Contents

List of Figures

List of Tables

Chapter 8

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. *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, you will learn the requirements of clustering methods for operating on large amounts of data. You will also study how to compute dissimilarities between objects represented by various attribute or variable types. You will study several clustering techniques, organized into the following categories: *partitioning methods, hierarchical methods, density-based methods*, *grid-based methods*, and *model-based methods*. Clustering can also be used for *outlier detection*, which forms the final topic of this chapter.

8.1 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 in many applications.

Cluster analysis is an important human activity. Early in childhood, one learns how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious classification schemes. Cluster analysis has been widely used in numerous applications, including pattern recognition, data analysis, image processing, and market research. By clustering, one can identify crowded and sparse regions, and therefore, discover overall distribution patterns and interesting correlations among data attributes.

"What are some typical applications of clustering?"

In business, clustering may 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 motor insurance policy holders with a high average claim cost, as well as the identification of groups of houses in a city according to house type, value, and geographical location. It may also help classify documents on the WWW for information discovery. 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 classification and characterization, operating on the detected clusters.

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 studied extensively 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 **conceptual clustering**, a group of objects forms a class only if it is describable by a concept. This differs from conventional clustering which measures similarity based on geometric distance. Conceptual clustering consists of two components: (1) it discovers the appropriate classes, and (2) it forms descriptions for each class, as in classification. The guideline of striving for high intraclass similarity and low interclass similarity still applies.

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 where its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining.

1. **Scalability**: Many clustering algorithms work well in small data sets containing less than 200 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.

2. **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.

3. **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 which can detect clusters of arbitrary shape.

4. **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 are often quite sensitive to input parameters. Parameters are often hard to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but also makes the quality of clustering difficult to control.

5. **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.

6. **Insensitivity to the order of input records**: Some clustering algorithms are sensitive to the order of input data, e.g., the same set of data, when presented with different orderings to such an algorithm, may generate dramatically different clusters. It is important to develop algorithms which are insensitive to the order of input.

7. **High dimensionality**: A database or a data warehouse may 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. It is challenging to cluster data objects in high-dimensional space, especially considering that data in high-dimensional space can be very sparse and highly skewed.

8. **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 cash stations (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 customer requirements per region. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

9. **Interpretability and usability**: Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied up with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering 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 and outlier analysis.

8.2 Types of data in clustering analysis

In this section, we study the types of data which often occur in clustering 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, etc. Main memory-based clustering algorithms typically operate on either of the following two data structures.

1. **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, race, and so on. The structure is in the form of a relational table, or n-by-p matrix (n objects $\times$ p variables), as shown in (8.1).

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix} \tag{8.1}$$

2. **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 as shown below,

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix} \tag{8.2}$$

   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 (8.2). Measures of dissimilarity are discussed throughout this section.

The data matrix is often called a **two-mode** matrix, whereas the dissimilarity matrix is called a **one-mode** matrix, since the rows and columns of the former represent different entities, while those of the latter represent the same entity. 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.

"How can dissimilarity, $d(i,j)$, be assessed?", you may wonder.

In this section, we discuss how object dissimilarity can be computed for objects described by *interval-scaled* variables, by *binary* variables, by *nominal, ordinal*, and *ratio-scaled* variables, or combinations of these variable types. The dissimilarity data can later be used to compute clusters of objects.

8.2.1 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, one may like 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,

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \cdots + |x_{nf} - m_f|), \tag{8.3}$$

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

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

$$z_{if} = \frac{x_{if} - m_f}{s_f}. \tag{8.4}$$

   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 3 under normalization techniques for data preprocessing.

"OK", you now ask, *"once I have standardized the data, how can I compute the dissimilarity between objects?"*

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

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \cdots + |x_{ip} - x_{jp}|^2}, \tag{8.5}$$

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

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

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|. \tag{8.6}$$

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

1. $d(i,j) \geq 0$: This states that distance is a nonnegative number;

2. $d(i,i) = 0$: This states that the distance of an object to itself is 0;

3. $d(i,j) = d(j,i)$: This states that distance is a symmetric function; and

4. $d(i,j) \leq d(i,h) + d(h,j)$: This is a *triangular inequality* which states that going directly from object i to object j in space is no more than making a detour over any other object h.

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

$$d(i,j) = (|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \cdots + |x_{ip} - x_{jp}|^q)^{1/q}, \tag{8.7}$$

where q is a positive integer. It represents the Manhattan distance when $q = 1$, and Euclidean distance when $q = 2$.

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

$$d(i,j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_p|x_{ip} - x_{jp}|^2}. \tag{8.8}$$

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

8.2.2 Binary variables

This section describes how to compute the dissimilarity between objects described by either *symmetric* or *asymmetric binary variables*.

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 I 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 a 2-by-2 contingency table, Table 8.1, 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 $p = q + r + s + t$.

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

Table 8.1: A contingency table for binary variables.

"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*. Similarity that is based on symmetric binary variables is called **invariant similarity** in that the result does not change when some or all of the binary variables are coded differently. For invariant similarities, the most well-known coefficient for assessing the dissimilarity between objects i and j is the **simple matching coefficient**, defined in (8.9).

$$d(i,j) = \frac{r+s}{q+r+s+t} \tag{8.9}$$

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 1's (a positive match) is then considered more significant than that of two zeros (a negative match). Therefore, such binary variables are often considered "monary" (as if having one state). The similarity based on such variables is called **noninvariant similarity**. For noninvariant similarities, the most well-known coefficient

is the **Jaccard coefficient**, defined in (8.10), where the number of negative matches, d, is considered unimportant and thus is ignored in the computation.

$$d(i,j) = \frac{r+s}{q+r+s} \tag{8.10}$$

When both symmetric and asymmetric binary variables occur in the same data set, the mixed variables approach described in Section 8.2.4 can be applied.

Example 8.1 Dissimilarity between binary variables. Suppose that a patient record table, Table 8.2, contains the attributes *name, gender, fever, cough, test-1, test-2, test-3*, and *test-4*, where *name* is an object-id, *gender* is a symmetric attribute, and the remaining attributes are asymmetric binary.

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	P	N	N	N	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.2: A relational table containing mostly binary attributes.

For asymmetric attribute values, let the values Y and P be set to 1, and the value N be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric variables. According to the Jaccard coefficient formula (8.10), the distance between each pair of the three patients, Jack, Mary and Jim, should be,

$$d(jack, mary) = \frac{0+1}{2+0+1} = 0.33 \tag{8.11}$$

$$d(jack, jim) = \frac{1+1}{1+1+1} = 0.67 \tag{8.12}$$

$$d(jim, mary) = \frac{1+2}{1+1+2} = 0.75 \tag{8.13}$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease 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. □

8.2.3 Nominal, ordinal, and ratio-scaled variables

This section discusses how to compute the dissimilarity between objects described by nominal, ordinal, and ratio-scaled variables.

Nominal variables

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

Let the number of states of a nominal variable be M. The states can be denoted by letters, symbols, or a set of integers, such as 1, 2, ..., 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 nominal variables?" The dissimilarity between two objects i and j can be computed using the **simple matching** approach as in (8.14):

$$d(i,j) = \frac{p-m}{p}, \tag{8.14}$$

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.

Nominal variables can be encoded by asymmetric binary variables by creating a new binary variable for each of the M nominal 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 nominal 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 8.2.2.

Ordinal variables

A **discrete ordinal variable** resembles a nominal 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. 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 not their actual magnitude. For example, the relative ranking in a particular sport 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 $1, .., M_f$.

"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 $1, .., M_f$. 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 [0-1] so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i-th object in the f-th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \tag{8.15}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 8.2.1 for interval-scaled variables, using z_{if} to represent the f value for the ith object.

Ratio-scaled variables

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

$$Ae^{Bt} \quad or \quad Ae^{-Bt}, \tag{8.16}$$

where A and B are positive constants. Typical examples include the growth of a bacteria population, or the decay of a radio-active element.

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

1. 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.

2. 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 8.2.1. Notice that for some ratio-scaled variables, one may also apply log-log transformation, or other transformations, depending on the definition and application.

3. 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 be dependent on the given application.

8.2.4 Variables of mixed types

Sections 8.2.1 to 8.2.3 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*, *nominal*, *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 variables together, performing a separate cluster analysis for each variable type. This is feasible if these analyses derive satisfactory results. However, in real applications, it is unlikely that a separate cluster analysis per variable type will generate satisfactory 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 [0,1].

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

$$d(i,j) = \frac{\Sigma_{f=1}^{p}, \delta_{ij}^{(f)} d_{ij}^{(f)}}{\Sigma_{f=1}^{p} \delta_{ij}^{(f)}} \tag{8.17}$$

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, $d_{ij}^{(f)}$, is computed dependent on its type:

1. If f is binary or nominal: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$, otherwise $d_{ij}^{(f)} = 1$.
2. 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.
3. If f is ordinal or ratio-scaled: compute the ranks r_{if} and $z_{if} = \frac{r_{if}-1}{M_f - 1}$, and treat z_{if} as interval-scaled.

Thus, the dissimilarity between objects can be computed even when the variables describing the objects are of different types.

8.3 A categorization of major clustering methods

There exist a large number of clustering algorithms in the literature. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and 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.

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

1. **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** which 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 two popular heuristic methods: (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. For finding 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 8.4.

2. **Hierarchical methods**.

   A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the "*bottom-up*" approach, starts with each object forming a separate group. It successively merges the objects or groups close to one another, until all 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 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 worrying about a combinatorial number of different choices. However, a major problem of such techniques is that they cannot correct erroneous decisions.

   It can be advantageous to combine iterative relocation and hierarchical agglomeration by first using a hierarchical agglomerative algorithm and then refining the result using iterative relocation. Some scalable clustering algorithms, such as BIRCH and CURE, have been developed based on such an integrated approach. Hierarchical clustering methods are studied in Section 8.5.

3. **Density-based methods**.

   Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty 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, i.e., 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 is a typical density-based method which grows clusters according to a density threshold. OPTICS is a density-based method which computes an augmented clustering ordering for automatic and interactive cluster analysis. Density-based clustering methods are studied in Section 8.6.

4. **Grid-based methods**.

   Grid-based methods quantize the object space into a finite number of cells which 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. CLIQUE and Wave-Cluster are two clustering algorithms which are both grid-based and density-based. Grid-based clustering methods are studied in Section 8.7.

5. **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. Model-based clustering methods are studied in Section 8.8.

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 which require the integration of several clustering techniques.

Algorithm 8.4.1 (k-means) The k-means algorithm for partitioning based on the mean value of the objects in the cluster.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the squared-error criterion.

Method:
1) arbitrarily choose k objects 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 8.1: The k-means algorithm.

In the following sections, we examine each of the above five clustering methods in detail. We also introduce algorithms which integrate the ideas of several clustering methods. Outlier analysis, which typically involves clustering, is described in Section 8.9.

8.4 Partitioning methods

Given a database 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, often called a *similarity function*, such as distance, so that the objects within a cluster are "similar", whereas the objects of different clusters are "dissimilar" in terms of the database attributes. Recall that Equation (??) can be used to transform between dissimilarity and similarity coefficients.

8.4.1 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 intra-cluster similarity is high whereas the inter-cluster 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 "*center of gravity*".

"How does the k−means algorithm work?" It proceeds as follows. First, it randomly selects k of the objects which initially each represent 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 **squared-error criterion** is used, defined as

$$E = \Sigma_{i=1}^{k} \Sigma_{p \in C_i} |p - m_i|^2, \tag{8.18}$$

where E is the sum of square-error for all objects in the database, p is the point in space representing the given object, and m_i is the mean of cluster C_i (both p and m_i are multidimensional). This criterion tries to make the resulting k clusters as compact and as separate as possible. The k-means procedure is summarized in Figure 8.1.

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

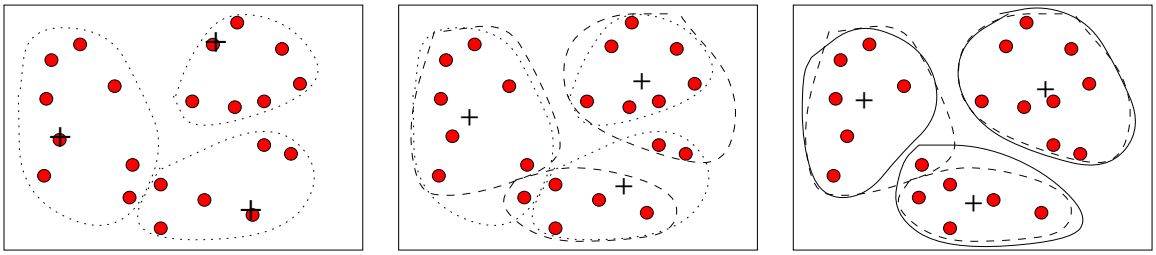

Figure 8.2: Clustering of a set of objects based on the k-means method. (The mean of each cluster is marked by a "+").

