Fuzzy set theory is useful for data mining systems performing rule-based classification. It provides operations for combining fuzzy measurements. Suppose that in addition to the fuzzy sets for *income*, we defined the fuzzy sets *junior_employee* and *senior_employee* for the attribute *years_employed*. Suppose also that we have a rule that, say, tests *high_income* and *senior_employee* in the rule antecedent (IF part) for a given employee, x. If these two fuzzy measures are ANDed together, the minimum of their measure is taken as the measure of the rule. In other words,

$$m_{(high\_income\ AND\ senior\_employee)}(x) = min(m_{high\_income}(x), m_{senior\_employee}(x)).$$

This is akin to saying that a chain is as strong as its weakest link. If the two measures are ORed, the maximum of their measure is taken as the measure of the rule. In other words,

$$m_{(high\_income\ OR\ senior\_employee)}(x) = max(m_{high\_income}(x), m_{senior\_employee}(x)).$$

Intuitively, this is like saying that a rope is as strong as its strongest strand.

Given a tuple to classify, more than one fuzzy rule may apply. Each applicable rule contributes a vote for membership in the categories. Typically, the truth values for each predicted category are summed, and these sums are combined. Several procedures exist for translating the resulting fuzzy output into a *defuzzified* or crisp value that is returned by the system.

Fuzzy logic systems have been used in numerous areas for classification, including market research, finance, health care, and environmental engineering.

6.11 Prediction

"*What if we would like to predict a continuous value, rather than a categorical label?*" Numeric prediction is the task of predicting continuous (or ordered) values for given input. For example, we may wish to predict the salary of college graduates with 10 years of work experience, or the potential sales of a new product given its price. By far, the most widely used approach for numeric prediction (hereafter referred to as prediction) is **regression**, a statistical methodology that was developed by Sir Frances Galton (1822–1911), a mathematician who was also a cousin of Charles Darwin. In fact, many texts use the terms "regression" and "numeric prediction" synonymously. However, as we have seen, some classification techniques (such as backpropagation, support vector machines, and k-nearest-neighbor classifiers) can be adapted for prediction. In this section, we discuss the use of regression techniques for prediction.

Regression analysis can be used to model the relationship between one or more *independent* or **predictor** variables and a *dependent* or **response** variable (which is continuous-valued). In the context of data mining, the predictor variables are the attributes of interest describing the tuple (i.e., making up the attribute vector). In general, the values of the predictor variables are known. (Techniques exist for handling cases where such values may be missing.) The response variable is what we want to predict—it is what we referred to in Section 6.1 as the predicted attribute. Given a tuple described by predictor variables, we want to predict the associated value of the response variable.

Regression analysis is a good choice when all of the predictor variables are continuous-valued as well. Many problems can be solved by *linear regression*, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to a linear one. For reasons of space, we cannot give a fully detailed treatment of regression. Instead, this section provides an intuitive introduction to the topic. Section 6.11.1 discusses straight-line regression analysis (which involves a single predictor variable) and multiple linear regression analysis (which involves two or more predictor variables). Section 6.11.2 provides some pointers on dealing with nonlinear regression. Section 6.11.3 mentions other regression-based methods, such as generalized linear models, Poisson regression, log-linear models, and regression trees.

Several software packages exist to solve regression problems. Examples include SAS (*www.sas.com*), SPSS (*www.spss.com*), and S-Plus (*www.insightful.com*). Another useful resource is the book *Numerical Recipes in C*, by Press, Flannery, Teukolsky, and Vetterling, and its associated source code.

6.11.1 Linear Regression

Straight-line regression analysis involves a response variable, y, and a single predictor variable, x. It is the simplest form of regression, and models y as a linear function of x. That is,

$$y = b + wx, \tag{6.48}$$

where the variance of y is assumed to be constant, and b and w are **regression coefficients** specifying the Y-intercept and slope of the line, respectively. The regression coefficients, w and b, can also be thought of as weights, so that we can equivalently write,

$$y = w_0 + w_1 x. \tag{6.49}$$

These coefficients can be solved for by the **method of least squares**, which estimates the best-fitting straight line as the one that minimizes the error between the actual data and the estimate of the line. Let D be a training set consisting of values of predictor variable, x, for some population and their associated values for response variable, y. The training set contains $|D|$ data points of the form $(x_1, y_1), (x_2, y_2), \ldots, (x_{|D|}, y_{|D|})$.[12] The regression coefficients can be estimated using this method with the following equations:

$$w_1 = \frac{\sum_{i=1}^{|D|}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|}(x_i - \bar{x})^2} \tag{6.50}$$

[12]Note that earlier, we had used the notation $(\boldsymbol{X_i}, y_i)$ to refer to training tuple i having associated class label y_i, where $\boldsymbol{X_i}$ was an attribute (or feature) *vector* (that is, $\boldsymbol{X_i}$ was described by more than one attribute). Here, however, we are dealing with just one predictor variable. Since the $\boldsymbol{X_i}$ here are one-dimensional, we use the notation x_i over $\boldsymbol{X_i}$ in this case.

$$w_0 = \bar{y} - w_1 \bar{x} \tag{6.51}$$

where $\bar{x}$ is the mean value of $x_1, x_2, \ldots, x_{|D|}$, and $\bar{y}$ is the mean value of $y_1, y_2, \ldots, y_{|D|}$. The coefficients w_0 and w_1 often provide good approximations to otherwise complicated regression equations.

Example 6.11 **Straight-line regression using the method of least squares.** Table 6.7 shows a set of paired data where x is the number of years of work experience of a college graduate and y is the

Table 6.7 Salary data.

x years experience	*y salary (in $1000s)*
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

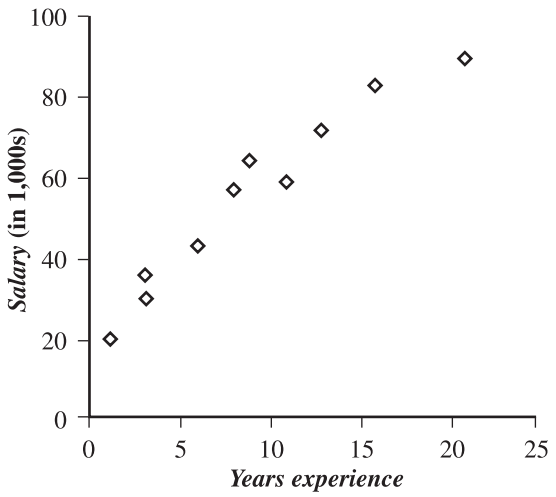


Figure 6.26 Plot of the data in Table 6.7 for Example 6.11. Although the points do not fall on a straight line, the overall pattern suggests a linear relationship between x (*years experience*) and y (*salary*).

corresponding salary of the graduate. The 2-D data can be graphed on a *scatter plot*, as in Figure 6.26. The plot suggests a linear relationship between the two variables, x and y. We model the relationship that salary may be related to the number of years of work experience with the equation $y = w_0 + w_1 x$.

