)}
endfor;
```

Figure 12.6 Nested loop algorithm for $DB(r,\pi)$-outlier detection.

objects within a distance r from $\boldsymbol{o}_i$, the inner loop can be terminated because $\boldsymbol{o}_i$ already violates (Eq. 12.10), and thus is not a $DB(r,\pi)$-outlier. On the other hand, if the inner loop completes for $\boldsymbol{o}_i$, this means that $\boldsymbol{o}_i$ has less than $\pi \cdot n$ neighbors in a radius of r, and thus is a $DB(r,\pi)$-outlier.

The straightforward nested loop approach takes $O(n^2)$ time. Surprisingly, the actual CPU runtime is often linear with respect to the data set size. For most nonoutlier objects, the inner loop terminates early when the number of outliers in the data set is small, which should be the case most of the time. Correspondingly, only a small fraction of the data set is examined.

When mining large data sets where the complete set of objects cannot be held in main memory, the nested loop approach is still costly. Suppose the main memory has m pages for the mining. Instead of conducting the inner loop object by object, in such a case, the outer loop uses $m-1$ pages to hold as many objects as possible and uses the remaining one page to run the inner loop. The inner loop cannot stop until all objects in the $m-1$ pages are identified as not being outliers, which is very unlikely to happen. Correspondingly, it is likely that the algorithm has to incur $O((\frac{n}{b})^2)$ input/output (I/O) cost, where b is the number of objects that can be held in one page.

The major cost in the nested loop method comes from two aspects. First, to check whether an object is an outlier, the nested loop method tests the object against the whole data set. To improve, we need to explore how to determine the outlierness of an object from the neighbors that are close to the object. Second, the nested loop method checks objects one by one. To improve, we should try to group objects according to their proximity, and check the outlierness of objects group by group most of the time. Section 12.4.2 introduces how to implement the preceding ideas.

12.4.2 A Grid-Based Method

CELL is a grid-based method for distance-based outlier detection. In this method, the data space is partitioned into a multidimensional grid, where each cell is a hypercube that has a diagonal of length $\frac{r}{2}$, where r is a distance threshold parameter. In other words, if there are l dimensions, the length of each edge of a cell is $\frac{r}{2\sqrt{l}}$.

Consider a 2-D data set, for example. Figure 12.7 shows part of the grid. The length of each edge of a cell is $\frac{r}{2\sqrt{2}}$.

Consider the cell C in Figure 12.7. The neighboring cells of C can be divided into two groups. The cells immediately next to C constitute the *level-1* cells (labeled "1" in the figure), and the cells one or two cells away from C in any direction constitute the *level-2* cells (labeled "2" in the figure). The two levels of cells have the following properties:

- **Level-1 cell property**: Given any possible point, $\boldsymbol{x}$ of C, and any possible point, $\boldsymbol{y}$, in a level-1 cell, then $dist(\boldsymbol{x},\boldsymbol{y}) \leq r$.
- **Level-2 cell property**: Given any possible point, $\boldsymbol{x}$ of C, and any point, $\boldsymbol{y}$, such that $dist(\boldsymbol{x},\boldsymbol{y}) \geq r$, then $\boldsymbol{y}$ is in a level-2 cell.

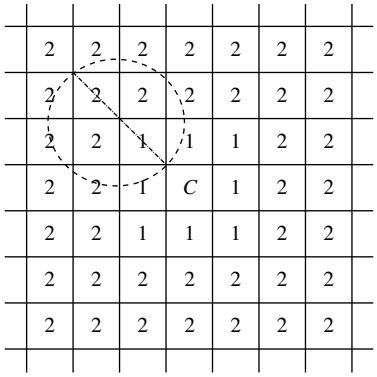

Figure 12.7 Grids in the CELL method.

Let a be the number of objects in cell C, b_1 be the total number of objects in the level-1 cells, and b_2 be the total number of objects in the level-2 cells. We can apply the following rules.

- **Level-1 cell pruning rule**: Based on the level-1 cell property, if $a + b_1 > \lceil \pi n \rceil$, then every object $\boldsymbol{o}$ in C is not a $DB(r,\pi)$-outlier because all those objects in C and the level-1 cells are in the r-neighborhood of $\boldsymbol{o}$, and there are at least $\lceil \pi n \rceil$ such neighbors.
- **Level-2 cell pruning rule**: Based on the level-2 cell property, if $a + b_1 + b_2 < \lceil \pi n \rceil + 1$, then all objects in C are $DB(r,\pi)$-outliers because each of their r-neighborhoods has less than $\lceil \pi n \rceil$ other objects.

Using the preceding two rules, the CELL method organizes objects into groups using a grid—all objects in a cell form a group. For groups satisfying one of the two rules, we can determine that either all objects in a cell are outliers or nonoutliers, and thus do not need to check those objects one by one. Moreover, to apply the two rules, we need only check a limited number of cells close to a target cell instead of the whole data set.

Using the previous two rules, many objects can be determined as being either nonoutliers or outliers. We only need to check the objects that cannot be pruned using the two rules. Even for such an object, $\boldsymbol{o}$, we need only compute the distance between $\boldsymbol{o}$ and the objects in the level-2 cells with respect to $\boldsymbol{o}$. This is because all objects in the level-1 cells have a distance of at most r to $\boldsymbol{o}$, and all objects not in a level-1 or level-2 cell must have a distance of more than r from $\boldsymbol{o}$, and thus cannot be in the r-neighborhood of $\boldsymbol{o}$.

When the data set is very large so that most of the data are stored on disk, the CELL method may incur many random accesses to disk, which is costly. An alternative method was proposed, which uses a very small amount of main memory (around 1% of the data