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 non-convex shapes, or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

Example 8.2 Suppose that there are a set of objects located in space as depicted in the rectangle shown in Figure 8.2a). Let $k = 3$, that is, the user would like to cluster the objects into three clusters.

According to Algorithm 8.4.1, 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 8.2a).

This kind of grouping will update the cluster centers. That is, the mean value of each cluster is recalculated based on the objects in the cluster. Relative to these new centers, objects are re-distributed to the cluster domains based on which cluster center is the nearest. Such a re-distribution forms new silhouettes encircled by dashed curves, as shown in Figure 8.2b).

This process iterates, leading to Figure 8.2c). Eventually, no re-distribution of the objects in any cluster occurs and so the process terminates. The resulting clusters are returned by the clustering process. □

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 which often yields good results is to first apply a hierarchical agglomeration algorithm to determine the number of clusters and to find an initial classification, and then use iterative relocation to improve the classification.

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 using 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, resulting in the **k-prototypes** method.

The **EM** (Expectation Maximization) algorithm extends the k-means paradigm in a different way: Instead of assigning each object to a dedicated cluster, it assigns each object to a cluster according to a weight representing the 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 effort on 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 which must be retained in main memory, thereby turning a secondary-memory based algorithm into a main memory-based algorithm.

Figure 8.3: Four cases of the cost function for k-medoids clustering.

Representative object-based technique: The k-medoids method

The k-means algorithm is sensitive to outliers since an object with an extremely large value may substantially distort the distribution of data.

"How might the algorithm be modified to diminish such sensitivity?", you may wonder.

Instead of taking the mean value of the objects in a cluster as a reference point, the **medoid** can be used, which is the most centrally located object in a cluster. Thus the partitioning method can still be performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. This forms the basis of the **k-medoids method**.

The basic strategy of k-medoids clustering algorithms is to find k clusters in n objects by first arbitrarily finding a representative object (the medoid) for each cluster. Each remaining object is clustered with the medoid to which it is the most similar. The strategy then iteratively replaces one of the medoids by one of the non-medoids as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function which measures the average dissimilarity between an object and the medoid of its cluster. To determine whether a non-medoid object, o_random, is a good replacement for a current medoid, o_j, the following four cases are examined for each of the non-medoid objects, p.

- **Case 1**: p currently belongs to medoid o_j. If o_j is replaced by o_{random} as a medoid and p is closest to one of o_i, $i \neq j$, then p is re-assigned to o_i.
- **Case 2**: p currently belongs to medoid o_j. If o_j is replaced by o_{random} as a medoid and p is closest to o_{random}, then p is re-assigned to o_{random}.
- **Case 3**: p currently belongs to medoid o_i, $i \neq j$. If o_j is replaced by o_{random} as a medoid and and p is still closest to o_i, then the assignment does not change.
- **Case 4**: p currently belongs to medoid o_i, $i \neq j$. If o_j is replaced by o_{random} as a medoid and p is closest to o_{random}, then p is re-assigned to o_{random}.

Figure 8.3 illustrates the four cases. Each time a re-assignment occurs, a difference in square-error E is contributed to the cost function. Therefore, the cost function calculates the *difference* in square- error value if a current medoid is replaced by a non-medoid object. The total cost of swapping is the sum of costs incurred by all non-medoid objects. If the total cost is negative, then o_j is replaced or swapped with o_{random} since the actual square-error E would be reduced. If the total cost is positive, the current medoid o_j is considered acceptable, and nothing is changed in the iteration. A general k-medoids algorithm is presented in Figure 8.4.

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

"Which method is more robust — $k-means$ or $k-medians$?" 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.

8.4.2 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 partition 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.

Algorithm 8.4.2 (k-medoids) A general k-medoids algorithm for partitioning based on medoid or central objects.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the sum of the dissimilarities of all the objects to their nearest medoid.

Method:

```
1)  arbitrarily choose k objects as the initial medoids;
2)  repeat
3)        assign each remaining object to the cluster with the nearest medoid;
4)        randomly select a non-medoid object, o_random;
5)        compute the total cost, S, of swapping o_j with o_random;
6)        if S < 0 then swap o_j with o_random to form the new set of k medoids;
7)  until no change;
```

□

Figure 8.4: The k-medoids algorithm.

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 sampled medoid is not among the best k medoids. For example, if an object o_i is one of the medoids in the best k medoids but it is not selected during sampling, CLARA will never find the best clustering. This is, therefore, a tradeoff for efficiency. A good clustering based on samples 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. The clustering process can be presented as searching a graph where every node is a potential solution, i.e., a set of k medoids. The clustering obtained after replacing a single medoid is called the *neighbor* of the current clustering. The number of neighbors to be randomly tried is restricted by a user-specified parameter. If a better neighbor is found (i.e., having a lower square-error), CLARANS moves to the neighbor's node and the process starts again; otherwise the current clustering produces a local optimum. If the local optimum is found, CLARANS starts with new randomly selected nodes in search for a new local optimum.

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 performance of CLARANS can be further improved by exploring spatial data structures, such as R*-trees, and some focusing techniques.

8.5 Hierarchical methods

A hierarchical clustering method works by grouping data objects into a tree of clusters. Hierarchical clustering methods can be further classified into *agglomerative* and *divisive* hierarchical clustering, depending on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion. The quality of a pure hierarchical clustering

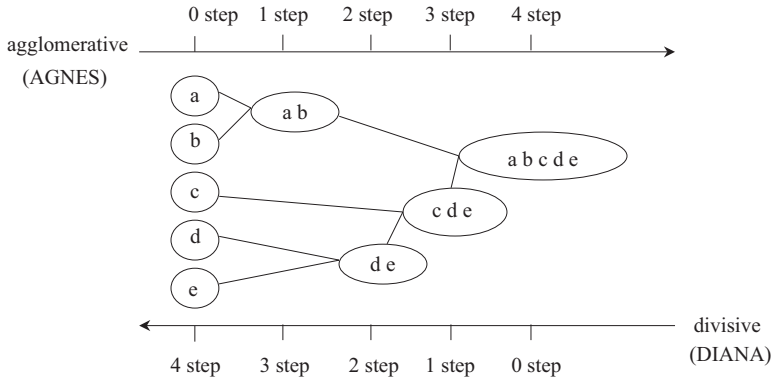

Figure 8.5: Agglomerative and divisive hierarchical clustering on data objects $\{a, b, c, d, e\}$.

method suffers from its inability to perform adjustment once a merge or split decision has been executed. Recent studies have emphasized the integration of hierarchical agglomeration with iterative relocation methods.

8.5.1 Agglomerative and divisive hierarchical clustering

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

1. **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 clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of inter-cluster similarity.

2. **Divisive hierarchical clustering**: This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the distance between the two closest clusters is above a certain threshold distance.

Example 8.3 Figure 8.5 shows the application of **AGNES** (AGglomerative NESting), an agglomerative hierarchical clustering method, and **DIANA** (DIvisia ANAlysis), a divisive hierarchical clustering method, to a data set of five objects, $\{a, b, c, d, e\}$. Initially, AGNES places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters C_1 and C_2 may be merged if an object in C_1 and an object in C_2 form the minimum Euclidean distance between any two objects from different clusters. This is a **single-link** approach in that each cluster is represented by all of the objects in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. The cluster merging process repeats until all of the objects are eventually merged to form one cluster.

In DIANA, all of the objects are used to form one initial cluster. The cluster is split according to some principle, such as the maximum Euclidean distance between the closest neighboring objects in the cluster. The cluster splitting process repeats until, eventually, each new cluster contains only a single object. □

In either agglomerative or divisive hierarchical clustering, one can specify the desired number of clusters as a termination condition.

Four widely-used measures for distance between clusters are as follows, where m_i is the mean for cluster C_i, n_i is the number of objects in C_i, and $|p - p'|$ is the distance between two objects or points p and p'.

- **Minimum distance**: $d_{min}(C_i, C_j) = min_{p \in Ci, p' \in C_j} |p - p'|$
- **Maximum distance**: $d_{max}(C_i, C_j) = max_{p \in Ci, p' \in C_j} |p - p'|$

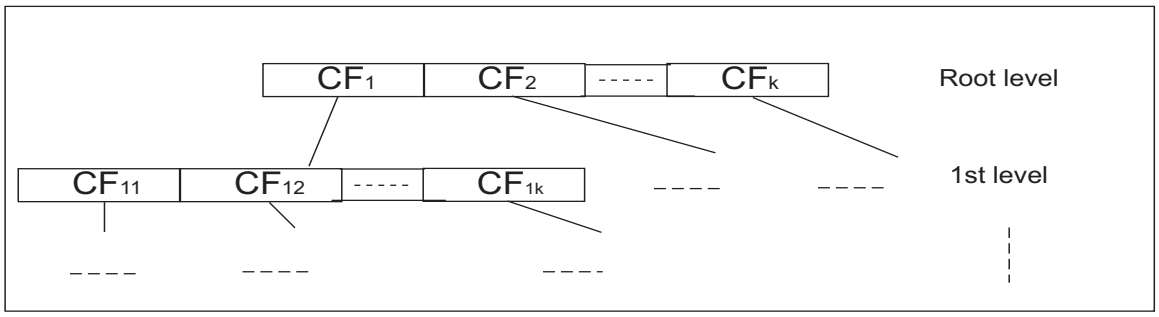

Figure 8.6: A CF-tree structure.

- **Mean distance**: $d_{mean}(C_i, C_j) = |m_i - m_j|$
- **Average distance**: $d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \Sigma_{p \in C_i} \Sigma_{p' \in C_j}$

"What are some of the difficulties with hierarchical clustering?" The hierarchical clustering method, though simple, often encounters difficulties regarding the selection of merge or split points. Such a decision is critical because once a group of objects is merged or split, the process at the next step will operate on the newly generated clusters. It will neither undo what was done previously, nor perform object swapping between clusters. Thus merge or split decisions, if not well chosen at some step, may lead to low quality clusters. Moreover, the method does not scale well since the decision of merge or split needs to examine and evaluate a good number of objects or clusters.

One promising direction for improving the clustering quality of hierarchical methods is to integrate hierarchical clustering with other clustering techniques for multiple phase clustering. A few such methods are introduced in the following subsections. The first, called BIRCH, begins by partitioning objects hierarchically using tree structures, and then applies other clustering algorithms to refine the clusters. The second, called CURE, represents each cluster by a certain fixed number of representative objects and then shrinks them toward the center of the cluster by a specified fraction. The third, called ROCK, merges clusters based on their inter-connectivity. The fourth, called CHAMELEON, explores dynamic modeling in hierarchical clustering.

8.5.2 BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an integrated hierarchical clustering method. It introduces two concepts, that of **clustering feature** and **clustering feature tree** (**CF tree**), which are used to summarize cluster representations. These structures help the clustering method achieve good speed and scalability in large databases. BIRCH is also effective for incremental and dynamic clustering of incoming objects.

Let's have a closer look at the above-mentioned structures. A **clustering feature** (**CF**) is a triplet summarizing information about subclusters of objects. Given N d-dimensional points or objects $\{o_i\}$ in a subcluster, then the CF of the subcluster is defined as

$$CF = (N, \vec{LS}, SS), \tag{8.19}$$

where N is the number of points in the subcluster, $\vec{LS}$ is the linear sum on N points, i.e., $\sum_{i=1}^{N} \vec{o_i}$, and SS is the square sum of data points, i.e., $\sum_{i=1}^{N} \vec{o_i}^2$.

A clustering feature is essentially a summary of the statistics for the given subcluster: the zero-th, first, and second moments of the subcluster from a statistical point of view. It registers crucial measurements for computing clusters and utilizes storage efficiently since it summarizes the information about the subclusters of objects instead of storing all objects.

A **CF tree** is a height-balanced tree which stores the clustering features for a hierarchical clustering. An example is shown in Figure 8.6. By definition, a non-leaf node in a tree has descendents or "children". The non-leaf 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 non-leaf node. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters influence the size of the resulting tree.

"How does the BIRCH algorithm work?" It consists of two phases:

- **Phase 1**: BIRCH scans the database to build an initial in-memory CF tree, which can be viewed as a multilevel compression of the data that tries to preserve the inherent clustering structure of the data.
- **Phase 2**: BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF-tree.

For Phase 1, the CF tree is built dynamically as objects are inserted. Thus, the method is incremental. An object is inserted to 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 it is passed towards 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 smaller 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 of 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.

After the CF tree is built, any clustering algorithm, such as a typical partitioning algorithm, can be used with the CF tree in Phase 2.