Given the above data, we compute $\bar{x} = 9.1$ and $\bar{y} = 55.4$. Substituting these values into Equations (6.50) and (6.51), we get

$$w_1 = \frac{(3-9.1)(30-55.4)+(8-9.1)(57-55.4)+\cdots+(16-9.1)(83-55.4)}{(3-9.1)^2+(8-9.1)^2+\cdots+(16-9.1)^2} = 3.5$$

$$w_0 = 55.4-(3.5)(9.1) = 23.6$$

Thus, the equation of the least squares line is estimated by $y = 23.6 + 3.5x$. Using this equation, we can predict that the salary of a college graduate with, say, 10 years of experience is \$58,600. ■

Multiple linear regression is an extension of straight-line regression so as to involve more than one predictor variable. It allows response variable y to be modeled as a linear function of, say, n predictor variables or attributes, $A_1, A_2, \ldots, A_n$, describing a tuple, $\boldsymbol{X}$. (That is, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$.) Our training data set, D, contains data of the form $(\boldsymbol{X_1}, y_1)$, $(\boldsymbol{X_2}, y_2)$, ... , $(\boldsymbol{X}_{|\boldsymbol{D}|}, y_{|D|})$, where the $\boldsymbol{X_i}$ are the n-dimensional training tuples with associated class labels, y_i. An example of a multiple linear regression model based on two predictor attributes or variables, A_1 and A_2, is

$$y = w_0 + w_1 x_1 + w_2 x_2, \tag{6.52}$$

where x_1 and x_2 are the values of attributes A_1 and A_2, respectively, in $\boldsymbol{X}$. The method of least squares shown above can be extended to solve for w_0, w_1, and w_2. The equations, however, become long and are tedious to solve by hand. Multiple regression problems are instead commonly solved with the use of statistical software packages, such as SAS, SPSS, and S-Plus (see references above.)

6.11.2 Nonlinear Regression

"How can we model data that does not show a linear dependence? For example, what if a given response variable and predictor variable have a relationship that may be modeled by a polynomial function?" Think back to the straight-line linear regression case above where dependent response variable, y, is modeled as a linear function of a single independent predictor variable, x. What if we can get a more accurate model using a nonlinear model, such as a parabola or some other higher-order polynomial? **Polynomial regression** is often of interest when there is just one predictor variable. It can be modeled by adding polynomial terms to the basic linear model. By applying transformations to the variables, we can convert the nonlinear model into a linear one that can then be solved by the method of least squares.

Example 6.12 Transformation of a polynomial regression model to a linear regression model. Consider a cubic polynomial relationship given by

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3. \tag{6.53}$$

To convert this equation to linear form, we define new variables:

$$x_1 = x \qquad x_2 = x^2 \qquad x_3 = x^3 \tag{6.54}$$

Equation (6.53) can then be converted to linear form by applying the above assignments, resulting in the equation $y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$, which is easily solved by the method of least squares using software for regression analysis. Note that polynomial regression is a special case of multiple regression. That is, the addition of high-order terms like x^2, x^3, and so on, which are simple functions of the single variable, x, can be considered equivalent to adding new independent variables. ■

In Exercise 15, you are asked to find the transformations required to convert a nonlinear model involving a power function into a linear regression model.

Some models are intractably nonlinear (such as the sum of exponential terms, for example) and cannot be converted to a linear model. For such cases, it may be possible to obtain least square estimates through extensive calculations on more complex formulae.

Various statistical measures exist for determining how well the proposed model can predict y. These are described in Section 6.12.2. Obviously, the greater the number of predictor attributes is, the slower the performance is. Before applying regression analysis, it is common to perform attribute subset selection (Section 2.5.2) to eliminate attributes that are unlikely to be good predictors for y. In general, regression analysis is accurate for prediction, except when the data contain outliers. Outliers are data points that are highly inconsistent with the remaining data (e.g., they may be way out of the expected value range). Outlier detection is discussed in Chapter 7. Such techniques must be used with caution, however, so as not to remove data points that are valid, although they may vary greatly from the mean.

6.11.3 Other Regression-Based Methods

Linear regression is used to model continuous-valued functions. It is widely used, owing largely to its simplicity. *"Can it also be used to predict categorical labels?"* **Generalized linear models** represent the theoretical foundation on which linear regression can be applied to the modeling of categorical response variables. In generalized linear models, the variance of the response variable, y, is a function of the mean value of y, unlike in linear regression, where the variance of y is constant. Common types of generalized linear models include **logistic regression** and **Poisson regression**. Logistic regression models the probability of some event occurring as a linear function of a set of predictor variables. Count data frequently exhibit a Poisson distribution and are commonly modeled using Poisson regression.

Log-linear models approximate *discrete* multidimensional probability distributions. They may be used to estimate the probability value associated with data cube cells. For example, suppose we are given data for the attributes *city, item, year*, and *sales*. In the log-linear method, all attributes must be categorical; hence continuous-valued attributes (like *sales*) must first be discretized. The method can then be used to estimate the probability of each cell in the 4-D base cuboid for the given attributes, based on the 2-D cuboids for *city* and *item*, *city* and *year*, *city* and *sales*, and the 3-D cuboid for *item, year*, and *sales*. In this way, an iterative technique can be used to build higher-order data cubes from lower-order ones. The technique scales up well to allow for many dimensions. Aside from prediction, the log-linear model is useful for data compression (since the smaller-order cuboids together typically occupy less space than the base cuboid) and data smoothing (since cell estimates in the smaller-order cuboids are less subject to sampling variations than cell estimates in the base cuboid).

Decision tree induction can be adapted so as to predict continuous (ordered) values, rather than class labels. There are two main types of trees for prediction—*regression trees* and *model trees*. **Regression trees** were proposed as a component of the CART learning system. (Recall that the acronym CART stands for *Classification and Regression Trees*.) Each regression tree leaf stores a continuous-valued prediction, which is actually the average value of the predicted attribute for the training tuples that reach the leaf. Since the terms "regression" and "numeric prediction" are used synonymously in statistics, the resulting trees were called "regression trees," even though they did not use any regression equations. By contrast, in **model trees**, each leaf holds a regression model—a multivariate linear equation for the predicted attribute. Regression and model trees tend to be more accurate than linear regression when the data are not represented well by a simple linear model.

6.12 Accuracy and Error Measures

Now that you may have trained a classifier or predictor, there may be many questions going through your mind. For example, suppose you used data from previous sales to train a classifier to predict customer purchasing behavior. You would like an estimate of how accurately the classifier can predict the purchasing behavior of future customers, that is, future customer data on which the classifier has not been trained. You may even have tried different methods to build more than one classifier (or predictor) and now wish to compare their accuracy. But what is accuracy? How can we estimate it? Are there strategies for increasing the accuracy of a learned model? These questions are addressed in the next few sections. Section 6.12.1 describes measures for computing classifier accuracy. Predictor error measures are given in Section 6.12.2. We can use these measures in techniques for accuracy estimation, such as the *holdout, random subsampling*, *k-fold cross-validation*, and *bootstrap* methods (Section 6.13). In Section 6.14, we'll learn some tricks for increasing model accuracy, such as *bagging* and *boosting*. Finally, Section 6.15 discusses **model selection** (i.e., choosing one classifier or predictor over another).

7 Cluster Analysis

Imagine that you are given a set of data objects for analysis where, unlike in classification, the class label of each object is not known. This is quite common in large databases, because assigning class labels to a large number of objects can be a very costly process. *Clustering* is the process of grouping the data into classes or *clusters*, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters. Dissimilarities are assessed based on the attribute values describing the objects. Often, distance measures are used. Clustering has its roots in many areas, including data mining, statistics, biology, and machine learning.