BIRCH tries to produce the best clusters with the available resources. With a limited amount of main memory, an important consideration is to minimize the time required for I/O. BIRCH applies a multiphase clustering technique: A single scan of data set yields a basic good clustering, and one or more additional scans can (optionally) be used to further improve the quality. The computation complexity of the algorithm is $O(n)$, where n is the number of objects to be clustered.

"How effective is BIRCH?" 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.

8.5.3 CURE: Clustering Using REpresentatives

Most clustering algorithms either favor clusters with spherical shape and similar sizes, or are fragile in the presence of outliers. **CURE** (Clustering Using REpresentatives) is an alternative method which integrates hierarchical and partitioning algorithms. It overcomes the problem of favoring clusters with spherical shape and similar sizes, and is more robust with respect to outliers.

CURE employs a novel hierarchical clustering algorithm that adopts a middle ground between the agglomerative (bottom-up) and divisive (top-down) approaches. Instead of using a single centroid or object to represent a cluster, a fixed number of representative points in space are chosen. The representative points of a cluster are generated by first selecting well-scattered objects for the cluster and then "shrinking" or moving them toward the cluster center by a specified fraction, or *shrinking factor*. At each step of the algorithm, the two clusters with the closest pair of representative points (where each point in the pair is from a different cluster) are merged.

Having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes. The shrinking or condensing of clusters helps dampen the effects of outliers. Therefore, CURE is more robust to outliers and identifies clusters having non-spherical shapes and wide variance in size. It scales well for large databases without sacrificing clustering quality.

To handle large databases, CURE employs a combination of random sampling and partitioning: a random sample is first partitioned, and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters.

The major steps of the CURE algorithm are briefly outlined as follows:

1. Draw a random sample, S, containing s objects.
2. Partition sample S into p partitions, each of size s/p.
3. Partially cluster the partitions into s/pq clusters, for some $q > 1$.

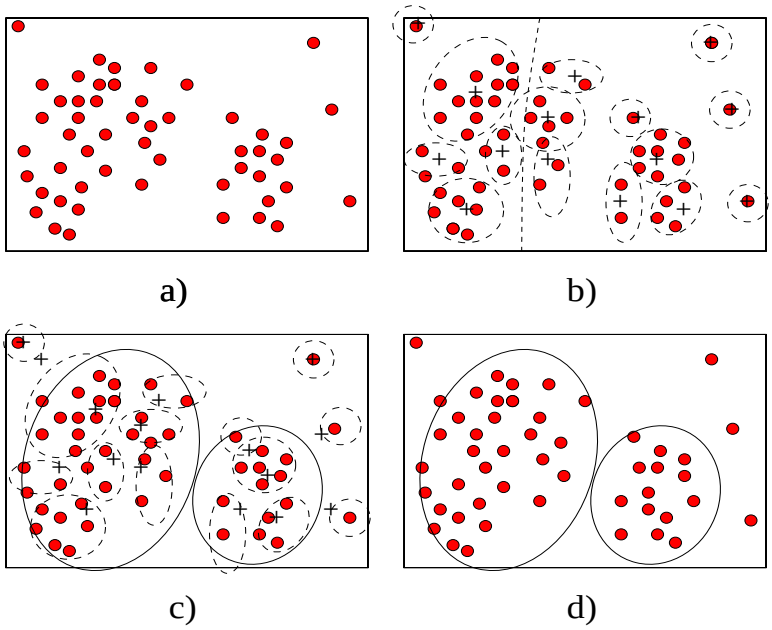

Figure 8.7: Clustering of a set of points (or objects) by CURE. a) A random sample of objects. b) The objects are partitioned, and partially clustered. Representative points for each cluster are marked by a "+". c) The partial clusters are further clustered. For each new cluster, the representative points are "shrinked" or moved towards the cluster center. d) The final clusters are of non-spherical shape.

4. Eliminate outliers by random sampling. If a cluster grows too slowly, remove it.

5. Cluster the partial clusters. The representative points falling in each newly-formed cluster are "shrinked" or moved towards the cluster center by a user-specified fraction, or shrinking factor, α. These points then represent and capture the shape of the cluster.

6. Mark the data with the corresponding cluster labels.

Let's look at an example of how CURE works.

Example 8.4 Imagine that there are a set of points (or objects) located in a rectangular region. Suppose that you would like to cluster the objects into two clusters, therefore $p = 2$.

First, $s = 52$ objects are sampled as shown in Figure 8.7a). These objects are then distributed into two partitions, each containing $52/2 = 26$ points. Suppose $q = 2$. We partially cluster the partitions into $52/(2 \times 2) = 13$ clusters based on minimal mean distance. The partial clusters are indicated by dashed curves, as shown in Figure 8.7b). Each cluster representative is marked by a "+". The partial clusters are further clustered, resulting in the two clusters illustrated by solid curves in Figure 8.7c). Each new cluster is "shrinked" or condensed by moving its representative points towards the cluster center by a fraction, α. The representative points capture the shape of each cluster. Thus, the initial objects are partitioned into two clusters, with the outliers excluded, as shown in Figure 8.7d). □

CURE produces high quality clusters in the existence of outliers, allowing clusters of complex shapes and different sizes. The algorithm requires one scan of the entire database. Given n objects, the complexity of CURE is of $O(n)$. *"How sensitive is CURE to its user-specified parameters, such as the sample size, number of desired clusters, and shrinking fraction, α?"* A sensitivity analysis showed that although some parameters can be varied without impacting the quality of clustering, the parameter setting in general does have a significant influence on the results.

ROCK is an alternative agglomerative hierarchical clustering algorithm that, unlike CURE, is suited for clustering categorical attributes. It measures the similarity of two clusters by comparing the *aggregate inter-connectivity* of two clusters against a user-specified static *inter-connectivity model*, where the **inter-connectivity** of two clusters C_1 and C_2 is defined by the number of *cross links* between the two clusters, and $link(p_i, p_j)$ is the number of common

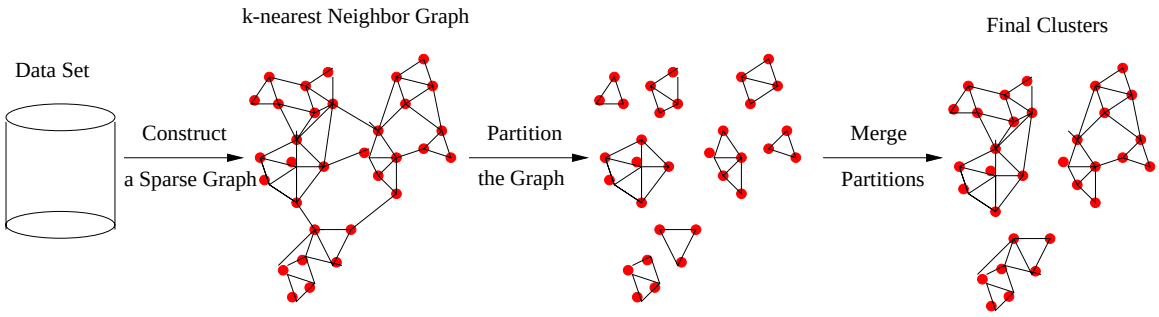

Figure 8.8: CHAMELEON: Hierarchical clustering based on k-nearest neighbors and dynamic modeling.

neighbors between two points p_i and p_j. In other words, cluster similarity is based on the number of points from different clusters who have neighbors in common.

ROCK first constructs a sparse graph from a given data similarity matrix using a similarity threshold and the concept of shared neighbors. It then performs a hierarchical clustering algorithm on the sparse graph.

8.5.4 CHAMELEON: A hierarchical clustering algorithm using dynamic modeling

CHAMELEON is a clustering algorithm which explores dynamic modeling in hierarchical clustering. In its clustering process, two clusters are merged if the inter-connectivity and closeness (proximity) between two clusters are highly related to the internal inter-connectivity and closeness of objects within the clusters. The merge process based on the dynamic model facilitates the discovery of natural and homogeneous clusters, and applies to all types of data as long as a similarity function is specified.

CHAMELEON is derived based on the observation of the weakness of two hierarchical clustering algorithms: CURE and ROCK. CURE and related schemes ignore information about the aggregate inter-connectivity of objects in two different clusters; whereas ROCK and related schemes ignore information about the closeness of two clusters while emphasizing their inter-connectivity.

"How does CHAMELEON work?" CHAMELEON first uses a graph partitioning algorithm to cluster the data objects into a large number of relatively small subclusters. It then uses an agglomerative hierarchical clustering algorithm to find the genuine clusters by repeatedly combining these clusters. To determine the pairs of most similar subclusters, it takes into account both the inter-connectivity as well as the closeness of the clusters, especially the internal characteristics of the clusters themselves. Thus it does not depend on a static, user-supplied model, and can automatically adapt to the internal characteristics of the clusters being merged.

Let's look at CHAMELEON in closer detail. As shown in Figure 8.8, CHAMELEON represents its objects based on the commonly used k-nearest neighbor graph approach. Each vertex of the k-nearest neighbor graph represents a data object, and there exists an edge between two vertices (objects), if one object is among the k-most similar objects of the other. The k-nearest neighbor graph G_k captures the concept of neighborhood dynamically: the neighborhood radius of an object is determined by the *density* of the region in which this object resides. In a dense region, the neighborhood is defined narrowly; in a sparse region, it is defined more widely. This tends to result in more natural clusters, in comparison with density-based methods such as DBSCAN (described in Section 8.6) which instead uses a global neighborhood. Moreover, the density of the region is recorded as the weight of the edges. That is, the edge of a dense region tends to weigh more than that of a sparse region.

CHAMELEON determines the similarity between each pair of clusters C_i and C_j according to their *relative inter-connectivity* $RI(C_i, C_j)$, and their *relative closeness* $RC(C_i, C_j)$.

The **relative inter-connectivity** $RI(C_i, C_j)$ between two clusters C_i and C_j is defined as the absolute inter-connectivity between C_i and C_j, normalized with respect to the internal inter-connectivity of the two clusters C_i and C_j. That is,

$$RI(C_i, C_j) = \frac{|EC_{\{C_i, C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)}, \tag{8.20}$$

where $EC_{\{C_i, C_j\}}$ is the *edge-cut* of the cluster containing both C_i and C_j so that the cluster is broken into C_i and

C_j, and similarly, EC_{C_i} (or EC_{C_j}) is the *size of its min-cut bisector* (i.e., the weighted sum of edges that partition the graph 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

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}, \tag{8.21}$$

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).

It has been shown that CHAMELEON has more power at discovering arbitrarily-shaped clusters of high quality than CURE and DBSCAN. However, the processing cost for high-dimensional data may require $O(n^2)$ time for n objects in the worst case.

8.6 Density-based methods

To discover clusters with arbitrary shape, density-based clustering methods have been developed. These typically regard clusters as dense regions of objects in the data space which are separated by regions of low density (representing noise).

8.6.1 DBSCAN: A density-based clustering method based on connected regions with sufficiently high density

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. The algorithm grows regions with sufficiently high density into clusters, and discovers clusters of arbitrary shape in spatial databases with noise. It defines a cluster as a maximal set of *density-connected* points.

The basic ideas of density-based clustering involve a number of new definitions. We intuitively present these definitions, and then follow up with an example.

1. The neighborhood within a radius ϵ of a given object is called the **ϵ-neighborhood** of the object.

2. If the ϵ-neighborhood of an object contains at least a minimum number, $MinPts$, of objects, then the object is called a **core object**.

3. Given a set of objects, D, we say that an object p is **directly density-reachable** from object q if p is within the ϵ-neighborhood of q, and q is a core object.

4. An object p is **density-reachable** from object q with respect to ϵ and $MinPts$ in a set of objects, D, if there is a chain of objects $p_1, \ldots, p_n$, $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i with respect to ϵ and $MinPts$, for $1 \leq i \leq n$, $p_i \in D$.

5. An object p is **density-connected** to object q with respect to ϵ and $MinPts$ in a set of objects, D, if there is an object $o \in D$ such that both p and q are density-reachable from o with respect to ϵ and $MinPts$.

Density reachability is the transitive closure of direct density reachability, and this relationship is asymmetric. Only core objects are mutually density reachable. Density connectivity, however, is a symmetric relation.

Example 8.5 Consider Figure 8.9 for a given ϵ represented by the radius of the circles, and, say, let $MinPts = 3$. Based on the above definitions:

- Of the labeled points, M, P, O, and R are core objects since each is in an ϵ-neighborhood containing at least three points.

- M is directly density-reachable from P, and Q is directly density-reachable from M.

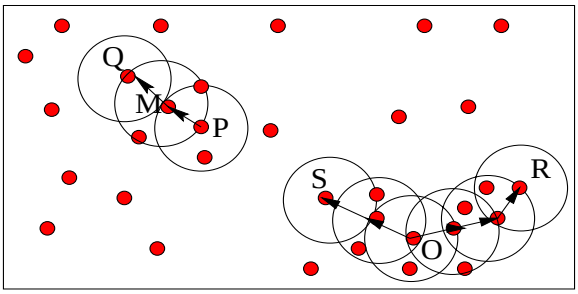

Figure 8.9: Density reachability and density connectivity in density-based clustering.