In this chapter, we study the requirements of clustering methods for large amounts of data. We explain how to compute dissimilarities between objects represented by various attribute or variable types. We examine several clustering techniques, organized into the following categories: *partitioning methods, hierarchical methods, density-based methods*, *grid-based methods*, *model-based methods*, *methods for high-dimensional data* (such as *frequent pattern–based methods*), and *constraint-based clustering*. Clustering can also be used for *outlier detection*, which forms the final topic of this chapter.

7.1 What Is Cluster Analysis?

The process of grouping a set of physical or abstract objects into classes of *similar* objects is called **clustering**. 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. A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering-based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Cluster analysis is an important human activity. Early in childhood, we learn how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious clustering schemes. By automated clustering, we can identify dense and sparse regions in object space and, therefore, discover overall distribution patterns and interesting correlations among data attributes. Cluster analysis has been widely used in numerous applications, including market research, pattern recognition, data analysis, and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value, and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used to help classify documents on the Web for information discovery.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their *similarity*. Clustering can also be used for **outlier detection**, where outliers (values that are "far away" from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on the detected clusters and the selected attributes or features.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning, spatial database technology, biology, and marketing. Owing to the huge amounts of data collected in databases, cluster analysis has recently become a highly active topic in data mining research.

As a branch of statistics, cluster analysis has been extensively studied for many years, focusing mainly on *distance-based cluster analysis*. Cluster analysis tools based on *k*-means, *k*-medoids, and several other methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS. In machine learning, clustering is an example of **unsupervised learning**. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and class-labeled training examples. For this reason, clustering is a form of **learning by observation**, rather than *learning by examples*. In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in *large databases*. Active themes of research focus on the *scalability* of clustering methods, the effectiveness of methods for clustering *complex shapes and types of data*, *high-dimensional* clustering techniques, and methods for clustering *mixed numerical and categorical data* in large databases.

Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining:

- **Scalability**: Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a *sample* of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.
- **Ability to deal with different types of attributes**: Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.
- **Discovery of clusters with arbitrary shape**: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.
- **Minimal requirements for domain knowledge to determine input parameters**: Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.
- **Ability to deal with noisy data**: Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.
- **Incremental clustering and insensitivity to the order of input records**: Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clusterings depending on the order of presentation of the input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.
- **High dimensionality**: A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high-dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

- **Constraint-based clustering**: Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.
- **Interpretability and usability**: Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

With these requirements in mind, our study of cluster analysis proceeds as follows. First, we study different types of data and how they can influence clustering methods. Second, we present a general categorization of clustering methods. We then study each clustering method in detail, including partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. We also examine clustering in high-dimensional space, constraint-based clustering, and outlier analysis.

7.2 Types of Data in Cluster Analysis

In this section, we study the types of data that often occur in cluster analysis and how to preprocess them for such an analysis. Suppose that a data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on. Main memory-based clustering algorithms typically operate on either of the following two data structures.

- **Data matrix** (or *object-by-variable structure*): This represents n objects, such as persons, with p **variables** (also called *measurements* or *attributes*), such as age, height, weight, gender, and so on. The structure is in the form of a relational table, or n-by-p matrix (n objects $\times p$ variables):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix} \tag{7.1}$$

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

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix} \tag{7.2}$$

where $d(i, j)$ is the measured **difference** or **dissimilarity** between objects i and j. In general, $d(i, j)$ is a nonnegative number that is close to 0 when objects i and j are highly similar or "near" each other, and becomes larger the more they differ. Since $d(i, j) = d(j, i)$, and $d(i, i) = 0$, we have the matrix in (7.2). Measures of dissimilarity are discussed throughout this section.

The rows and columns of the data matrix represent different entities, while those of the dissimilarity matrix represent the same entity. Thus, the data matrix is often called a **two-mode** matrix, whereas the dissimilarity matrix is called a **one-mode** matrix. Many clustering algorithms operate on a dissimilarity matrix. If the data are presented in the form of a data matrix, it can first be transformed into a dissimilarity matrix before applying such clustering algorithms.

In this section, we discuss how object dissimilarity can be computed for objects described by *interval-scaled* variables; by *binary* variables; by *categorical, ordinal*, and *ratio-scaled* variables; or combinations of these variable types. Nonmetric similarity between complex objects (such as documents) is also described. The dissimilarity data can later be used to compute clusters of objects.

7.2.1 Interval-Scaled Variables

This section discusses *interval-scaled variables* and their standardization. It then describes distance measures that are commonly used for computing the dissimilarity of objects described by such variables. These measures include the *Euclidean, Manhattan*, and *Minkowski distances*.

"*What are interval-scaled variables?*" **Interval-scaled variables** are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature.

The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. In general, expressing a variable in smaller units will lead to a larger range for that variable, and thus a larger effect on the resulting clustering structure. To help avoid dependence on the choice of measurement units, the data should be standardized. Standardizing measurements attempts to give all variables an equal weight. This is particularly useful when given no prior knowledge of the data. However, in some applications, users may intentionally want to give more

weight to a certain set of variables than to others. For example, when clustering basketball player candidates, we may prefer to give more weight to the variable height.

"*How can the data for a variable be standardized?*" To standardize measurements, one choice is to convert the original measurements to unitless variables. Given measurements for a variable f, this can be performed as follows.

1. Calculate the **mean absolute deviation**, s_f:

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \cdots + |x_{nf} - m_f|), \tag{7.3}$$

   where $x_{1f}, \ldots, x_{nf}$ are n measurements of f, and m_f is the *mean* value of f, that is, $m_f = \frac{1}{n}(x_{1f} + x_{2f} + \cdots + x_{nf})$.

2. Calculate the **standardized measurement**, or **z-score**:

$$z_{if} = \frac{x_{if} - m_f}{s_f}. \tag{7.4}$$

   The mean absolute deviation, s_f, is more robust to outliers than the standard deviation, σ_f. When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_{if} - m_f|$) are not squared; hence, the effect of outliers is somewhat reduced. There are more robust measures of dispersion, such as the *median absolute deviation*. However, the advantage of using the mean absolute deviation is that the z-scores of outliers do not become too small; hence, the outliers remain detectable.

Standardization may or may not be useful in a particular application. Thus the choice of whether and how to perform standardization should be left to the user. Methods of standardization are also discussed in Chapter 2 under normalization techniques for data preprocessing.

After standardization, or without standardization in certain applications, the dissimilarity (or similarity) between the objects described by interval-scaled variables is typically computed based on the distance between each pair of objects. The most popular distance measure is **Euclidean distance**, which is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2}, \tag{7.5}$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{in})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ are two n-dimensional data objects.

Another well-known metric is **Manhattan (or city block) distance**, defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{in} - x_{jn}|. \tag{7.6}$$

Both the Euclidean distance and Manhattan distance satisfy the following mathematic requirements of a distance function:

1. $d(i, j) \geq 0$: Distance is a nonnegative number.
2. $d(i, i) = 0$: The distance of an object to itself is 0.
3. $d(i, j) = d(j, i)$: Distance is a symmetric function.
4. $d(i, j) \leq d(i, h) + d(h, j)$: Going directly from object i to object j in space is no more than making a detour over any other object h (*triangular inequality*).

Example 7.1 **Euclidean distance and Manhattan distance.** Let $\boldsymbol{x_1} = (1, 2)$ and $\boldsymbol{x_2} = (3, 5)$ represent two objects as in Figure 7.1. The Euclidean distance between the two is $\sqrt{(2^2 + 3^2)} = 3.61$. The Manhattan distance between the two is $2 + 3 = 5$. ■

Minkowski distance is a generalization of both Euclidean distance and Manhattan distance. It is defined as

$$d(i, j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \cdots + |x_{in} - x_{jn}|^p)^{1/p}, \tag{7.7}$$

where p is a positive integer. Such a distance is also called L_p norm, in some literature. It represents the Manhattan distance when $p = 1$ (i.e., L_1 norm) and Euclidean distance when $p = 2$ (i.e., L_2 norm).

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

$$d(i, j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_m|x_{in} - x_{jn}|^2}. \tag{7.8}$$

Weighting can also be applied to the Manhattan and Minkowski distances.

7.2.2 Binary Variables

Let us see how to compute the dissimilarity between objects described by either *symmetric* or *asymmetric binary variables*.

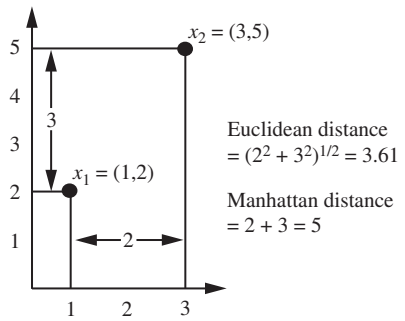


Figure 7.1 Euclidean and Manhattan distances between two objects.

A **binary variable** has only two states: 0 or 1, where 0 means that the variable is absent, and 1 means that it is present. Given the variable *smoker* describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Treating binary variables as if they are interval-scaled can lead to misleading clustering results. Therefore, methods specific to binary data are necessary for computing dissimilarities.

"*So, how can we compute the dissimilarity between two binary variables?*" One approach involves computing a dissimilarity matrix from the given binary data. If all binary variables are thought of as having the same weight, we have the 2-by-2 contingency table of Table 7.1, where q is the number of variables that equal 1 for both objects i and j, r is the number of variables that equal 1 for object i but that are 0 for object j, s is the number of variables that equal 0 for object i but equal 1 for object j, and t is the number of variables that equal 0 for both objects i and j. The total number of variables is p, where $p = q + r + s + t$.

"*What is the difference between symmetric and asymmetric binary variables?*" A binary variable is **symmetric** if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute *gender* having the states *male* and *female*. Dissimilarity that is based on symmetric binary variables is called **symmetric binary dissimilarity**. Its dissimilarity (or distance) measure, defined in Equation (7.9), can be used to assess the dissimilarity between objects i and j.

$$d(i, j) = \frac{r + s}{q + r + s + t}. \tag{7.9}$$

A binary variable is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a disease *test*. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 (e.g., *HIV positive*) and the other by 0 (e.g., *HIV negative*). Given two asymmetric binary variables, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary variables are often considered "monary" (as if having one state). The dissimilarity based on such variables is called **asymmetric binary dissimilarity**, where the number of negative

Table 7.1 A contingency table for binary variables.

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

matches, t, is considered unimportant and thus is ignored in the computation, as shown in Equation (7.10).

$$d(i, j) = \frac{r+s}{q+r+s}. \tag{7.10}$$

Complementarily, we can measure the distance between two binary variables based on the notion of *similarity* instead of *dissimilarity*. For example, the **asymmetric binary similarity** between the objects i and j, or $sim(i, j)$, can be computed as,

$$sim(i, j) = \frac{q}{q+r+s} = 1 - d(i, j). \tag{7.11}$$

The coefficient $sim(i, j)$ is called the **Jaccard coefficient**, which is popularly referenced in the literature.

When both symmetric and asymmetric binary variables occur in the same data set, the mixed variables approach described in Section 7.2.4 can be applied.

Example 7.2 **Dissimilarity between binary variables.** Suppose that a patient record table (Table 7.2) contains the attributes *name, gender, fever, cough, test-1, test-2, test-3*, and *test-4*, where *name* is an object identifier, *gender* is a symmetric attribute, and the remaining attributes are asymmetric binary.

For asymmetric attribute values, let the values Y (*yes*) and P (*positive*) be set to 1, and the value N (*no* or *negative*) be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric variables. According to Equation (7.10), the distance between each pair of the three patients, Jack, Mary, and Jim, is

$$\begin{aligned} d(Jack, Mary) &= \tfrac{0+1}{2+0+1} = 0.33 \\ d(Jack, Jim) &= \tfrac{1+1}{1+1+1} = 0.67 \\ d(Mary, Jim) &= \tfrac{1+2}{1+1+2} = 0.75 \end{aligned}$$

Table 7.2 A relational table where patients are described by binary attributes.

name	*gender*	*fever*	*cough*	*test-1*	*test-2*	*test-3*	*test-4*
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	Y	N	N	N	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

These measurements suggest that Mary and Jim are unlikely to have a similar disease because they have the highest dissimilarity value among the three pairs. Of the three patients, Jack and Mary are the most likely to have a similar disease. ■

7.2.3 Categorical, Ordinal, and Ratio-Scaled Variables

"How can we compute the dissimilarity between objects described by categorical, ordinal, and ratio-scaled variables?"

Categorical Variables

A **categorical variable** is a generalization of the binary variable in that it can take on more than two states. For example, *map_color* is a categorical variable that may have, say, five states: *red, yellow, green, pink*, and *blue*.

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

"How is dissimilarity computed between objects described by categorical variables?" The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p}, \tag{7.12}$$

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

Example 7.3 **Dissimilarity between categorical variables.** Suppose that we have the sample data of Table 7.3, except that only the *object-identifier* and the variable (or attribute) *test-1* are available, where *test-1* is categorical. (We will use *test-2* and *test-3* in later examples.) Let's compute the dissimilarity matrix (7.2), that is,

Table 7.3 A sample data table containing variables of mixed type.

object identifier	*test-1 (categorical)*	*test-2 (ordinal)*	*test-3 (ratio-scaled)*
1	code-A	excellent	445
2	code-B	fair	22
3	code-C	good	164
4	code-A	excellent	1,210

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

Since here we have one categorical variable, *test-1*, we set $p = 1$ in Equation (7.12) so that $d(i, j)$ evaluates to 0 if objects i and j match, and 1 if the objects differ. Thus, we get

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

■

Categorical variables can be encoded by asymmetric binary variables by creating a new binary variable for each of the M states. For an object with a given state value, the binary variable representing that state is set to 1, while the remaining binary variables are set to 0. For example, to encode the categorical variable *map_color*, a binary variable can be created for each of the five colors listed above. For an object having the color *yellow*, the *yellow* variable is set to 1, while the remaining four variables are set to 0. The dissimilarity coefficient for this form of encoding can be calculated using the methods discussed in Section 7.2.2.

Ordinal Variables

A **discrete ordinal variable** resembles a categorical variable, except that the M states of the ordinal value are ordered in a meaningful sequence. Ordinal variables are very useful for registering subjective assessments of qualities that cannot be measured objectively. For example, professional ranks are often enumerated in a sequential order, such as *assistant*, *associate*, and *full* for professors. A **continuous ordinal variable** looks like a set of continuous data of an unknown scale; that is, the relative ordering of the values is essential but their actual magnitude is not. For example, the relative ranking in a particular sport (e.g., gold, silver, bronze) is often more essential than the actual values of a particular measure. Ordinal variables may also be obtained from the discretization of interval-scaled quantities by splitting the value range into a finite number of classes. The values of an ordinal variable can be mapped to *ranks*. For example, suppose that an ordinal variable f has M_f states. These ordered states define the ranking $1, \ldots, M_f$.