- Based on the previous observation, Q is (indirectly) density-reachable from P. However, P is not density-reachable from Q. Similarly, R and S are density-reachable from O.
- O, R and S are all density-connected.

□

A **density-based cluster** is a set of density-connected objects that is maximal with respect to density-reachability. Every object not contained in any cluster is considered to be *noise*.

"How does DBSCAN find clusters?" DBSCAN checks the ϵ-neighborhood of each point in the database. If the ϵ-neighborhood of a point p contains more than $MinPts$, a new cluster with p as a core object is created. DBSCAN then iteratively collects directly density-reachable objects from these core objects, which may involve the merge of a few density-reachable clusters. The process terminates when no new point can be added to any cluster.

The computational complexity of DBSCAN is $O(n\ log\ n)$, where n is the number of database objects. The algorithm is sensitive to the user-defined parameters. DBSCAN is further discussed in the following section, which compares it to an alternative density-based clustering method called OPTICS.

8.6.2 OPTICS: Ordering Points To Identify the Clustering Structure

Although DBSCAN (the density-based clustering algorithm described in Section 8.6.1) can cluster objects given input parameters such as ϵ and $MinPts$, it still leaves the user with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. Actually, this is a problem associated with many other clustering algorithms. Such parameter settings are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms are very sensitive to such parameter values: slightly different settings may lead to very different clusterings of the data. Moreover, high-dimensional real data sets often have very skewed distributions. There does not even exist a global parameter setting for which the result of a clustering algorithm may accurately describe the intrinsic clustering structure.

To help overcome this difficulty, a cluster ordering method called **OPTICS** (Ordering Points To Identify the Clustering Structure) was proposed. Rather than produce a data set clustering explicitly, OPTICS computes an augmented *cluster ordering* for automatic and interactive cluster analysis. This ordering represents the density-based clustering structure of the data. It contains information which is equivalent to density-based clustering obtained from a wide range of parameter settings.

By examining DBSCAN, one can easily see that for a constant $MinPts$-value, density-based clusters with respect to a higher density (i.e., a lower value for ϵ) are *completely contained* in density-connected sets obtained with respect to a lower density. Recall that the parameter ϵ is a distance – it is the neighborhood radius. Therefore, in order to produce a set or ordering of density-based clusters, we provide a set of distance parameter values. To construct the different clusterings simultaneously, the objects should be 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, two values need to be stored for each object – *core-distance* and *reachability-distance*:

- The **core-distance** of an object p is the smallest ϵ' value that makes p a core object. If p is not a core object, the core-distance of p is undefined.

- The **reachability-distance** of an object p and another object o is the greater value of the core-distance of p and the Euclidean distance between p and q. If p is not a core object, the reachability-distance between p and q is undefined.

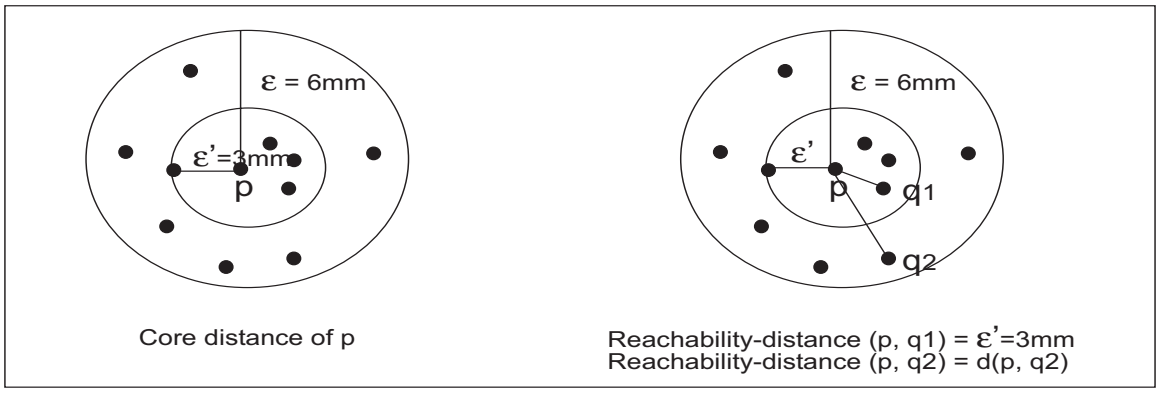

Figure 8.10: OPTICS terminology.

Example 8.6 Figure 8.10 illustrates the concepts of core-distance and reachability-distance. Suppose that $\epsilon = 6$mm and $MinPts = 5$. The core-distance of p is the distance, ϵ', between p and the fourth closest data object. □

"How are these values used?" The OPTICS algorithm creates an ordering of the objects in a database, additionally storing the core-distance and a suitable reachability-distance for each object. Such information is sufficient for the extraction of all density-based clusterings with respect to any distance ϵ' that is smaller than the distance ϵ used in generating the order.

The cluster ordering of a data set can be represented graphically, which helps in its understanding. For example, Figure 8.11 is the reachability plot for a simple 2-dimensional data set, which presents a general overview of how the data are structured and clustered. Methods have also been developed for viewing clustering structures of high-dimensional data.

Because of the structural equivalence of the OPTICS algorithm to DBSCAN, the OPTICS algorithm has the same run-time complexity as that of DBSCAN, i.e., $O(n\ log\ n)$. Spatial indexing structures can be used to further enhance its performance.

8.6.3 DENCLUE: Clustering based on density distribution functions

DENCLUE (DENsity-based CLUstEring) is a clustering method based on a set of density distribution functions. The method is built on the following ideas: (1) the influence of each data point can be formally modeled using a

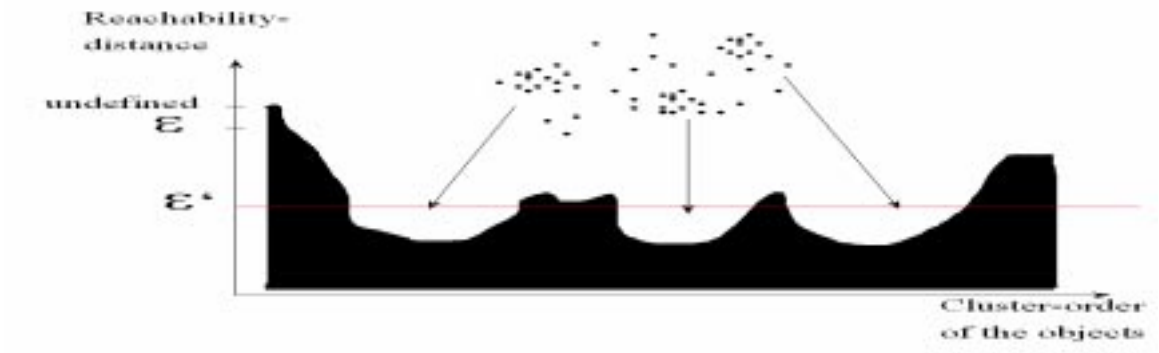

Figure 8.11: Cluster ordering in OPTICS.

mathematical function, called an *influence function*, which describes the impact of a data point within its neighborhood; (2) the overall density of the data space can be modeled analytically as the sum of the influence function of all data points; and (3) clusters can then be determined mathematically by identifying *density attractors*, where density attractors are local maxima of the overall density function.

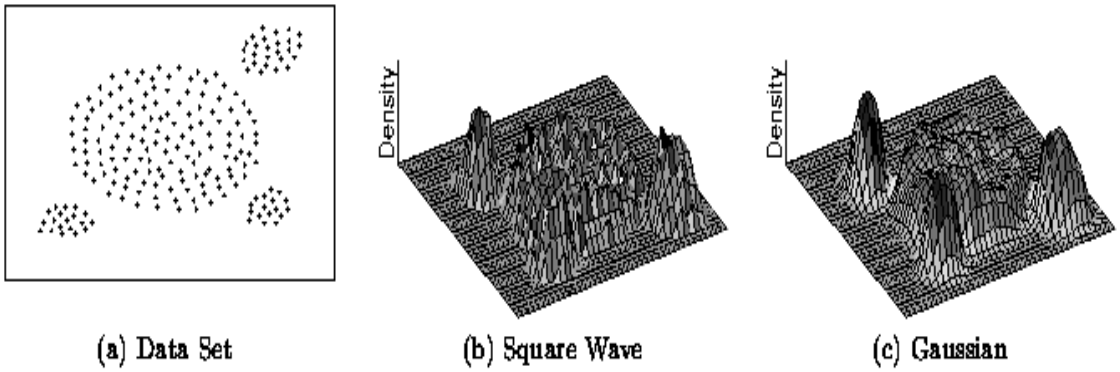

Figure 8.12: Possible density functions for a 2-D data set. Figure is from [14].

Let x and y be objects in F^d, a d-dimensional feature space. The **influence function** of data object y on x is a function $f_B^y : F^d \to R_0^+$, which is defined in terms of a basic influence function f_B,

$$f_B^y(x) = f_B(x, y). \tag{8.22}$$

In principle, the influence function can be an arbitrary function that can determine the distance between two objects in a neighborhood. It should be reflexive and symmetric, such as the Euclidean distance function (Section 8.2.1), a *square wave influence function*,

$$f_{Square}(x, y) = \begin{cases} 0 & if\ d(x, y) > \sigma \\ 1 & otherwise \end{cases} \tag{8.23}$$

or a *Gaussian influence function*,

$$f_{Gauss}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}. \tag{8.24}$$

The **density function** at an object $x \in F^d$ is defined as the sum of influence functions of all data points. Given n data objects, $D = \{x_1, \ldots, x_n\} \subset F^d$, the density function at x is defined as

$$f_B^D(x) = \sum_{i=1}^{n} f_B^{x_i}(x). \tag{8.25}$$

For example, the density function that results from the Gaussian influence function (8.24) is

$$f_{Gaussian}^D(x) = \sum_{i=1}^{n} e^{-\frac{d(x,x_i)^2}{2\sigma^2}} \tag{8.26}$$

From the density function, one can define the *gradient* of a function, and the *density attractor* which is the local maxima of the overall density function. For a continuous and differentiable influence function, a hill climbing algorithm guided by the gradient, can be used to determine the density attractor of a set of data points. Figure 8.12 shows square wave and Gaussian density functions for a 2-D data set.

Based on these notions, both *center-defined cluster* and *arbitrary-shape cluster* can be formally defined. A **center-defined cluster** is a subset C that is *density-extracted*, i.e., with its density function value no less than a threshold ξ; otherwise (i.e., if its density function value is less than ξ), it is considered an outlier. An **arbitrary-shape cluster** is a set of C's, each being density-extracted, with the density function value no less than a threshold ξ, and where

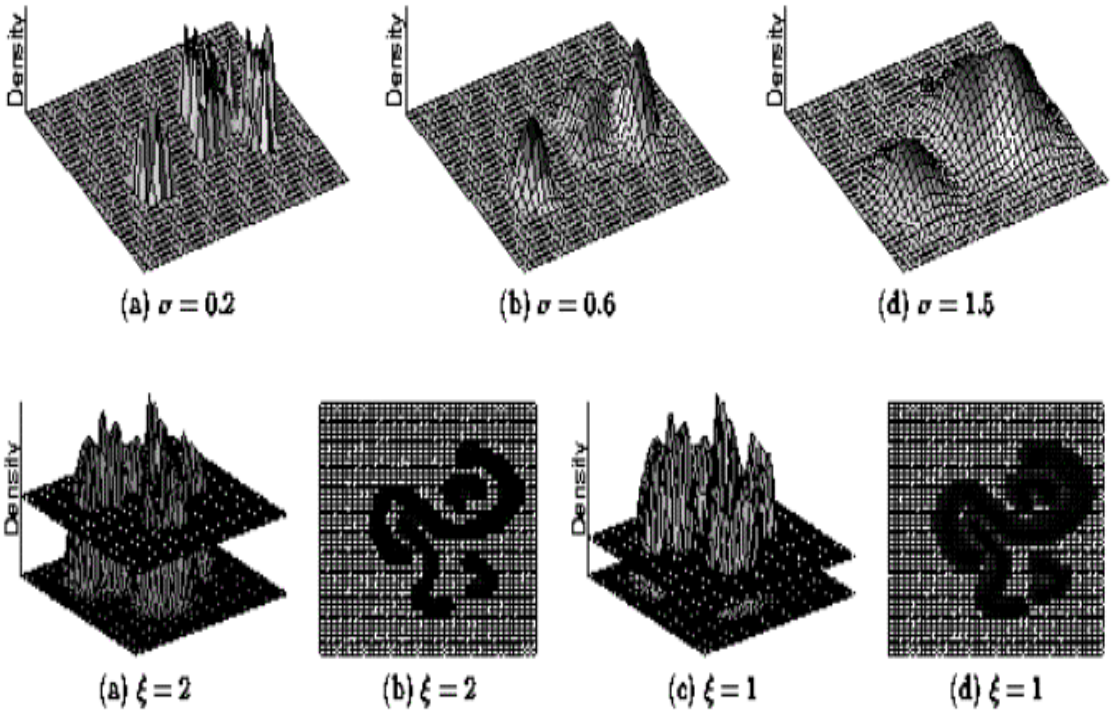

Figure 8.13: Examples of center-defined clusters (top row) and arbitrary-shaped clusters (bottom row). Figure is from [14].

there exists a path P from each region to another, and the density function value for each point along the path is no less than ξ. Examples of center-defined and arbitrary-shape clusters are shown in Figure 8.13.

"What major advantages does DENCLUE have in comparison with other clustering algorithms, in general?" There are several: (1) it has a solid mathematical foundation, and generalizes other clustering methods, including partition-based, hierarchical, and locality-based methods, (2) it has good clustering properties for data sets with large amounts of noise, (3) it allows a compact mathematical description of arbitrarily shaped clusters in high-dimensional data sets, and (4) it uses grid cells yet only keeps information about grid cells that do actually contain data points. It manages these cells in a tree-based access structure, and thus is significantly faster than some influential algorithms, such as DBSCAN (by a factor of up to 45). However, the method requires careful selection of the density parameter σ and noise threshold ξ, as the selection of such parameters may significantly influence the quality of the clustering results.

8.7 Grid-based methods

The grid-based clustering approach uses a multiresolution grid data structure. It quantizes the space into a finite number of cells which 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.

Some typical example of the grid-based approach include STING, which explores statistical information stored in the grid cells; WaveCluster, which clusters objects using a wavelet transform method; and CLIQUE, which represents a grid- and density-based approach for clustering in high-dimensional data space.

8.7.1 STING: A STatistical INformation Grid approach

STING (STatistical INformation Grid) is a grid-based multiresolution clustering technique in which the spatial area

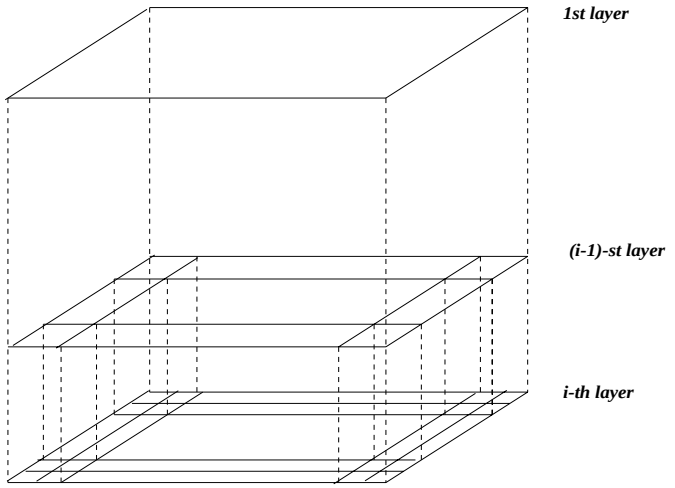

Figure 8.14: A hierarchical structure for STING clustering.

is divided into rectangular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution, and these cells 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) are precomputed and stored. These statistical parameters are useful for query processing, as described below.

Figure 8.14 shows a hierarchical structure for STING clustering. 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 attribute-dependent parameters, m (mean), s (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). When the data are loaded into the database, the parameters, *count*, m, s, *min*, *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 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?" It can be used in a top-down, grid-based method 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 range of probability) 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 requirements of the query.

"What advantages does STING offer over other clustering methods?" There are several: (1) the grid-based computation is *query-independent* since 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 in $O(n)$, 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.

Since STING uses a multiresolution approach to perform 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 the parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all of 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.

8.7.2 WaveCluster: Clustering using wavelet transformation

WaveCluster is a multiresolution clustering algorithm which first summarizes the data by imposing a multidimensional grid structure onto the data space. It then uses a *wavelet transformation* to transform the original feature space, finding dense regions in the transformed space.

In this approach, each grid cell summarizes the information of a group of points which map into the cell. This summary information typically fits into main memory for use by the multiresolution wavelet transform and the subsequent cluster analysis.

"But what is a wavelet transform?"

A **wavelet transform** is a signal processing technique that decomposes a signal into different frequency subbands. The wavelet model can be applied to n-dimensional signals by applying a one-dimensional wavelet transform n times. In applying a wavelet transform, data are converted from the spatial domain (where data are points in a feature or attribute space) into the *frequency* domain (showing the data at different scales or resolutions) so that the natural clusters in the data become more distinguishable. Clusters can then be identified by searching for dense regions in the new domain. Wavelet transforms are also discussed in Chapter 3, where they are used for data reduction by compression. Additional references to the technique are given in the bibliographic notes.

"Why is wavelet transformation useful for clustering?" It offers the following advantages.

- First, it provides unsupervised clustering. It uses hat-shape filters which emphasize regions where the points cluster, while at the same time, suppressing weaker information outside of the cluster boundaries. Thus, dense regions in the original feature space act as attractors for nearby points, and as inhibitors for points that are further away. This means that the clusters in the data automatically stand out and "clear" the regions around them. Thus, another advantage is that wavelet transformation can automatically result in the removal of outliers.

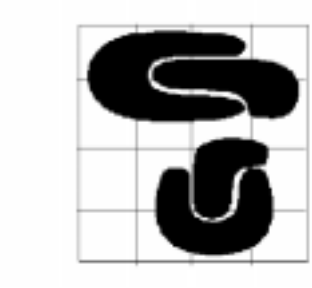

Figure 8.15: A sample of 2-dimensional feature space. Figure is from [30].

- The multiresolution property of wavelet transformations can help in the detection of clusters at varying levels of accuracy. For example, Figure 8.15 shows a sample of 2-dimensional feature space, where each point in the image represents the attribute or feature values of one object in the spatial data set. Figure 8.16 shows the resulting wavelet transformation at different resolutions, from a fine scale (scale 1) to a coarse scale (scale 3). At each level, the four sub-bands into which the original data are decomposed are shown. The sub-band shown at the upper-left quadrant emphasizes the average neighborhood around each data point . The sub-band at the upper-right quadrant emphasizes the horizontal edges of the data. The sub-band at the lower-left quadrant emphasizes the vertical edges, while the sub-band at the lower-right quadrant emphasizes the corners.
- Wavelet-based clustering is very fast, with a computational complexity of $O(n)$ where n is the number of objects in the database. The algorithm implementation can be made parallel.

WaveCluster is a grid-based and density-based algorithm. It conforms with many of the requirements of a good clustering algorithm: It handles large data sets efficiently, discovers clusters with arbitrary shape, successfully handles

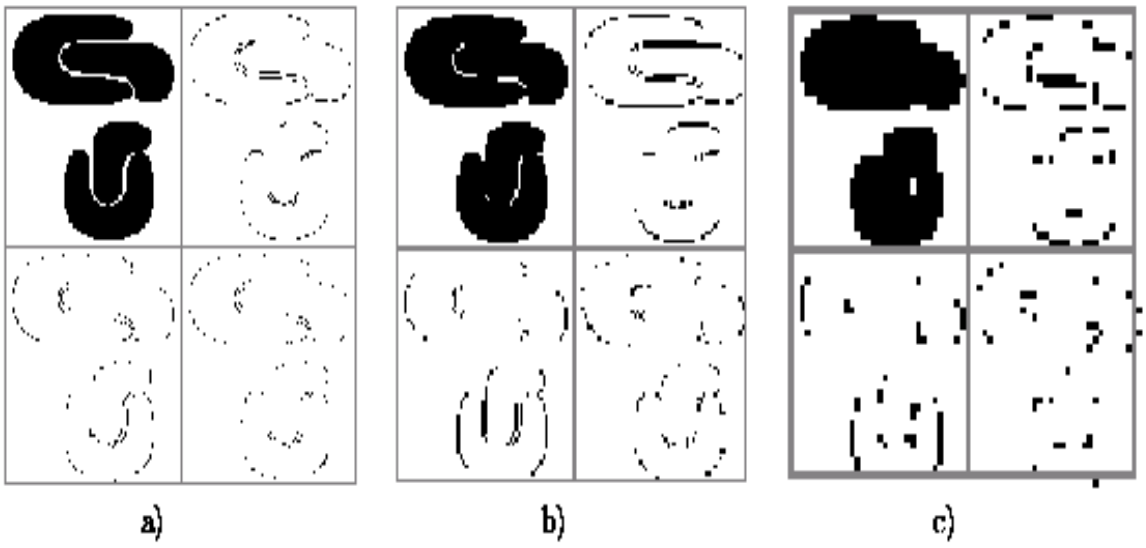

Figure 8.16: Multiresolution of the feature space in Figure 8.15 at a) scale 1 (high resolution); b) scale 2 (medium resolution); c) scale 3 (low resolution). Figure is from [30].

outliers, is insensitive to the order of input, and does not require the specification of input parameters such as the number of clusters or a neighborhood radius. In experimental studies, WaveCluster was found to outperform BIRCH, CLARANS and DBSCAN in terms of both efficiency and clustering quality. However, WaveCluster is only applicable to low-dimensional data.

8.7.3 CLIQUE: Clustering high-dimensional space

The CLIQUE (CLustering In QUEst) clustering algorithm integrates density-based and grid-based clustering. It is useful for clustering high dimensional data in large databases.

CLIQUE is based on the following:

- Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. CLIQUE's clustering identifies the sparse and the "crowded" *areas in space* (or **units**), thereby discovering the overall distribution patterns of the data set.
- A unit is **dense** if the fraction of total data points contained in it exceeds an input model parameter. In CLIQUE, a cluster is defined as a maximal set of *connected dense units*.

"How does CLIQUE work?" CLIQUE performs multidimensional clustering in two steps:

1. First, CLIQUE partitions the n-dimensional data space into non-overlapping rectangular units, identifying the dense units among these. This is done (in 1-D) for each dimension. For example, Figure 8.17 shows dense rectangular units found with respect to *Age* for the dimensions *Salary* and (number of weeks of) *Vacation*. The subspaces representing these dense units are intersected to form a *candidate* search space in which dense units of higher dimensionality may exist.

   "How does CLIQUE search for dense units of higher dimensionality?" The identification of actual dense units within the candidate search space is based on the *Apriori property* used in association rule mining[1]. In general, the property employs prior knowledge of items in the search space so that portions of the space can be pruned.

[1] Association rule mining is described in detail in Chapter 6. In particular, the Apriori property is described in Section 6.2.1.

The property, adapted for CLIQUE, states the following: *If a k-dimensional unit is dense, then so are its projections in $(k-1)$-dimensional space.* That is, given a k-dimensional candidate dense unit, if we check its $(k-1)$-th projection units and find any that are not dense, then we know that the k-th dimensional unit cannot be dense either. Therefore, we can generate potential or candidate dense units in $k-$dimensional space from the dense units found in $(k-1)$-dimensional. In general, the resulting space searched is much smaller than the original space. The dense units are then examined in order to determine the clusters.

2. CLIQUE generates a minimal description for each cluster as follows. For each cluster, it determines the maximal region that covers the cluster of connected dense units. It then determines a minimal cover for each cluster.

"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 tuples and does not presume any canonical data distribution. It scales linearly with the size of input and has good scalability as the number of dimensions in the data is increased. However, the accuracy of the clustering result may be degraded at the expense of the simplicity of the method.

8.8 Model-based clustering methods

Model-based clustering methods attempt to optimize the fit between the given data and some mathematical model. Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions. Model-based clustering methods follow two major approaches: a *statistical approach* or a *neural network approach*. Examples of each approach are described in this section.

8.8.1 Statistical approach

Conceptual clustering is a form of clustering in machine learning which, given a set of unlabeled objects, produces a classification scheme over the objects. Unlike conventional clustering which primarily identifies groups of like objects, conceptual clustering goes one step further by also finding characteristic descriptions for each group, where each group represents a concept or class. Hence, conceptual clustering is a two-step process: first, clustering is performed, followed by characterization. Here, clustering quality is not solely a function of the individual objects. Rather, it incorporates factors such as the generality and simplicity of the derived concept descriptions.

Most methods of conceptual clustering adopt a statistical approach which uses probability measurements in determining the concepts or clusters. Probabilistic descriptions are typically used to represent each derived concept.

COBWEB is a popular and simple method of incremental conceptual clustering[2]. Its input objects are described by categorical attribute-value pairs. COBWEB creates a hierarchical clustering in the form of a **classification tree**.

"But, what is a classification tree? Is it the same as a decision tree?"