"How are ordinal variables handled?" The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is a variable from a set of ordinal variables describing

n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the ith object is x_{if}, and f has M_f ordered states, representing the ranking $1, \ldots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \ldots, M_f\}$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto [0.0,1.0] so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the ith object in the fth variable by
$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \tag{7.13}$$
3. Dissimilarity can then be computed using any of the distance measures described in Section 7.2.1 for interval-scaled variables, using z_{if} to represent the f value for the ith object.

Example 7.4 **Dissimilarity between ordinal variables.** Suppose that we have the sample data of Table 7.3, except that this time only the *object-identifier* and the continuous ordinal variable, *test*-2, are available. There are three states for *test*-2, namely *fair*, *good*, and *excellent*, that is $M_f = 3$. For step 1, if we replace each value for *test*-2 by its rank, the four objects are assigned the ranks 3, 1, 2, and 3, respectively. Step 2 normalizes the ranking by mapping rank 1 to 0.0, rank 2 to 0.5, and rank 3 to 1.0. For step 3, we can use, say, the Euclidean distance (Equation (7.5)), which results in the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}$$

■

Ratio-Scaled Variables

A **ratio-scaled variable** makes a positive measurement on a nonlinear scale, such as an exponential scale, approximately following the formula

$$Ae^{Bt} \quad \text{or} \quad Ae^{-Bt} \tag{7.14}$$

where A and B are positive constants, and t typically represents time. Common examples include the growth of a bacteria population or the decay of a radioactive element.

"*How can I compute the dissimilarity between objects described by ratio-scaled variables?*" There are three methods to handle ratio-scaled variables for computing the dissimilarity between objects.

- Treat ratio-scaled variables like interval-scaled variables. This, however, is not usually a good choice since it is likely that the scale may be distorted.
- Apply **logarithmic transformation** to a ratio-scaled variable f having value x_{if} for object i by using the formula $y_{if} = \log(x_{if})$. The y_{if} values can be treated as interval-valued, as described in Section 7.2.1. Notice that for some ratio-scaled variables, log-log or other transformations may be applied, depending on the variable's definition and the application.
- Treat x_{if} as continuous ordinal data and treat their ranks as interval-valued.

The latter two methods are the most effective, although the choice of method used may depend on the given application.

Example 7.5 **Dissimilarity between ratio-scaled variables.** This time, we have the sample data of Table 7.3, except that only the *object-identifier* and the ratio-scaled variable, *test-3*, are available. Let's try a logarithmic transformation. Taking the *log* of *test-3* results in the values 2.65, 1.34, 2.21, and 3.08 for the objects 1 to 4, respectively. Using the Euclidean distance (Equation (7.5)) on the transformed values, we obtain the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1.31 & 0 & & \\ 0.44 & 0.87 & 0 & \\ 0.43 & 1.74 & 0.87 & 0 \end{bmatrix}$$

■

7.2.4 Variables of Mixed Types

Sections 7.2.1 to 7.2.3 discussed how to compute the dissimilarity between objects described by variables of the same type, where these types may be either *interval-scaled*, *symmetric binary, asymmetric binary*, *categorical, ordinal*, or *ratio-scaled*. However, in many real databases, objects are described by a *mixture* of variable types. In general, a database can contain all of the six variable types listed above.

"*So, how can we compute the dissimilarity between objects of mixed variable types?*" One approach is to group each kind of variable together, performing a separate cluster analysis for each variable type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate cluster analysis per variable type will generate compatible results.

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

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

$$d(i, j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}, \tag{7.15}$$

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

- If f is interval-based: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}}$, where h runs over all nonmissing objects for variable f.
- If f is binary or categorical: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$; otherwise $d_{ij}^{(f)} = 1$.
- If f is ordinal: compute the ranks r_{if} and $z_{if} = \frac{r_{if} - 1}{M_f - 1}$, and treat z_{if} as interval-scaled.
- If f is ratio-scaled: either perform logarithmic transformation and treat the transformed data as interval-scaled; or treat f as continuous ordinal data, compute r_{if} and z_{if}, and then treat z_{if} as interval-scaled.

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

Example 7.6 **Dissimilarity between variables of mixed type.** Let's compute a dissimilarity matrix for the objects of Table 7.3. Now we will consider *all* of the variables, which are of different types. In Examples 7.3 to 7.5, we worked out the dissimilarity matrices for each of the individual variables. The procedures we followed for *test*-1 (which is categorical) and *test*-2 (which is ordinal) are the same as outlined above for processing variables of mixed types. Therefore, we can use the dissimilarity matrices obtained for *test*-1 and *test*-2 later when we compute Equation (7.15). First, however, we need to complete some work for *test*-3 (which is ratio-scaled). We have already applied a logarithmic transformation to its values. Based on the transformed values of 2.65, 1.34, 2.21, and 3.08 obtained for the objects 1 to 4, respectively, we let $max_h x_h = 3.08$ and $min_h x_h = 1.34$. We then normalize the values in the dissimilarity matrix obtained in Example 7.5 by dividing each one by $(3.08 - 1.34) = 1.74$. This results in the following dissimilarity matrix for *test*-3:

$$\begin{bmatrix} 0 & & & \\ 0.75 & 0 & & \\ 0.25 & 0.50 & 0 & \\ 0.25 & 1.00 & 0.50 & 0 \end{bmatrix}$$

We can now use the dissimilarity matrices for the three variables in our computation of Equation (7.15). For example, we get $d(2, 1) = \frac{1(1)+1(1)+1(0.75)}{3} = 0.92$. The resulting dissimilarity matrix obtained for the data described by the three variables of mixed types is:

$$\begin{bmatrix} 0 & & & \\ 0.92 & 0 & & \\ 0.58 & 0.67 & 0 & \\ 0.08 & 1.00 & 0.67 & 0 \end{bmatrix}$$

If we go back and look at Table 7.3, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for *test*-1 and *test*-2. This is confirmed by the dissimilarity matrix, where $d(4, 1)$ is the lowest value for any pair of different objects. Similarly, the matrix indicates that objects 2 and 4 are the least similar. ■

7.2.5 Vector Objects

In some applications, such as information retrieval, text document clustering, and biological taxonomy, we need to compare and cluster complex objects (such as documents) containing a large number of symbolic entities (such as keywords and phrases). To measure the distance between complex objects, it is often desirable to abandon traditional metric distance computation and introduce a nonmetric similarity function.

There are several ways to define such a similarity function, $s(\boldsymbol{x}, \boldsymbol{y})$, to compare two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. One popular way is to define the similarity function as a **cosine measure** as follows:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^t \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}, \tag{7.16}$$

where $\boldsymbol{x}^t$ is a transposition of vector $\boldsymbol{x}$, $||\boldsymbol{x}||$ is the Euclidean norm of vector $\boldsymbol{x}$,[1] $||\boldsymbol{y}||$ is the Euclidean norm of vector $\boldsymbol{y}$, and s is essentially the cosine of the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. This value is invariant to rotation and dilation, but it is not invariant to translation and general linear transformation.

[1]The Euclidean normal of vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$ is defined as $\sqrt{x_1^2 + x_2^2 + \ldots + x_p^2}$. Conceptually, it is the length of the vector.

When variables are binary-valued (0 or 1), the above similarity function can be interpreted in terms of shared features and attributes. Suppose an object $\boldsymbol{x}$ possesses the ith attribute if $x_i = 1$. Then $\boldsymbol{x}^t \cdot \boldsymbol{y}$ is the number of attributes possessed by both $\boldsymbol{x}$ and $\boldsymbol{y}$, and $|\boldsymbol{x}||\boldsymbol{y}|$ is the geometric mean of the number of attributes possessed by $\boldsymbol{x}$ and the number possessed by $\boldsymbol{y}$. Thus $s(\boldsymbol{x}, \boldsymbol{y})$ is a measure of relative possession of common attributes.

Example 7.7 **Nonmetric similarity between two objects using cosine.** Suppose we are given two vectors, $\boldsymbol{x} = (1, 1, 0, 0)$ and $\boldsymbol{y} = (0, 1, 1, 0)$. By Equation (7.16), the similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$ is $s(\boldsymbol{x}, \boldsymbol{y}) = \frac{(0+1+0+0)}{\sqrt{2}\sqrt{2}} = 0.5$. ■

A simple variation of the above measure is

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^t \cdot \boldsymbol{y}}{\boldsymbol{x}^t \cdot \boldsymbol{x} + \boldsymbol{y}^t \cdot \boldsymbol{y} - \boldsymbol{x}^t \cdot \boldsymbol{y}} \tag{7.17}$$

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

Notice that there are many ways to select a particular similarity (or distance) function or normalize the data for cluster analysis. There is no universal standard to guide such selection. The appropriate selection of such measures will heavily depend on the given application. One should bear this in mind and refine the selection of such measures to ensure that the clusters generated are meaningful and useful for the application at hand.

7.3 A Categorization of Major Clustering Methods

Many clustering algorithms exist in the literature. It is difficult to provide a crisp categorization of clustering methods because these categories may overlap, so that a method may have features from several categories. Nevertheless, it is useful to present a relatively organized picture of the different clustering methods.

In general, the major clustering methods can be classified into the following categories.

Partitioning methods: Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$. That is, it classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. Notice that the second requirement can be relaxed in some fuzzy partitioning techniques. References to such techniques are given in the bibliographic notes.

Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an **iterative relocation technique** that attempts to

improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or very different. There are various kinds of other criteria for judging the quality of partitions.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of a few popular heuristic methods, such as (1) the *k-means* algorithm, where each cluster is represented by the mean value of the objects in the cluster, and (2) the *k-medoids* algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. To find clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended. Partitioning-based clustering methods are studied in depth in Section 7.4.

Hierarchical methods: A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the *bottom-up* approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The *divisive approach*, also called the *top-down* approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices. However, such techniques cannot correct erroneous decisions. There are two approaches to improving the quality of hierarchical clustering: (1) perform careful analysis of object "linkages" at each hierarchical partitioning, such as in Chameleon, or (2) integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algorithm to group objects into *microclusters*, and then performing *macroclustering* on the microclusters using another clustering method such as iterative relocation, as in BIRCH. Hierarchical clustering methods are studied in Section 7.5.

Density-based methods: Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of *density*. Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a

minimum number of points. Such a method can be used to filter out noise (outliers) and discover clusters of arbitrary shape.

DBSCAN and its extension, OPTICS, are typical density-based methods that grow clusters according to a density-based connectivity analysis. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions. Density-based clustering methods are studied in Section 7.6.

Grid-based methods: Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.

STING is a typical example of a grid-based method. WaveCluster applies wavelet transformation for clustering analysis and is both grid-based and density-based. Grid-based clustering methods are studied in Section 7.7.

Model-based methods: Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking "noise" or outliers into account and thus yielding robust clustering methods.

EM is an algorithm that performs expectation-maximization analysis based on statistical modeling. COBWEB is a conceptual learning algorithm that performs probability analysis and takes *concepts* as a model for clusters. SOM (or self-organizing feature map) is a neural network-based algorithm that clusters by mapping high-dimensional data into a 2-D or 3-D feature map, which is also useful for data visualization. Model-based clustering methods are studied in Section 7.8.

The choice of clustering algorithm depends both on the type of data available and on the particular purpose of the application. If cluster analysis is used as a descriptive or exploratory tool, it is possible to try several algorithms on the same data to see what the data may disclose.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category. Furthermore, some applications may have clustering criteria that require the integration of several clustering techniques.

Aside from the above categories of clustering methods, there are two classes of clustering tasks that require special attention. One is *clustering high-dimensional data*, and the other is *constraint-based clustering*.

Clustering high-dimensional data is a particularly important task in cluster analysis because many applications require the analysis of objects containing a large

number of features or dimensions. For example, text documents may contain thousands of terms or keywords as features, and DNA microarray data may provide information on the expression levels of thousands of genes under hundreds of conditions. Clustering high-dimensional data is challenging due to the curse of dimensionality. Many dimensions may not be relevant. As the number of dimensions increases, the data become increasingly sparse so that the distance measurement between pairs of points become meaningless and the average density of points anywhere in the data is likely to be low. Therefore, a different clustering methodology needs to be developed for high-dimensional data. CLIQUE and PROCLUS are two influential *subspace clustering methods*, which search for clusters in subspaces (or subsets of dimensions) of the data, rather than over the entire data space. **Frequent pattern–based clustering**, another clustering methodology, extracts *distinct frequent patterns* among subsets of dimensions that occur frequently. It uses such patterns to group objects and generate meaningful clusters. pCluster is an example of frequent pattern–based clustering that groups objects based on their pattern similarity. High-dimensional data clustering methods are studied in Section 7.9.

Constraint-based clustering is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints. A constraint expresses a user's expectation or describes "properties" of the desired clustering results, and provides an effective means for communicating with the clustering process. Various kinds of constraints can be specified, either by a user or as per application requirements. Our focus of discussion will be on *spatial clustering* with the existence of obstacles and clustering under user-specified constraints. In addition, *semi-supervised clustering* is described, which employs, for example, pairwise constraints (such as pairs of instances labeled as belonging to the same or different clusters) in order to improve the quality of the resulting clustering. Constraint-based clustering methods are studied in Section 7.10.

In the following sections, we examine each of the above clustering methods in detail. We also introduce algorithms that integrate the ideas of several clustering methods. Outlier analysis, which typically involves clustering, is described in Section 7.11. In general, the notation used in these sections is as follows. Let D be a data set of n objects to be clustered. An **object** is described by d variables (attributes or dimensions) and therefore may also be referred to as a *point* in d-dimensional object space. Objects are represented in bold italic font (e.g., $\boldsymbol{p}$).

7.4 Partitioning Methods

Given D, a data set of n objects, and k, the number of clusters to form, a **partitioning algorithm** organizes the objects into k partitions ($k \leq n$), where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance, so that the objects within a cluster are "similar," whereas the objects of different clusters are "dissimilar" in terms of the data set attributes.

7.4.1 Classical Partitioning Methods: *k*-Means and *k*-Medoids

The most well-known and commonly used partitioning methods are *k-means*, *k-medoids*, and their variations.

Centroid-Based Technique: The *k*-Means Method

The ***k*-means algorithm** takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster's *centroid* or *center of gravity*.