Figure 8.18 shows a classification tree for animal data, based on Fisher (1987). A classification tree differs from a decision tree. Each node in a classification tree refers to a concept, and contains a probabilistic description of that concept which summarizes the objects classified under the node. The probabilistic description includes the probability of the concept, and conditional probabilities of the form $P(A_i = V_{ij}|C_k)$, where $A_i = V_{ij}$ is an attribute-value pair and C_k is the concept class. (Counts are accumulated and stored at each node for computation of the probabilities). This is unlike decision trees, which label branches rather than nodes and use logical rather than probabilistic descriptors[3]. The sibling nodes at a given level of a classification tree are said to form a **partition**. To classify an object using a classification tree, a partial matching function is employed to descend the tree along a path of 'best' matching nodes.

COBWEB uses a heuristic evaluation measure called *category utility* to guide construction of the tree. **Category utility (CU)** is defined as

$$\frac{\Sigma_{k=1}^{n} P(C_k)[\Sigma_i \Sigma_j P(A_i = V_{ij}|C_k)^2 - \Sigma_i \Sigma_j P(A_i = V_{ij}^2]}{n}, \tag{8.27}$$

[2] It is incremental in that it takes one object at a time as input, rather than all objects at once.
[3] Decision trees are described in Chapter 7 on Classification and Prediction.

where n is the number of nodes, concepts, or "categories" forming a partition $\{C_1, C_2, .., C_n\}$ at the given level of the tree. In other words, category utility is the increase in the expected number of attribute values that can be correctly guessed given a partition (where this expected number is represented by the first term of the numerator in Equation 8.27) over the expected number of correct guesses with no such knowledge (where this number is the second term of the numerator). Category utility rewards inter-class similarity and intra-class *dis*similarity where:

- **inter-class similarity** is the probability $P(C_k|A_i = V_{ij})$. The larger this value is, the fewer the objects in contrasting classes that share this attribute-value pair, and the more predictive the pair is of the class.
- **intra-class similarity** is the probability $P(A_i = V_{ij}|C_k)$. The larger this value is, the greater the proportion of class members sharing the value, and the more predictable the value is of class members;

Let's have a look at how COBWEB works. COBWEB incrementally incorporates objects into a classification tree.

"Given a new object," you wonder, *"how does COBWEB decide where to incorporate it into the classification tree?"* COBWEB descends the tree along an appropriate path, updating counts along the way, in search of the 'best host' or node at which to classify the object. This decision is based on temporarily placing the object in each node, and computing the category utility of the resulting partition. The placement which results in the highest category utility should be a good host for the object.

"Hmm, but what if the object does not really belong to any of the concepts represented in the tree so far? What if it is better to create a new node for the given object?"

That is a good point. In fact, COBWEB also computes the category utility of the partition that would result if a new node were to be created for the object. This is compared to the above computation based on the existing nodes. The object is then placed in an existing class, or a new class is created for it, based on the partition with the highest category utility value. Notice that COBWEB has the ability to automatically adjust the number of classes in a partition. It does not need to rely on the user to provide such an input parameter.

The two operators mentioned above are highly sensitive to the input order of the object. COBWEB has two additional operators which help make it less sensitive to input order. These are **merging** and **splitting**. When an object is incorporated, the two best hosts are considered for merging into a single class. Furthermore, COBWEB considers splitting the children of the best host among the existing categories. These decisions are based on category utility. The merging and splitting operators allow COBWEB to perform a bidirectional search — e.g., a merge can undo a previous split.

"What are the limitations of COBWEB?" The method has several. First, it is based on the assumption that probability distributions on separate attributes are statistically independent of one other. This assumption is, however, not always true since correlation between attributes often exists. Moreover, the probability distribution representation of clusters makes it quite expensive to update and store the clusters. This is especially so when the attributes have a large number of values since their time and space complexities depend not only on the number of attributes, but also on the number of values for each attribute. Furthermore, the classification tree is not height-balanced for skewed input data, which may cause the time and space complexity to degrade dramatically.

CLASSIT is an extension of COBWEB for incremental clustering of continuous (or real valued) data. It stores a continuous normal distribution (i.e., mean and standard deviation) for each individual attribute in each node and uses a modified category utility measure which is an integral over continuous attributes instead of a sum over discrete attributes as in COBWEB. However, it suffers similar problems as COBWEB and thus is not suitable for clustering large database data. Additional research is needed in the application of conceptual clustering methods to data mining.

8.8.2 Neural network approach

The neural network approach to clustering tends to represent each cluster as an *exemplar*. An exemplar acts as a "prototype" of the cluster and does not necessarily have to correspond to a particular data example or object. New objects can be distributed to the cluster whose exemplar is the most similar, based on some distance measure. The attributes of an object assigned to a cluster can be predicted from the attributes of the cluster's exemplar.

In this section, we discuss two prominent methods of the neural network approach to clustering. The first in *competitive learning*, and the second is *self-organizing feature maps*, both of which involve competing neural units.

Competitive learning involves an hierarchical architecture of several units (or artificial "neurons") which compete in a "winner-takes-all" fashion for the object that is currently being presented to the system. Figure 8.19 shows an example of a competitive learning system. Each circle represents a unit. The winning unit within a cluster becomes *active* (indicated by a filled circle), while the others are *inactive* (indicated by empty circles). Connections between layers are *excitatory* – a unit in a given layer can receive inputs from all of the units in the next lower level. The configuration of active units in a layer represents the input pattern to the next higher layer. The units within a cluster at a given layer compete with one another to respond to the pattern that is output from the layer below. Connections within layers are *inhibitory* so that only one unit in any given cluster may be active. The winning unit adjusts the weights on its connections between other units in the cluster so that it will respond even more strongly to future objects that are the same or similar to the current one. If we view the weights as defining an exemplar, then new objects are assigned to the cluster with the closest exemplar. The number of clusters, and of units per cluster are input parameters.

At the end of the clustering (or any clustering, in general), each cluster can be thought of as a new "feature" that detects some regularity in the objects. Thus, the resulting clusters can be viewed as a mapping of low-level features to higher-level features.

With **self-organizing feature maps (SOMs)**, clustering is also performed by having several units compete for the current object. The unit whose weight vector is closest to the current object becomes the winning or active unit. So as to move even closer to the input object, the weights of the winning unit are adjusted, as well as those of its nearest neighbors. SOMs assume that there is some topology or ordering among the input objects, and that the units will eventually take on this structure in space. The organization of units is said to form a feature map. SOMs are believed to resemble processing that can occur in the brain.

The neural network approach to clustering has strong theoretical links with actual brain processing. Further research is required in making it readily applicable to large databases due to long processing times and the intricacies of complex data.

8.9 Outlier analysis

"What is an outlier?"

Very often, there exist data objects that do not comply with the general behavior or model of the data. Such data objects, which are grossly different from or inconsistent with the remaining set of data, are called **outliers**.

Outliers can be caused by measurement or execution error. For example, the display of a person's age as -999 could be caused by a program default setting of an unrecorded age. Alternatively, outliers may be the result of inherent data variability. The salary of the chief executive officer of a company, for instance, could naturally stand out as an outlier among the salaries of the other employees in the firm.

Many data mining algorithms try to minimize the influence of outliers, or eliminate them all together. This, however, could result in the loss of important hidden information since "one person's noise could be another person's signal". In other words, the outliers themselves may be of particular interest, such as in the case of fraud detection, where outliers may indicate fraudulent activity. Thus, outlier detection and analysis is an interesting data mining task, referred to as **outlier mining**.

Outlier mining has wide applications. As mentioned above, it can be used in fraud detection, e.g., by detecting unusual usage of credit cards or telecommunication services. In addition, it is useful in customized marketing for identifying the spending behavior of customers with extremely low or extremely high incomes, or in medical analysis for finding unusual responses to various medical treatments.

Outlier mining can be described as follows: Given a set of n data points or objects, and k, the expected number of outliers, find the top k objects which are considerably dissimilar, exceptional, or inconsistent with respect to the remaining data. The outlier mining problem can be viewed as two subproblems: (1) define what data can be considered as inconsistent in a given data set; and (2) find an efficient method to mine the outliers so defined.

The problem of defining outliers is nontrivial. If a regression model is used for data modeling, analysis of the residuals can give a good estimation for data "extremeness". The task becomes tricky, however, when finding outliers in time series data as they may be hidden in trend, seasonal, or other cyclic changes. When multidimensional data are analyzed, not any particular one, but rather, a *combination* of dimension values may be extreme. For non-numeric (i.e., categorical data), the definition of outliers requires special consideration.

"What about using data visualization methods for outlier detection?", you may wonder.

This may seem to be an obvious choice, since human eyes are very fast and effective at noticing data inconsistencies. However, this does not apply to data containing cyclic plots, where apparently outlying values could be perfectly valid values in reality. Data visualization methods are weak in detecting outliers in data with may categorical attributes, or in data of high-dimensionality since human eyes are good at visualizing numeric data of only two to three dimensions.

In this section, we instead examine computer-based methods for outlier detection. These can be categorized into three approaches: the *statistical approach*, the *distance-based approach*, and the *deviation-based approach*, each of which are studied here. Notice that while clustering algorithms discard outliers as noise, they can be modified to include outlier detection as a byproduct of their execution. In general, users must check that each outlier discovered by these approaches is indeed a "real" outlier.

8.9.1 Statistical-based outlier detection

The statistical approach to outlier detection assumes a distribution or probability model for the given data set (e.g., a normal distribution) and then identifies outliers with respect to the model using a *discordancy test*. Application of the test requires knowledge of the data set parameters (such as the assumed data distribution), knowledge of distribution parameters (such as the mean and variance), and the expected number of outliers.

"How does the discordancy testing work?" A statistical discordancy test examines two hypotheses: a *working hypothesis* and an *alternative hypothesis*. A **working hypothesis**, H, is a statement that the entire data set of n objects comes from an initial distribution model, F, i.e.,

$$H : o_i \in F, where\ i = 1, 2, \ldots, n.$$

The hypothesis is retained if there is no statistically significant evidence supporting its rejection. A **discordancy test** verifies whether an object o_i is significantly large (or small) in relation to the distribution F. Different test statistics have been proposed for use as a discordancy test, depending on the available knowledge of the data. Assuming that some statistic T has been chosen for discordancy testing, and the value of the statistic for object o_i is v_i, then the distribution of T is constructed. Significance probability $SP(v_i) = Prob(T > v_i)$ is evaluated. If some $SP(v_i)$ is sufficiently small, then o_i is discordant and the working hypothesis is rejected. An **alternative hypothesis**, $\overline{H}$, which states that o_i comes from another distribution model, G, is adopted. The result is very much dependent on which F model is chosen since o_i may be an outlier under one model, and a perfectly valid value under another.

The alternative distribution is very important in determining the power of the test, i.e. the probability that the working hypothesis is rejected when o_i is really an outlier. There are different kinds of alternative distributions.

- **Inherent alternative distribution**: In this case, the working hypothesis that all of the objects come from distribution F is rejected in favor of the alternative hypothesis that all of the objects arise from another distribution, G:

  $$\overline{H} : o_i \in G, where\ i = 1, 2, \ldots, n.$$

  F and G may be different distributions, or differ only in parameters of the same distribution. There are constraints on the form of the G distribution in that it must have potential to produce outliers. For example, it may have a different mean or dispersion, or a longer tail.

- **Mixture alternative distribution**: The mixture alternative states that discordant values are not outliers in the F population, but contaminants from some other population. In this case, the alternative hypothesis is:

  $$\overline{H} : o_i \in (1 - \lambda)F + \lambda G, where\ i = 1, 2, \ldots, n.$$

- **Slippage alternative distribution**: This alternative states that all of the objects (apart from some prescribed small number) arise independently from the initial model F with parameters μ and σ^2, while the remaining objects are independent observations from a modified version of F in which the parameters have been shifted.

There are two basic types of procedures for detecting outliers:

- **Block procedures**: In this case, either all of the suspect objects are treated as outliers, or all of them are accepted as consistent.

- **Consecutive (or sequential) procedures**: An example of such a procedure is the *inside-out* procedure. Its main idea is that the object that is least "likely" to be an outlier is tested first. If it is found to be an outlier, then all of the more extreme values are also considered outliers; otherwise, the next most extreme object is tested, and so on. This procedure tends to be more effective than block procedures.

"How effective is the statistical approach at outlier detection?" A major drawback is that most tests are for single attributes, yet many data mining problems require finding outliers in multidimensional space. Moreover, the statistical approach requires knowledge about parameters of the data set, such as the data distribution. However, in many cases, the data distribution may not be known. Statistical methods do not guarantee that all outliers will be found for the cases where no specific test was developed, or the observed distribution cannot be adequately modeled with any standard distribution.

8.9.2 Distance-based outlier detection

The notion of distance-based outliers was introduced to counter the main limitations imposed by statistical methods.

"What is a distance-based outlier?"

An object o in a data set S is a **distance-based (DB) outlier** with parameters p and d, i.e., $DB(p, d)$, if at least a fraction p of the objects in S lie at a distance greater than d from o. In other words, rather than relying on statistical tests, we can think of distance-based outliers as those objects who do not have "enough" neighbors, where neighbors are defined based on distance from the given object. In comparison with statistical-based methods, distance-based outlier detection generalizes or unifies the ideas behind discordancy testing for standard distributions. Therefore, a distance-based outlier is also called a *unified outlier*, or *UO-outlier*. Distance-based outlier detection avoids the excessive computation that can be associated with fitting the observed distribution into some standard distribution and in selecting discordancy tests.

For many discordancy tests, it can be shown that if an object o is an outlier according to the given test, then o is also a $DB(p, d)$ outlier for some suitably defined p and d. For example, if objects that lie 3 or more standard deviations from the mean are considered to be outliers, assuming a normal distribution, then this definition can be "unified" by a $DB(0.9988, 0.13\sigma)$-outlier.

Several efficient algorithms for mining distance-based outliers have been developed. These are outlined as follows.

- **Index-based algorithm**: Given a data set, the index-based algorithm uses multidimensional indexing structures, such as R-trees or k-d trees, to search for neighbors of each object o within radius d around that object. Let M be the maximum number of objects within the d-neighborhood of an outlier. Therefore, once $M + 1$ neighbors of object o are found, it is clear that o is not an outlier. This algorithm has a worst case complexity of $O(k * n^2)$, where k is the dimensionality, and n is the number of objects in the data set. The index-based algorithm scales well as k increases. However, this complexity evaluation takes only the search time into account even though the task of building an index, in itself, can be computationally intensive.

- **Nested-loop algorithm**: The nested-loop algorithm has the same computational complexity as the index-based algorithm but avoids index structure construction and tries to minimize the number of I/O's. It divides the memory buffer space into two halves, and the data set into several logical blocks. By carefully choosing the order in which blocks are loaded into each half, I/O efficiency can be achieved.

- **Cell-based algorithm**: To avoid $O(n^2)$ computational complexity, a cell-based algorithm was developed for memory-resident data sets. Its complexity is $O(c^k + n)$, where c is a constant depending on the number of cells, and k is the dimensionality. In this method, the data space is partitioned into cells with a side length equal to $\frac{d}{2\sqrt{k}}$. Each cell has two *layers* surrounding it. The first layer is one cell thick, while the second is $2\sqrt{k}$ cells thick, rounded up to the closest integer. The algorithm counts outliers on a *cell-by-cell* rather than object-by-object basis. For a given cell, it accumulates three counts — the number of objects in the cell, in the cell and the first layer together, and in the cell and both layers together. Let's refer to these counts as *cell_count*, *cell_+_1_layer_count*, and *cell_+_2_layers_count*, respectively.

  "How are outliers determined in this method?" Let M be the maximum number of outliers that can exist in the d-neighborhood of an outlier.

- An object o in the current cell is considered an outlier only if *cell_+_1_layer_count* is less than or equal to M. If this condition does not hold, then all of the objects in the cell can be removed from further investigation as they cannot be outliers.
- If *cell_+_2_layers_count* is less than or equal to M, then *all* of the objects in the cell are considered outliers. Otherwise, if this number is more than M, then it is possible that some of the objects in the cell may be outliers. To detect these outliers, object-by-object processing is used where, for each object o in the cell, objects in the second layer of o are examined. For objects in the cell, only those objects having less than M $d-$neighbors in both their first and second layers are considered to be outliers.

A variation to the algorithm is linear with respect to n and guarantees that no more than three passes over the data set are required. It can be used for large, disk-resident data sets, yet does not scale well for high dimensions.

Distance-based outlier detection requires the user to set both the p and d parameters. Finding suitable settings for these parameters can involve much trial and error.

8.9.3 Deviation-based outlier detection

Deviation-based outlier detection does not use statistical tests or distance-based measures to identify exceptional objects. Instead, it identifies outliers by examining the main characteristics of objects in a group. Objects that "deviate" from this description are considered outliers. Hence, in this approach the term "deviations" is typically used to refer to outliers. In this section, we study two techniques for deviation-based outlier detection. The first sequentially compares objects in a set, while the second employs an OLAP data cube approach.

Sequential exception technique

The sequential exception technique simulates the way in which humans can distinguish unusual objects from among a series of supposedly-like objects. It uses implicit redundancy of the data. Given a set S of n objects, it builds a sequence of subsets, $\{S_1, S_2, .., S_m\}$, of these objects with $2 \leq m \leq n$ such that

$$S_{j-1} \subset S_j, \text{ where } S_j \subseteq S.$$

Dissimilarities are assessed between subsets in the sequence. The technique introduces the following key terms.

- **Exception set**: This is the set of deviations or outliers. It is defined as the smallest subset of objects whose removal results in the greatest reduction of dissimilarity in the residual set[4].
- **Dissimilarity function**: This function does not require a metrical distance between the objects. It is any function that, if given a set of objects, returns a low value if the objects are similar to one another. The greater the dissimilarity is among the objects, the higher is the value returned by the function. The dissimilarity of a subset is incrementally computed based on the subset prior to it in the sequence. Given a subset of n numbers $\{x_1, .., x_n\}$, a possible dissimilarity function is the variance of the numbers in the set, i.e.,

$$\frac{1}{n}\Sigma_{i=1}^{n}(x_i - \overline{x})^2, \tag{8.28}$$

  where $\overline{x}$ is the mean of the n numbers in the set. For character strings, the dissimilarity function may be in the form of a pattern string (e.g., containing wildcard characters) that is used to cover all of the patterns seen so far. The dissimilarity increases when the pattern covering all of the strings in S_{j-1} does not cover any string in S_j that is not in S_{j-1}.
- **Cardinality function**: This is typically the count of the number of objects in a given set.

[4]For interested readers, this is equivalent to the greatest reduction in *Kolmogorov complexity* for the amount of data discarded. Reference to Kolmogorov complexity is given in the bibliographic notes.

- **Smoothing factor**: This is a function that is computed for each subset in the sequence. It assess how much the dissimilarity can be reduced by removing the subset from the original set of objects. This value is scaled by the cardinality of the set. The subset whose smoothing factor value is the largest is the exception set.

The general task of finding an exception set can be NP-hard (i.e., intractable). A sequential approach is computationally feasible and can be implemented using a linear algorithm.

"How does this technique work?" Instead of assessing the dissimilarity of the current subset with respect to its complementary set, the algorithm selects a sequence of subsets from the set for analysis. For every subset, it determines the dissimilarity difference of the subset with respect to the *preceding* subset in the sequence.

"Can't the order of the subsets in the sequence affect the results?" To help alleviate any possible influence of the input order on the results, the above process can be repeated several times, each with a different random ordering of the subsets. The subset with the largest smoothing factor value, among all of the iterations, becomes the exception set.

"How effective is this method?" The above method has been shown to be dependent on the dissimilarity function used. The task of defining this function is complicated by the fact that the nature of the exceptions is not known in advance. The option of looking for a universal dissimilarity function was rejected based on experiments with real life databases. The algorithm has linear complexity of $O(n)$, where n is the number of objects in the input, provided that the number of iterations is not very big. This complexity is also based on the notion that the computation of the dissimilarity function is incremental. That is, the dissimilarity of a given subset in a sequence can be computed from that of the previous subset.

OLAP data cube technique

An OLAP approach to deviation detection uses data cubes to identify regions of anomalies in large multidimensional data. This technique was described in detail in Chapter 2. For added efficiency, the deviation detection process is overlapped with cube computation. The approach is a form of *discovery-driven exploration* where precomputed measures indicating data exceptions are used to guide the user in data analysis, at all levels of aggregation. A cell value in the cube is considered an exception if it is significantly different from the expected value, based on a statistical model. The expected value of a cell is considered to be a function of all of the aggregated computed using the cell value. If the cell involves dimensions for which concept hierarchies have been defined, then the expected value of the cell also depends on its ancestors in the hierarchies. The method uses visual cues such as background color to reflect the degree of exception of each cell. The user can choose to drill-down on cells that are flagged as exceptions. The measure value of a cell may reflect exceptions occurring at more detailed or *lower levels* of the cube, where these exceptions are not visible from the current level.

The model considers variations and patterns in the measure value across *all of the dimensions* to which a cell belongs. For example, suppose that you have a data cube for sales data, and are viewing the sales summarized per month. With the help of the visual cues, you notice an increase in sales in December in comparison to all other months. This may seem like an exception in the time dimension. However, by drilling-down on the month of December to reveal the sales per item in that month, you note that there is a similar increase in sales for other items during December. Therefore, an increase in total sales in December is not an exception if the item dimension is considered. The model considers exceptions hidden at all aggregated group-by's of a data cube. Manual detection of such exceptions is difficult since the search space is typically very large, particularly when there are many dimensions involving concept hierarchies with several levels.

8.10 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 wide **applications** including market or customer segmentation, pattern recognition, biological studies, spatial data analysis, Web document classification, and many others. Cluster analysis can be used as a stand-alone data mining tool to gain insight into the data distribution, or serve as a preprocessing step for other data mining algorithms operating on the detected clusters.

- The quality of clustering can be assessed based on a measure of **dissimilarity** of objects, which can be computed for **various types of data**, including *interval-scaled*, variables, *binary* variables, *nominal, ordinal*, and *ratio-scaled* variables, or combinations of these variable types.
- Clustering is a dynamic field of research in data mining, with a large number of clustering algorithms developed. These algorithms can be **categorized** into *partitioning methods, hierarchical methods, density-based methods, grid-based methods*, and *model-based methods*.
- A **partitioning method** first creates an initial k partition, where k is the number of partitions to construct, then it uses an *iterative relocation technique* which attempts to improve the partitioning by moving objects from one group to another. Typical partition methods include k-means, k-medoids, CLARANS, and their improvements.
- 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 the rigidity of merge or split, hierarchical agglomeration often integrates other clustering techniques, such as iterative relocation. Typical such methods include BIRCH, CURE, and Chameleon.
- A **density-based method** cluster objects based on the notion of density. It either grows clusters according to density of neighborhood objects (such as in DBSCAN) or according to some density function (such as in DENCLUE). Typical density-based method include DBSCAN, OPTICS, and DENCLUE.
- A **grid-based method** first quantizes the object space into a finite number of cells which 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 and Wave-Cluster are two clustering algorithms which are both grid-based and density-based.
- A **model-based method** hypothesizes a model for each of the clusters and finds the best fit of the data to that model. Typical model-based methods include statistical approach, such as AutoClass, COBWEB and CLASSIT, and neural network approach, such as SOM.
- One person's noise could be another person's signal. **Outlier detection and analysis** is very useful for fraud detection, customized marketing, medical analysis, and many other tasks. Computer-based outlier analysis methods typically follow either a *statistical approach*, a *distance-based approach*, or a *deviation-based approach*.

Exercises

1. What is clustering? How is it different from classification?
2. Briefly outline how to compute the dissimilarity between objects described by the following types of variables:
   (a) asymmetric binary variables
   (b) nominal variables
   (c) ratio-scaled variables
   (d) numerical (interval-scaled) variables
3. Given are the following measurements for the variable *age*:

   18, 22, 25, 42, 28, 43, 33, 35, 56, 28.

   Standardize the variable by the following:

   (a) Compute the mean absolute deviation of *age*.
   (b) Compute the z-score for the first four measurements.
4. Given two objects represented by the following tuples: $(22, 1, 42, 10)$, and $(20, 0, 36, 8)$,

   (a) Compute the *Euclidean distance* between the two objects.

(b) Compute the *Manhattan distance* between the two objects.

(c) Compute the *Minkowski distance* between the two objects, using $q = 3$.

5. Table 8.3 contains the attributes *name, gender, trait-1, trait-2, trait-3*, and *trait-4*, where *name* is an object-id, *gender* is a symmetric attribute, and the remaining *trait* attributes are asymmetric, describing personal traits of individuals who desire a penpal. Suppose that a service exists which attempts to find pairs of compatible penpals.

name	gender	trait-1	trait-2	trait-3	trait-4
Kevan	M	N	P	P	N
Caroline	F	N	P	P	N
Erik	M	P	N	N	P
⋮	⋮	⋮	⋮	⋮	⋮

Table 8.3: A relational table containing binary attributes for a penpal matching service.

For asymmetric attribute values, let the values Y and P be set to 1, and the value N be set to 0.

Suppose that the distance between objects (potential penpals) is computed based only on the asymmetric variables.

(a) Show the *contingency matrix* for each pair given Kevan, Caroline, and Erik.

(b) Compute the *simple matching coefficient* for each pair.

(c) Compute the *Jaccard coefficient* for each pair.

(d) Who do you suggest would make the best pair of penpals? Which pair of individuals would be the least compatible?

(e) Suppose that we are to include the symmetric variable *gender* in our analysis. Based on the Jaccard coefficient, who would be the most compatible pair, and why?

6. Briefly describe the following approaches to clustering methods: partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Give examples in each case.

7. Suppose that the data mining task is to cluster the following eight points (with (x, y) representing location) into 3 clusters.

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

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

(a) the three cluster centres after the first round execution, and

(b) the final three clusters.

8. Use a diagram to illustrate how, for a constant *MinPts* value, *density-based clusters* with respect to a higher density (i.e., a lower value for ϵ, the neighborhood radius) are completely contained in density-connected sets obtained with respect to a lower density.

9. Human eyes are fast and effective at judging the quality of clustering methods for two-dimensional data. Can you design a data visualization method which may help humans visualize data clusters and judge the clustering quality for three-dimensional data? What about for even higher dimensional data?

10. Give an example of how specific clustering methods may be *integrated*, e.g., where one clustering algorithm is used as a preprocessing step for another.

11. Clustering has been popularly recognized as an important data mining task with broad applications. Give one application example for each of the following cases:

(a) An application which takes clustering as a major data mining function.

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

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

13. 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 places of work may be clustered so that typically one ATM is assigned per cluster. The clustering, however, may be constrained by factors involving the location of bridges, rivers, and highways which can affect ATM accessibility. Additional constraints may involve limitations on the number of ATM's per district forming the region. Given such constraints, how can clustering algorithms be modified to allow for *constraint-based clustering*?

Bibliographic notes

Clustering methods are discussed in several textbooks, such as Jain and Dubes [16], and Kaufman and Rousseeuw [18]. Methods for combining variables of different types into a single dissimilarity matrix were proposed by Ducker et al. [?], and later extended by Kaufman and Rousseeuw [18].

For partitioning methods, the k-means algorithm was first introduced by MacQueen [24]. The k-medoids algorithms of PAM and CLARA were proposed by Kaufman and Rousseeuw [18]. The k-modes algorithm was proposed by Huang [15], while the EM (Expectation Maximization) algorithm was introduced by Lauritzen [22]. The CLARANS algorithm was proposed by Ng and Han [26]. Ester, Kriegel, and Xu [8] proposed techniques for further improvement of the performance of CLARANS using efficient spatial access methods, such as R*-tree and focusing techniques.

Agglomerative hierarchical clustering, such as AGNES, and divisive hierarchical clustering, such as DIANA, were introduced by Kaufman and Rousseeuw [18]. An interesting direction for improving the clustering quality of hierarchical clustering methods is to integrate hierarchical clustering with distance-based clustering, iterative relocation, or other nonhierarchical clustering methods. Typical studies in this direction include the BIRCH algorithm by Zhang, Ramakrishnan, and Livny [32], CURE by Guha, Rastogi, and Shim [12], ROCK (for clustering categorical attributes) by Guha, Rastogi, and Shim [13], and CHAMELEON by Karypis, Han, and Kumar [17].

For density-based clustering methods, DBSCAN was proposed by Ester, Kriegel, Sander, and Xu [7], while Ankerst, Breunig, Kriegel, and Sa [2] developed the cluster ordering method, Optics. DBSCLASD (Xu, Ester, Kriegel, and Sander [?]) is an alternative to DBSCAN which does not require any input parameters, yet is slightly slower than DBSCAN. The DENCLUE algorithm, based on a set of density distribution functions, was proposed by Hinneburg and Keim [14].

Grid-based approaches to clustering have recently been studied. STING is a grid-based multiresolution approach proposed by Wang, Yang, and Muntz [31]. WaveCluster, developed by Sheikholeslami, Chatterjee, and Zhang [30]. is a multiresolution clustering approach which transforms the original feature space by wavelet transform. CLIQUE, developed by Agrawal, Gehrke, Gunopulos, and Raghavan [1], is an integrated, density-based and grid-based clustering method for clustering high-dimensional data.

For a set of seminal papaers on model-based clustering, see Shavlik and Dietterich [29]. Conceptual clustering was first introduced by Michalski and Stepp [25]. Other examples of the statistical clustering approach include COBWEB by Fisher [9], CLASSIT by Gennari, Langley, and Fisher [11], and AutoClass by Cheeseman and Stutz [6]. Studies of the neural network approach include competitive learning by Rumelhart and Zipser [27], and SOM (self-organizing feature maps) by Kohonen [21].

Outlier detection and analysis can be categorized into three approaches: the statistical approach, the distance-based approach, and the deviation-based approach. The statistical approach and discordancy tests are described in Barnett and Lewis [4]. Distance-based outlier detection is described in Knorr and Ng [19, 20]. The sequential problem approach to deviation-based outlier detection was introduced in Arning, Agrawal, and Raghavan [3]. Sarawagi, Agrawal, and Megiddo [28] introduced a discovery-driven method for identifying exceptions in large multidimensional data using OLAP data cubes.

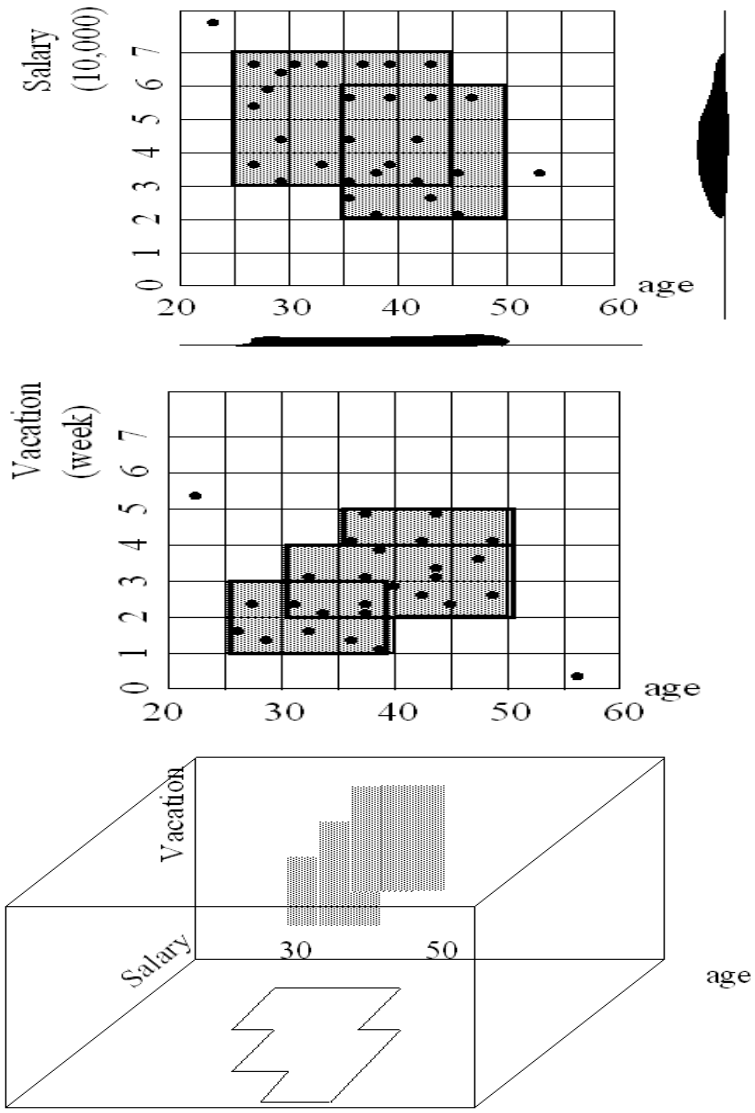

Figure 8.17: Dense units found with respect to *Age* for the dimensions *Salary* and *Vacation* are intersected in order to provide a candidate search space for dense units of higher dimensionality.

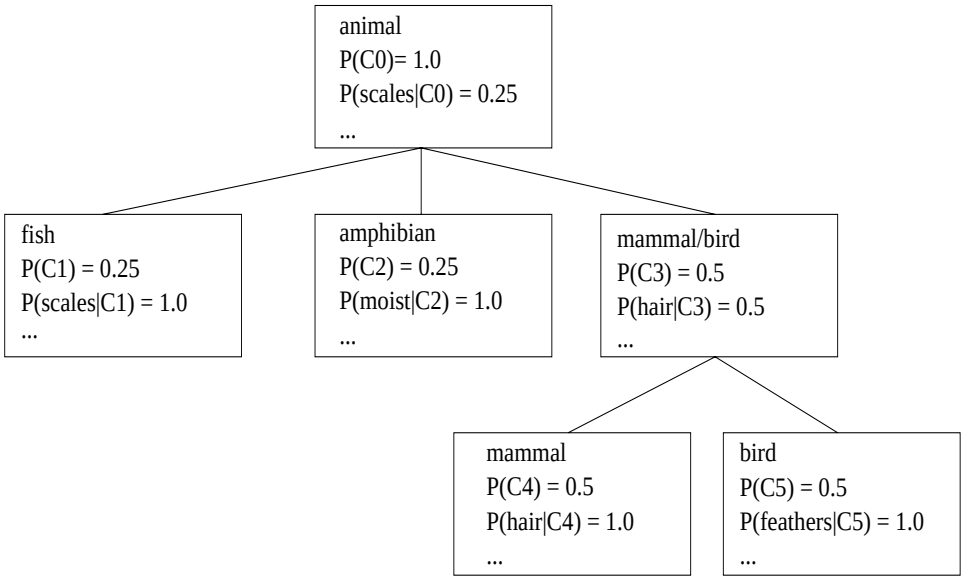

Figure 8.18: A classification tree.

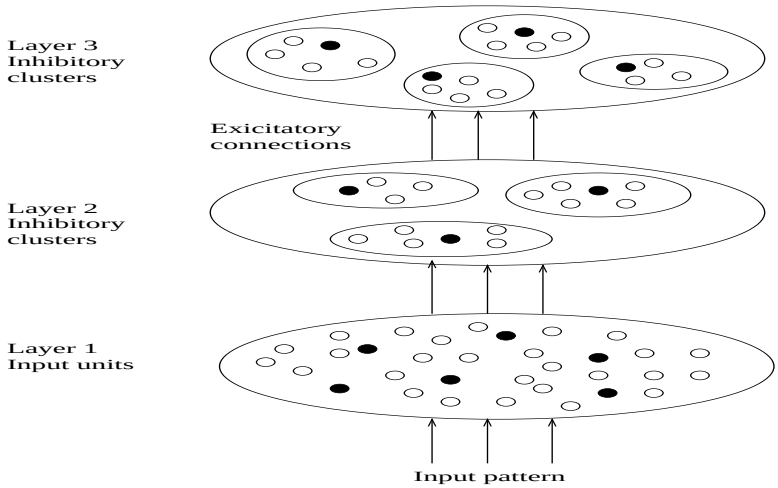

Figure 8.19: An architecture for competitive learning. The number of layers can be arbitrary.

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 94–105, Seattle, Washington, June 1998.

[2] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data*, pages 49–60, Philadelphia, PA, June 1999.

[3] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 164–169, Portland, Oregon, August 1996.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 322–331, Atlantic City, NJ, June 1990.

[6] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, Portland, Oregon, August 1996.

[8] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[9] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[10] D. Fisher. Optimization and simplification of hierarchical clusterings. In *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining (KDD'95)*, pages 118–123, Montreal, Canada, Aug. 1995.

[11] J. Gennari, P. Langley, and D. Fisher. Models of incremenral concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[12] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 73–84, Seattle, Washington, June 1998.

[13] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proc. 1999 Int. Conf. Data Engineering*, pages 512–521, Sydney, Australia, March 1999.

[14] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 58–65, New York, NY, August 1998.

[15] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2:283–304, 1998.

[16] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Printice Hall, 1988.

[17] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *COMPUTER*, 32:68–75, 1999.

[18] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[19] E. Knorr and R. Ng. A unified notion of outliers: Properties and computation. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 219–222, Newport Beach, California, August 1997.

[20] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 392–403, New York, NY, August 1998.

[21] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[22] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[23] M. Li and P. Vitanyi. *Applied Multivariate Statistical Analysis*. New York: Springer Verlag, 1991.

[24] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, 1:281–297, 1967.

[25] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach (Vol. 1)*. San Mateo, CA: Morgan Kaufmann, 1983.

[26] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[27] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[28] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proc. Int. Conf. of Extending Database Technology (EDBT'98)*, pages 168–182, Valencia, Spain, March 1998.

[29] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning*. Morgan Kaufmann, 1990.

[30] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 428–439, New York, NY, August 1998.

[31] W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proc. 1997 Int. Conf. Very Large Data Bases*, pages 186–195, Athens, Greece, Aug. 1997.

[32] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.