"*How does the k-means algorithm work?*" The k-means algorithm proceeds as follows. First, it randomly selects k of the objects, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the **square-error criterion** is used, defined as

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} |\boldsymbol{p} - \boldsymbol{m_i}|^2, \tag{7.18}$$

where E is the sum of the square error for all objects in the data set; $\boldsymbol{p}$ is the point in space representing a given object; and $\boldsymbol{m_i}$ is the mean of cluster C_i (both $\boldsymbol{p}$ and $\boldsymbol{m_i}$ are multidimensional). In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This criterion tries to make the resulting k clusters as compact and as separate as possible. The k-means procedure is summarized in Figure 7.2.

Example 7.8 **Clustering by *k*-means partitioning.** Suppose that there is a set of objects located in space as depicted in the rectangle shown in Figure 7.3(a). Let $k = 3$; that is, the user would like the objects to be partitioned into three clusters.

According to the algorithm in Figure 7.2, we arbitrarily choose three objects as the three initial cluster centers, where cluster centers are marked by a "+". Each object is distributed to a cluster based on the cluster center to which it is the nearest. Such a distribution forms silhouettes encircled by dotted curves, as shown in Figure 7.3(a).

Next, the cluster centers are updated. That is, the mean value of each cluster is recalculated based on the current objects in the cluster. Using the new cluster centers, the objects are redistributed to the clusters based on which cluster center is the nearest. Such a redistribution forms new silhouettes encircled by dashed curves, as shown in Figure 7.3(b).

This process iterates, leading to Figure 7.3(c). The process of iteratively reassigning objects to clusters to improve the partitioning is referred to as *iterative relocation*. Eventually, no redistribution of the objects in any cluster occurs, and so the process terminates. The resulting clusters are returned by the clustering process. ■

The algorithm attempts to determine k partitions that minimize the square-error function. It works well when the clusters are compact clouds that are rather well

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

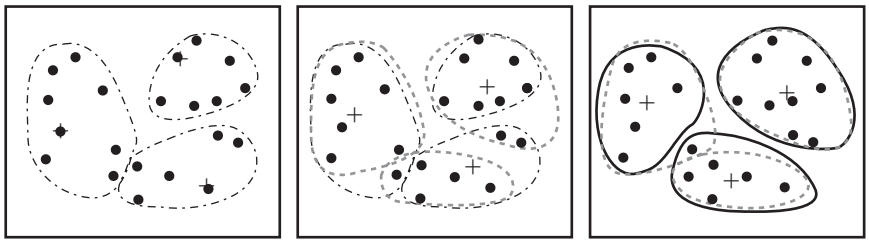
- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
(5) **until** no change;

Figure 7.2 The *k*-means partitioning algorithm.



(a) (b) (c)

Figure 7.3 Clustering of a set of objects based on the *k*-means method. (The mean of each cluster is marked by a "+".)

separated from one another. The method is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum.

The *k*-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users to specify *k*, the number of clusters, in advance can be seen as a disadvantage. The *k*-means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size. Moreover, it

is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value.

There are quite a few variants of the *k*-means method. These can differ in the selection of the initial *k* means, the calculation of dissimilarity, and the strategies for calculating cluster means. An interesting strategy that often yields good results is to first apply a hierarchical agglomeration algorithm, which determines the number of clusters and finds an initial clustering, and then use iterative relocation to improve the clustering.

Another variant to *k*-means is the ***k*-modes method,** which extends the *k*-means paradigm to cluster categorical data by replacing the means of clusters with modes, using new dissimilarity measures to deal with categorical objects and a frequency-based method to update modes of clusters. The *k*-means and the *k*-modes methods can be integrated to cluster data with mixed numeric and categorical values.

The **EM (Expectation-Maximization)** algorithm (which will be further discussed in Section 7.8.1) extends the *k*-means paradigm in a different way. Whereas the *k*-means algorithm assigns each object to a cluster, in EM each object is assigned to *each* cluster according to a weight representing its probability of membership. In other words, there are no strict boundaries between clusters. Therefore, new means are computed based on weighted measures.

"*How can we make the k-means algorithm more scalable?*" A recent approach to scaling the *k*-means algorithm is based on the idea of identifying three kinds of regions in data: regions that are compressible, regions that must be maintained in main memory, and regions that are discardable. An object is *discardable* if its membership in a cluster is ascertained. An object is *compressible* if it is not discardable but belongs to a tight subcluster. A data structure known as a *clustering feature* is used to summarize objects that have been discarded or compressed. If an object is neither discardable nor compressible, then it should be *retained in main memory*. To achieve scalability, the iterative clustering algorithm only includes the clustering features of the compressible objects and the objects that must be retained in main memory, thereby turning a secondary-memory-based algorithm into a main-memory-based algorithm. An alternative approach to scaling the *k*-means algorithm explores the microclustering idea, which first groups nearby objects into "microclusters" and then performs *k*-means clustering on the microclusters. Microclustering is further discussed in Section 7.5.

Representative Object-Based Technique: The *k*-Medoids Method

The *k*-means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data. This effect is particularly exacerbated due to the use of the *square*-error function (Equation (7.18)).

"*How might the algorithm be modified to diminish such sensitivity?*" Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of

the dissimilarities between each object and its corresponding reference point. That is, an **absolute-error criterion** is used, defined as

$$E = \sum_{j=1}^{k} \sum_{\boldsymbol{p} \in C_j} |\boldsymbol{p} - \boldsymbol{o_j}|, \tag{7.19}$$

where E is the sum of the absolute error for all objects in the data set; $\boldsymbol{p}$ is the point in space representing a given object in cluster C_j; and $\boldsymbol{o_j}$ is the representative object of C_j. In general, the algorithm iterates until, eventually, each representative object is actually the **medoid**, or most centrally located object, of its cluster. This is the basis of the *k*-**medoids method** for grouping n objects into k clusters.

Let's look closer at *k*-medoids clustering. The initial representative objects (or seeds) are chosen arbitrarily. The iterative process of replacing representative objects by nonrepresentative objects continues as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster. To determine whether a nonrepresentative object, $\boldsymbol{o_{random}}$, is a good replacement for a current representative object, $\boldsymbol{o_j}$, the following four cases are examined for each of the nonrepresentative objects, $\boldsymbol{p}$, as illustrated in Figure 7.4.

- **Case 1:** $\boldsymbol{p}$ currently belongs to representative object, $\boldsymbol{o_j}$. If $\boldsymbol{o_j}$ is replaced by $\boldsymbol{o_{random}}$ as a representative object and $\boldsymbol{p}$ is closest to one of the other representative objects, $\boldsymbol{o_i}$, $i \neq j$, then $\boldsymbol{p}$ is reassigned to $\boldsymbol{o_i}$.
- **Case 2:** $\boldsymbol{p}$ currently belongs to representative object, $\boldsymbol{o_j}$. If $\boldsymbol{o_j}$ is replaced by $\boldsymbol{o_{random}}$ as a representative object and $\boldsymbol{p}$ is closest to $\boldsymbol{o_{random}}$, then $\boldsymbol{p}$ is reassigned to $\boldsymbol{o_{random}}$.
- **Case 3:** $\boldsymbol{p}$ currently belongs to representative object, $\boldsymbol{o_i}$, $i \neq j$. If $\boldsymbol{o_j}$ is replaced by $\boldsymbol{o_{random}}$ as a representative object and $\boldsymbol{p}$ is still closest to o_i, then the assignment does not change.

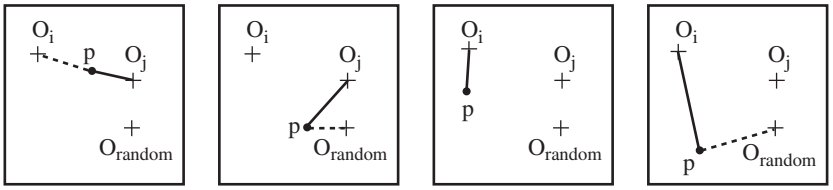


Figure 7.4 Four cases of the cost function for *k*-medoids clustering.

- **Case 4:** $\boldsymbol{p}$ currently belongs to representative object, $\boldsymbol{o_i}$, $i \neq j$. If $\boldsymbol{o_j}$ is replaced by $\boldsymbol{o}_{random}$ as a representative object and $\boldsymbol{p}$ is closest to $\boldsymbol{o}_{random}$, then $\boldsymbol{p}$ is reassigned to $\boldsymbol{o}_{random}$.

Each time a reassignment occurs, a difference in absolute error, E, is contributed to the cost function. Therefore, the cost function calculates the *difference* in absolute-error value if a current representative object is replaced by a nonrepresentative object. The total cost of swapping is the sum of costs incurred by all nonrepresentative objects. If the total cost is negative, then $\boldsymbol{o_j}$ is replaced or swapped with $\boldsymbol{o}_{random}$ since the actual absolute error E would be reduced. If the total cost is positive, the current representative object, $\boldsymbol{o_j}$, is considered acceptable, and nothing is changed in the iteration.

PAM (Partitioning Around Medoids) was one of the first k-medoids algorithms introduced (Figure 7.5). It attempts to determine k partitions for n objects. After an initial random selection of k representative objects, the algorithm repeatedly tries to make a better choice of cluster representatives. All of the possible pairs of objects are analyzed, where one object in each pair is considered a representative object and the other is not. The quality of the resulting clustering is calculated for each such combination. An object, $\boldsymbol{o_j}$, is replaced with the object causing the greatest reduction in error. The set of best objects for each cluster in one iteration forms the representative objects for the next iteration. The final set of representative objects are the respective medoids of the clusters. The complexity of each iteration is $O(k(n-k)^2)$. For large values of n and k, such computation becomes very costly.

Algorithm: k-medoids. PAM, a k-medoids algorithm for partitioning based on medoid or central objects.

Input:

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

Output: A set of k clusters.

Method:

(1) arbitrarily choose k objects in D as the initial representative objects or seeds;
(2) **repeat**
(3) assign each remaining object to the cluster with the nearest representative object;
(4) randomly select a nonrepresentative object, $\boldsymbol{o}_{\text{random}}$;
(5) compute the total cost, S, of swapping representative object, $\boldsymbol{o_j}$, with $\boldsymbol{o}_{\text{random}}$;
(6) **if** $S < 0$ **then** swap $\boldsymbol{o_j}$ with $\boldsymbol{o}_{\text{random}}$ to form the new set of k representative objects;
(7) **until** no change;

Figure 7.5 PAM, a k-medoids partitioning algorithm.

"Which method is more robust—k-means or k-medoids?" The k-medoids method is more robust than k-means in the presence of noise and outliers, because a medoid is less influenced by outliers or other extreme values than a mean. However, its processing is more costly than the k-means method. Both methods require the user to specify k, the number of clusters.

Aside from using the mean or the medoid as a measure of cluster center, other alternative measures are also commonly used in partitioning clustering methods. The *median* can be used, resulting in the *k-median* method, where the median or "middle value" is taken for each ordered attribute. Alternatively, in the *k-modes* method, the most frequent value for each attribute is used.

7.4.2 Partitioning Methods in Large Databases: From k-Medoids to CLARANS

"How efficient is the k-medoids algorithm on large data sets?" A typical k-medoids partitioning algorithm like PAM works effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called **CLARA** (Clustering LARge Applications), can be used.

The idea behind CLARA is as follows: Instead of taking the whole set of data into consideration, a small portion of the actual data is chosen as a representative of the data. Medoids are then chosen from this sample using PAM. If the sample is selected in a fairly random manner, it should closely represent the original data set. The representative objects (medoids) chosen will likely be similar to those that would have been chosen from the whole data set. CLARA draws multiple samples of the data set, applies PAM on each sample, and returns its best clustering as the output. As expected, CLARA can deal with larger data sets than PAM. The complexity of each iteration now becomes $O(ks^2 + k(n-k))$, where s is the size of the sample, k is the number of clusters, and n is the total number of objects.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best k medoids among a given data set, whereas CLARA searches for the best k medoids among the *selected sample* of the data set. CLARA cannot find the best clustering if any of the best sampled medoids is not among the best k medoids. That is, if an object $\boldsymbol{o_i}$ is one of the best k medoids but is not selected during sampling, CLARA will never find the best clustering. This is, therefore, a trade-off for efficiency. A good clustering based on sampling will not necessarily represent a good clustering of the whole data set if the sample is biased.

"How might we improve the quality and scalability of CLARA?" A k-medoids type algorithm called **CLARANS** (Clustering Large Applications based upon RANdomized Search) was proposed, which combines the sampling technique with PAM. However, unlike CLARA, CLARANS does not confine itself to any sample at any given time. While CLARA has a fixed sample at each stage of the search, CLARANS draws a sample with some randomness in each step of the search. Conceptually, the clustering process can be viewed as a search through a graph, where each node is a potential solution (a set of k medoids). Two nodes are *neighbors* (that is, connected by an arc in

the graph) if their sets differ by only one object. Each node can be assigned a cost that is defined by the total dissimilarity between every object and the medoid of its cluster. At each step, PAM examines all of the neighbors of the current node in its search for a minimum cost solution. The current node is then replaced by the neighbor with the largest descent in costs. Because CLARA works on a sample of the entire data set, it examines fewer neighbors and restricts the search to subgraphs that are smaller than the original graph. While CLARA draws a sample of nodes at the beginning of a search, CLARANS dynamically draws a random sample of neighbors in each step of a search. The number of neighbors to be randomly sampled is restricted by a user-specified parameter. In this way, CLARANS does not confine the search to a localized area. If a better neighbor is found (i.e., having a lower error), CLARANS moves to the neighbor's node and the process starts again; otherwise, the current clustering produces a local minimum. If a local minimum is found, CLARANS starts with new randomly selected nodes in search for a new local minimum. Once a user-specified number of local minima has been found, the algorithm outputs, as a solution, the best local minimum, that is, the local minimum having the lowest cost.

CLARANS has been experimentally shown to be more effective than both PAM and CLARA. It can be used to find the most "natural" number of clusters using a *silhouette coefficient*—a property of an object that specifies how much the object truly belongs to the cluster. CLARANS also enables the detection of outliers. However, the computational complexity of CLARANS is about $O(n^2)$, where n is the number of objects. Furthermore, its clustering quality is dependent on the sampling method used. The ability of CLARANS to deal with data objects that reside on disk can be further improved by focusing techniques that explore spatial data structures, such as R*-trees.

7.5 Hierarchical Methods

A hierarchical clustering method works by grouping data objects into a tree of clusters. Hierarchical clustering methods can be further classified as either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top-down (splitting) fashion. The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision has been executed. That is, if a particular merge or split decision later turns out to have been a poor choice, the method cannot backtrack and correct it. Recent studies have emphasized the integration of hierarchical agglomeration with iterative relocation methods.

7.5.1 Agglomerative and Divisive Hierarchical Clustering

In general, there are two types of hierarchical clustering methods:

- **Agglomerative hierarchical clustering:** This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger