Contents

Chapter 8

Cluster Analysis

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, market research, etc. By clustering, one can identify crowded and sparse regions, and therefore, discover overall distribution patterns and interesting correlations among data attributes. 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 a young scientific discipline under vigorous development. There are a large number of research papers scattered in many conference proceedings and periodicals, mostly in the fields of data mining, statistics, machine learning, spatial database, biology, marketing, and so on, with different emphases and different techniques. 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, focussing mainly on distance-based cluster analysis. Cluster analysis tools based on k-means, k-medoids, and several other methods have also been built in many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS.

In machine learning, cluster analysis often refers to **unsupervised learning**. Unlike classification, clustering does not rely on predefined classes and class-labeled training examples. For this reason, it 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, people have been studying 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. How can we develop clustering algorithms that are highly scalable in large databases?

2. **Ability to deal with different types of attributes**: Many algorithms are designed to cluster interval-based (numerical) data. However, many 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. Many parameters are 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 the city's rivers and highway networks, and customer requirements per region as constraints. A challenging task is to find groups of data with good clustering behavior and satisfying various constraints.

9. **Interpretability and usability**: Users may 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 discuss variations of different methods.

8.2 Types of data in clustering analysis

In this section, we study the types of data which ofter 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 relational table, or n-by-p matrix (n objects × 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. 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 that 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.

Clusters of objects are computed based on their similarities or dissimilarities. In this section, we first discuss how the quality of clustering can be assessed based on *correlation coefficients*, which can be converted to *dissimilarity coefficients* or *similarity coefficients*. We then discuss how the dissimilarity of objects 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.

8.2.1 Dissimilarities and similarities: Measuring the quality of clustering

Measures of *dissimilarity* or *dissimilarity coefficients* can be used to measure the quality of clustering. In general, **dissimilarity** $d(i,j)$ is a nonnegative number that is close to 0 when i and j are near each other, and becomes larger when they are more different.

Dissimilarities can be obtained by simple subjective ratings made by a set of observers or experts on how much certain objects differ from each other. For example, in social science, one may rate how close one subject, such as mathematics, is to another, such as biology, computer science, or physics. Alternatively, dissimilarities can be computed from correlation coefficients. Given n objects for clustering, the parametric **Pearson product-moment correlation** between two variables f and g is defined in (8.3), where f and g are variables describing the objects, m_f and m_g are the mean values of f and g, respectively, and x_{if} is the value of f for the ith object, and x_{ig} is the value of g for the ith object.

$$R(f,g) \quad = \quad \frac{\Sigma_{i=1}^{n}(x_{if} - m_f)(x_{ig} - m_g)}{\sqrt{\Sigma_{i=1}^{n}(x_{if} - m_f)^2}\sqrt{\Sigma_{i=1}^{n}(x_{ig} - m_g)^2}}. \tag{8.3}$$

The nonparametric *Spearman correlation* can also be used in some applications. Notice that R is used in statistics for multiple correlation or a matrix of correlations, while r is used for a correlation. Both coefficients are provided by most statistical packages, such as SPSS, SAS, and SPlus.

Conversion formula (8.4) can be used to compute dissimilarity coefficients, $d(f, g)$, from either parametric or nonparametric correlation coefficients, $R(f, g)$:

$$d(f,g) \quad = \quad (1 - R(f,g))/2. \tag{8.4}$$

Given variables with a high positive correlation, this assigns a dissimilarity coefficient that is close to zero. Variables with a strong negative correlation are assigned a dissimilarity coefficient that is close to 1 (meaning that the variables are very dissimilar).

In some applications, users may prefer to use conversion formula (8.5), where variables with a high positive or negative correlation are assigned the same high similarity value.

$$d(f,g) \quad = \quad 1 - |R(f,g)| \tag{8.5}$$

Alternatively, users may like to use a *similarity coefficient* $s(i, j)$ instead of a dissimilarity coefficient. Formula (8.6) can be used to transform between the two coefficients.

$$s(i,j) \quad = \quad 1 - d(i,j) \tag{8.6}$$

Notice that not all of the variables should be included in the clustering analysis. A variable that is meaningless to a given clustering is worse than useless, since it hides the useful information provided by other variables. For example, the telephone number of a person is often useless in the clustering of people according to their characteristics, such as age, height, weight, and so on. Such kind of "trash" variable should be given zero weight, excluding it from the clustering process.

8.2.2 Interval-scaled variables

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

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, such as changing 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, people may intentionally want to give more weight to a certain set of variables than others. For example, when clustering basketball player candidates, one may like to give more weight to the variable height.

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,

$$s_f \quad = \quad \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \cdots + |x_{nf} - m_f|) \tag{8.7}$$

where $x_{1f}, .., x_{nf}$ are n measures 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**, called **z-score**, as follows,

$$z_{if} \quad = \quad \frac{x_{if} - m_f}{s_f}. \tag{8.8}$$

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 users.

After standardization, or without standardization in certain applications, we compute the dissimilarity (or similarity) between the objects. Given interval-scaled variables, this is typically based on the distance between each pair of objects. There are a few approaches to defining the distance between objects. The most popular distance measure is **Euclidean distance**, which is defined as

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \cdots + |x_{ip} - x_{jp}|^2}. \tag{8.9}$$

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 by

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|. \tag{8.10}$$

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 point i to point j is no more than making a detour over any other point h.

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

$$d(i,j) = (|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \cdots + |x_{ip} - x_{jp}|^q)^{1/q}, \tag{8.11}$$

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.

$$d(i,j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_p|x_{ip} - x_{jp}|^2}. \tag{8.12}$$

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

8.2.3 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.

One approach is to compute 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 a is the number of variables that equal 1 for both objects i and j, b is the number of variables that equal 1 for object i but that are 0 for object j, c is the number of variables that equal 0 for object i but equal 1 for object j, and d is the number of variables that equal 0 for both objects i and j. The total number of variables is p, where $p = a + b + c + d$.

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 *male* and *female* in *gender*. Similarity based on symmetric binary variables is called **invariant similarity** in that the result does not object j

		1	0	sum
	1	a	b	$a+b$
object i	0	c	d	$c+d$
	sum	$a+c$	$b+d$	p

Table 8.1: A contingency table for binary variables.

change when some or all of the binary variables are coded differently. For invariant similarities, the most well-known coefficient is the **simple matching coefficient**, defined in (8.13).

$$d(i,j) \quad = \quad \frac{b+c}{a+b+c+d} \tag{8.13}$$

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*). The agreement of two 1's (a positive match) is then considered more significant than that of two zeros (a negative match). Such a "binary" variable can actually be considered as a "unary variable". The similarity based on such variables is called **noninvariant similarity**. For noninvariant similarities, the most well-known coefficient is the **Jaccard coefficient**, defined in (8.14), where the number of negative matches, d, is considered unimportant and thus is ignored in the computation.

$$d(i,j) \quad = \quad \frac{b+c}{a+b+c} \tag{8.14}$$

When both symmetric and asymmetric binary variables occur in the same data set, the mixed variables approach described in Section 8.2.5 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.

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 (8.14), the distance between each pair of the three patients, Jack, Mary and Jim, should be,

$$d(jack, mary) \quad = \quad \frac{b+c}{a+b+c} = \frac{0+1}{2+0+1} = 0.33 \tag{8.15}$$

$$d(jack, jim) \quad = \quad \frac{b+c}{a+b+c} = \frac{1+1}{1+1+1} = 0.67 \tag{8.16}$$

$$d(jim, mary) \quad = \quad \frac{b+c}{a+b+c} = \frac{1+2}{1+1+2} = 0.75 \tag{8.17}$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease. Of the three patients, Jack and Mary are the most likely to have a similar disease. □

8.2.4 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.

The dissimilarity between two objects i and j can be computed using the **simple matching** approach as in (8.18):

$$d(i,j) \quad = \quad \frac{p-m}{p}, \tag{8.18}$$

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 a large number of 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 8.2.3.

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 $1, .., M_f$.

The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is one of 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

$$z_{if} \quad = \quad \frac{r_{if}-1}{M_f-1}. \tag{8.19}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 8.2.2 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

$$Ae^{Bt} \quad or \quad Ae^{-Bt}, \tag{8.20}$$

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

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 8.2.2. 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.

8.2.5 Variables of mixed types

Sections 8.2.2 to 8.2.4 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. We need a method to 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 agreeable results. However, in real applications, it is unlikely that a separate cluster analysis per variable type will generate agreeable results.

A more preferable approach is to process all variable types together, performing a single cluster analysis. One such technique, proposed by (Ducker et al. 1965) and extended by (Kaufman and Rousseeuw 1990), combines the different variables into a single dissimilarity matrix and brings all 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

$$d(i,j) \quad = \quad \frac{\Sigma_{f=1}^{p}, \delta_{ij}^{(f)} d_{ij}^{(f)}}{\Sigma_{f=1}^{p} \delta_{ij}^{(f)}} \tag{8.21}$$

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 is relaxed in some fuzzy partitioning techniques which will be briefly discussed later in this chapter.

   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 means 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 of the hierarchical method is both the key to its success because it leads to smaller computation cost without worrying about a combinatorial number of different choices, as well as the key to its main problem because it cannot correct erroneous decisions.

   It can be advantageous to combine iterative relocation and hierarchical agglomeration by first using 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. Clustering methods have been developed based on the notion of *density*. The general idea is to continue growing the given cluster so 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**.

   A grid-based method quantizes the object space into a finite number of cells which form a grid structure. It then performs all of the clustering operations 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**.

   A model-based method hypothesizes a model for each of the clusters, and finds the best fit of the data to that 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.

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.

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.

8.4.1 Classical partitioning methods: k-means and k-medoids

The most well-known and commonly used partitioning methods are **k-means** proposed by (MacQueen 1967), and **k-medoids** proposed by (Kaufman and Rousseeuw 1987), 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 the cluster, which can be viewed as the cluster's *"center of gravity"*.

The algorithm proceeds as follows: First, it randomly selects k of the objects which 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

$$E \quad = \quad \Sigma_{i=1}^{k} \Sigma_{x \in C_i} |x - m_i|^2, \tag{8.22}$$

where x is the point in space representing the given object, and m_i is the mean of cluster C_i (both x 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

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: The k-means algorithm is implemented as follows.

```
arbitrarily choose $k$ objects as the initial cluster centers;
repeat
    (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;
    update the cluster means, i.e., calculate the mean value of the objects for each cluster;
until no change;
```

□

Figure 8.1: k-means algorithm.

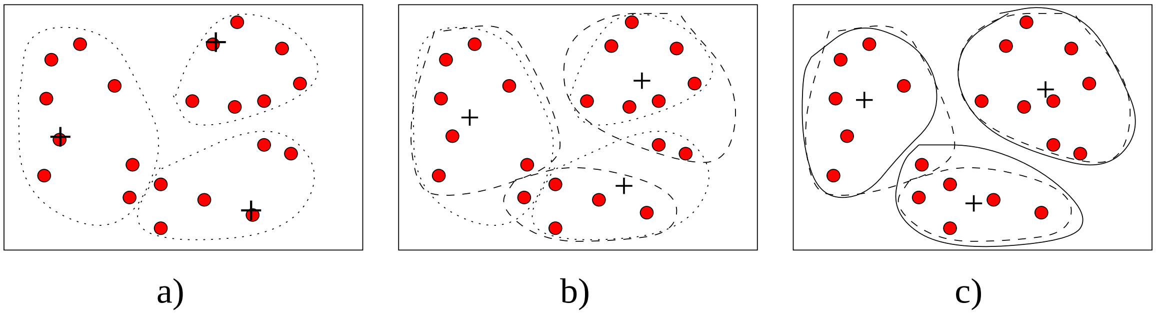

Figure 8.2: Clustering of a set of points based on the k-means method

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

The k-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users to specify k, the number of clusters, in advance can be seen as a disadvantage. The k-means method is not suitable for discovering clusters with 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 there are a set of objects located in a rectangle as shown in Figure 8.2. 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 3 objects (marked by "+") as the initial three cluster centers. Then each object is distributed to the chosen cluster domains based on which cluster center is the nearest. Such a distribution forms a silhouette encircled by dotted curve, as shown in Figure 8.2 a).

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 chosen cluster domains based on which cluster center is the nearest. Such a re-distribution forms a new silhouette encircled by dashed curve, as shown in Figure 8.2 b).

This process repeats, which leads to Figure 8.2 c). Eventually, no re-distribution of the objects in any cluster happens and the process terminates. The final clusters are the result of the clustering process. □

There are quite a few variants of the k-means method which differ in the selection of initial k means, the calculation of dissimilarity, and the strategies to calculate 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.

Algorithm 8.4.2 The k-medoids algorithm for partitioning based on the 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: The k-medoids algorithm is implemented as follows.

```
arbitrarily choose $k$ objects as the initial medoids;
repeat
    assign each object to the cluster corresponding to the nearest medoid;
    calculate the objective function, which is the sum of dissimilarities
        of all the objects to their nearest medoid;
    swap the medoid $x$ by an object $y$ if such a swap reduces the objective function;
until no change;
```

□

Figure 8.3: k-medoids algorithm.

Another variant to k-means is **k-modes** method by (Huang 1998) 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, which is called the **k-prototypes** method.

Another variant to k-means is called EM (Expectation Maximization) algorithm by (Lauritzen 1995), which extends the k-means paradigm in a different way: Instead of assigning each point to a dedicated cluster, it assigns each point to a cluster according to some weight representing the probability of membership. In another word, there are no strict boundaries between clusters. Therefore, new means are then computed based on weighted measures.

A recent effort on scaling k-means algorithm, proposed by (Bradley, Fayyad, and Reina 1998), is based on an idea of identifying regions of the data that are compressible, regions that must be maintained in main memory, and the regions that are discardable. A point is *discardable* if its membership in a cluster is ascertained, and such a point can be discarded and only the summarized clustering feature of such discarded points should be retained; a point is *compressible* if it is not discardable but belongs to a tight subcluster, and such a subcluster is summarized using its summarized clustering feature; and if a point is neither discardable nor compressible, and such a point should be *retained in main memory*. The iterative clustering algorithm will include only the summarized clustering features of the compressible points and the points which must retain in main memory in the iterative cluster analysis, and thus turns a secondary-memory based alogrithm into a main memory-based algorithm to achieve the scalability.

Representative point-based technique: The k-medoids method

The k-means algorithm is sensitive to outliers since an object with some extremely large value may substantially distort the distribution of data. Instead of taking the mean value of the objects in a cluster as a reference point, one may take a representative object in a cluster, called a *medoid*, which is the most centrally located point 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 with its corresponding reference point, which forms the basis of the k-medoids method.

PAM (partition around medoids) is a k-medoids type clustering algorithm. It finds k clusters in n objects by first finding a representative object (*medoid*) for each cluster. The initial set of medoids could be arbitrarily selected. It then iteratively replaces one of the medoids by one of the non-medoids as long as the total distance of the resulting clustering is improved.

The detailed algorithm of PAM is presented in Figure 8.3. The algorithm attempts to determine k partitions for n objects. After initial selection of k *medoids*, the algorithm repeatedly tries to make a better choice of medoids by analyzing all the possible pairs of objects such that one object is a medoid and the other is not. The measure of clustering quality is calculated for each such combination. The best choice of points in one iteration is chosen as the

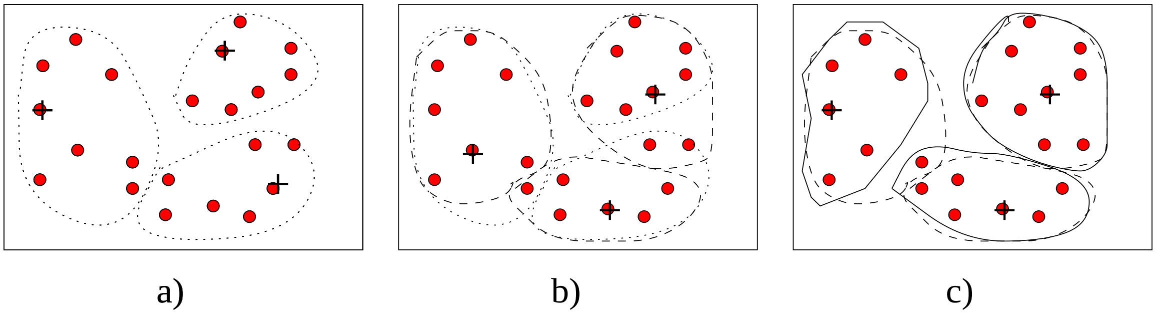

Figure 8.4: Clustering of a set of points based on the k-medoids method

medoids for the next iteration. The cost of a single iteration is $O(k(n-k)^2)$. For large values of n and k, the cost of such computation could be high.

Example 8.3 Suppose there are a set of object located in a rectangle as shown in Figure 8.4. Let $k = 3$, that is, the user would like to cluster the objects into three clusters.

According to Algorithm 8.4.2, we arbitrarily choose 3 objects (marked by "+") as the initial three cluster centers. Then each object is distributed to the chosen cluster domains based on which cluster center is the nearest. Such a distribution forms a silhouette encircled by dotted curve, as shown in Figure 8.2 a).

This kind of grouping will update the cluster centers. That is, the medoid of each cluster is recalculated based on the objects in the cluster. Regarding to these new centers, objects are re-distributed to the chosen cluster domains based on which cluster center is the nearest. Such a re-distribution forms a new silhouette encircled by dashed curve, as shown in Figure 8.4 b).

This process repeats, which leads to Figure 8.4 c). Eventually, no re-distribution of the objects in any cluster happens and the process terminates. The final clusters are the result of the clustering process. □

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 mean. However, its processing is more costly than the k-means method. Moreover, it also needs user to specify k, the number of clusters.

8.4.2 Partitioning methods in large databases: from k-medoids to CLARANS

A typical k-medoids partition algorithm like PAM works effectively for small data sets, but it does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called CLARA (clustering large applications) was developed by (Kaufman and Rousseeuw 1990).

The idea of CLARA is as follows: Instead of taking the whole set of data into consideration, only a small portion of the real data is chosen as a representative of the data, and *medoids* are chosen from this sample using PAM. If the sample is selected in a fairly random manner, it correctly represents the whole data set, and the representative objects (medoids) chosen will therefore be similar to those chosen from the whole data set. CLARA draws multiple samples of the data set, applies PAM on each sample, and gives the 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 points.

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 exactly the 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.

To improve the quality and scalability of CLARA, another clustering algorithm called CLARANS (Clustering Large Applications based upon RANdomized Search) was proposed by (Ng and Han 1994). It is also a k-medoids type algorithm and 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 every 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 parameter. If a better neighbor is found, 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. The computational complexity of every iteration in CLARANS is basically linearly proportional to the number of objects. It should be mentioned that CLARANS can be used to find the most natural number of clusters using *silhouette coefficient* which is a property of an object that specifies how much the object truly belongs to the cluster. CLARANS also enables the detection of outliers, i.e., the points that do not belong to any cluster. The performance of the CLARANS algorithm can been further improved by exploring spatial data structures, such as R*-trees, and some focusing techniques, as reported by (Ester, Kriegel and Xu 1995).

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 method suffers from its inability to perform adjustment once a merge or split decision is done. Recent studies have been emphasizing 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**: It starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters until all the objects are in a single cluster or until it satisfies certain termination condition. Most hierarchical clustering methods belong to this category. They differ only in their definition of between-cluster similarity.

   For example, a method called AGNES (Agglomerative Nesting) is introduced in (Kaufman and Rousseeuw 1990) and is also available in S-Plus. The method uses the *single-link* method, where each cluster is represented by all the data points 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. AGNES merges the nodes (i.e., individual objects or clusters) that have the least dissimilarity and goes on in a nondescending fashion.

2. **Divisive hierarchical clustering**: It does the reverse by starting with all objects in one cluster, subdividing it into smaller and smaller pieces until each object forms a cluster on its own or until it satisfies certain termination condition, such as a desired number of clusters is obtained or the distance between two closest clusters is above a certain threshold distance. Divisive methods are not generally available and rarely have been applied due to the difficulty of making a right decision of splitting at a high level. A divisive hierarchical clustering method, called DIANA (Divisia Analysis), is introduced in (Kaufman and Rousseeuw 1990), and is also available in S-Plus.

Merging of clusters is often based on the distance between clusters. The 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 points in C_i, and $|p - p'|$ is the distance between two points p and p'.

$$\begin{aligned}
d_{min}(C_i, C_j) &= min_{p \in Ci, p' \in C_j} |p - p'| \\
d_{mean}(C_i, C_j) &= |m_i - m_j| \\
d_{avg}(C_i, C_j) &= 1/(n_i n_j) \Sigma_{p \in C_i} \Sigma_{p' \in C_j} |p - p'| \\
d_{max}(C_i, C_j) &= max_{p \in Ci, p' \in C_j} |p - p'|
\end{aligned}$$

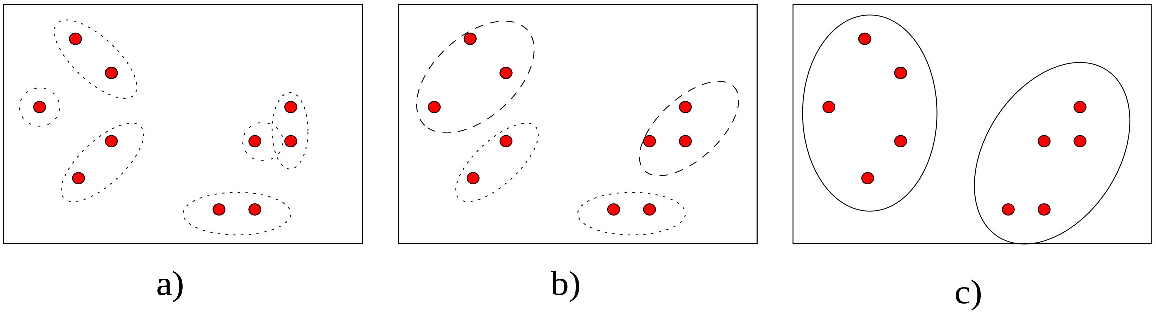

Figure 8.5: Clustering of a set of points based on the "Agglomerative Nesting" method

Example 8.4 Suppose there are a set of objects located in a rectangle as shown in Figure 8.5. An agglomerative hierarchical clustering method, called AGNES, works as follows. Initially, every object is placed into a cluster of its own. Then the clusters are merged step-by-step according to some principle such as merging the clusters with the minimum Euclidean distance between the closest objects in the cluster. Figure 8.5 *a*) shows that the closest (i.e., with minimum Euclidean distance) single object clusters are first merged into two object clusters. This cluster merging process repeats, and the closest clusters are further merged, as shown in Figure 8.5 *b*) and *c*). Eventually, all the objects are merged into one big cluster.

A divisive hierarchical clustering method, called DIANA (Divisia Analysis), works in the reverse order. That is, initially, all the objects are placed in one cluster. Then the cluster is split according to some principle, such as splitting the clusters according to the maximum Euclidean distance between the closest neighboring objects in the cluster. Figure 8.5 *c*) can be viewed as the result of the first split. This cluster splitting process repeats, and each cluster is further split according to the same criteria. Figure 8.5 *b*) and *a*) can be viewed as snapshots of such splitting. Eventually, every cluster will contain only a single object.

In either agglomerative or divisive hierarchical clustering, one can specify the desired number of clusters as a termination condition so that the hierarchical clustering process will terminate when the process reaches the desired number of clusters. □

The hierarchical clustering method, though simple, often encounters difficulties at making critical decisions for selection of correct 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 work on the newly generated clusters. That is, it will never undo what was done previously nor perform object swapping between clusters. Thus merge or split decisions, if not wise enough 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 to improve the clustering quality of hierarchical method 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, first partitions objects hierarchically using tree structures and then applies other clustering algorithms to form refined clusters. The second, called CURE, represents each cluster by a certain fixed number of representative points 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

An interesting integrated hierarchical clustering method, called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) was developed by (Zhang, Ramakrishnan and Livny 1996). It introduces two concepts, **clustering feature** and **CF tree** (**Clustering Feature tree**), and uses *CF tree* as a summarize cluster representation to achieve good speed and clustering scalability in large databases. It is also good for incremental and dynamical clustering of incoming data points.

A **clustering feature CF** is a triplet summarizing information about subclusters of points. Given N d-

dimensional points $\{X_i\}$ in a subcluster, CF is defined as

$$CF = (N, \vec{LS}, SS) \tag{8.23}$$

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

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

A **CF tree** is a height-balanced tree which stores the clustering features. It has two parameters: branching factor B and threshold T. The branching factor specifies the maximum number of children. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes. By changing the threshold value one can change the size of the tree. The non-leaf nodes store sums of their children's CFs, and thus, summarize the information about their children.

The BIRCH algorithm has the following two phases:

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

For Phase 1, the *CF tree* is built dynamically as data points are inserted. Thus, the method is an incremental one. A point 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 point, the information about it is passed towards the root of the tree. One can change the size of the *CF tree* by changing 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 value of threshold is 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 reading all the 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 are also introduced to deal with outliers and improve the quality of CF trees by additional scans of the data.

After 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 limited amount of main memory, an important consideration is to minimize the time required for I/O. It applies a multi-phase 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. So the computation complexity of the algorithm is $O(N)$, where N is the number of objects to be clustered.

Experiments have shown the linear scalability of the algorithm with respect to the number of points and good quality of clustering of the data. However, since each node in CF-tree can only hold a limited number of entries due to its size, a CF-tree node does not always correspond to 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 favors clusters with spherical shape and similar sizes, or are fragile in the presence of outliers. An interesting method, called CURE (Clustering Using REpresentatives), by (Guha, Rastogi and Shim 1998), integrates hierarchical and partitioning algorithms and overcomes the problem of favoring clusters with spherical shape and similar sizes.

CURE employs a novel hierarchical clustering algorithm that adopts a middle ground between the centroid-based and the all-point extremes. Instead of using a single centroid to represent a cluster, a fixed number of representative points are chosen to represent a cluster. These representative points are generated by first selecting well-scattered

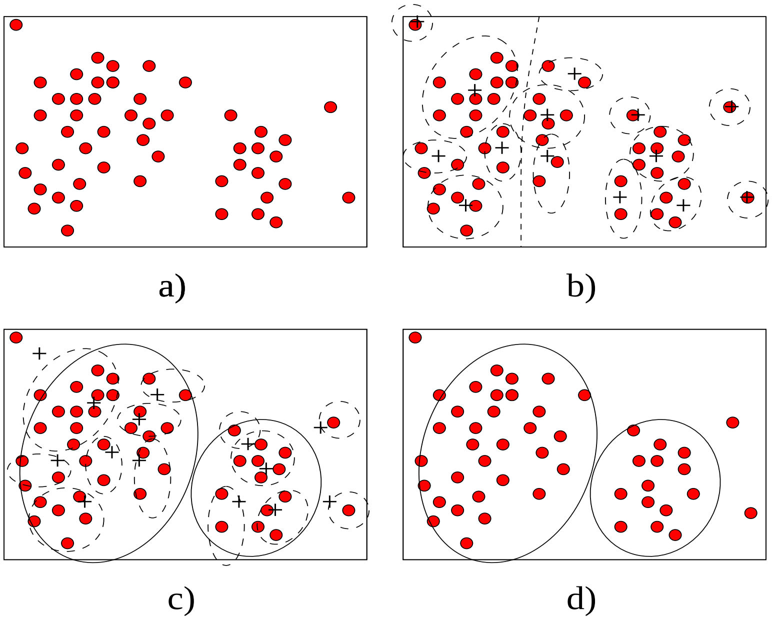

Figure 8.6: Clustering of a set of points by CURE

points from the cluster and then shrinking them toward the center of the cluster by a specified fraction (shrinking factor). The clusters with the closest pair of representative points are the clusters that are merged at each step of the algorithm.

Having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes and avoid single-link effect. The shrinking 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.

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, (2) partition sample s to p partitions, each of size s/p, (3) partially cluster partitions into s/pq clusters, for some $q > 1$, (4) eliminate outliers by random sampling: if a cluster grows too slow, eliminate it; (5) cluster partial clusters, a process of shrinking multiple representative points towards the gravity center by a fraction of α, specified by a user, where the multiple representatives capture the shape of the cluster; and (6) mark data with the corresponding cluster labels.

Here we present an example to show how CURE works.

Example 8.5 Suppose there are a set of objects located in a rectangle region. Let $p = 2$, that is, the user would like to cluster the objects into two clusters.

First, 50 objects are sampled as shown in Figure 8.6 *a*). Then these objects are partitioned initially into two clusters, each containing 25 points. We partially cluster the partitions into 10 clusters based on minimal mean distance. Each cluster representative is marked by a small cross, as shown in Figure 8.6 *b*). These representatives are shifted towards the center of gravity by a fraction α, as shown in Figure 8.6 *c*). They capture the shape of the cluster and form two clusters. Thus, the objects are partitioned into two clusters, with the outliers removed, as shown in Figure 8.6 *d*). □

CURE produces high quality clusters in the existence of outliers, complex shapes of clusters with different size. It scales well for large databases without sacrificing clustering quality. CURE needs a few user-specified parameters, such as the size of the random sample, the number of desired clusters, and the shrinking fraction α. One question is the sensitivity of these parameters to the results of clustering. A sensitivity analysis has been provided on the effect of changing parameters. It shows 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.

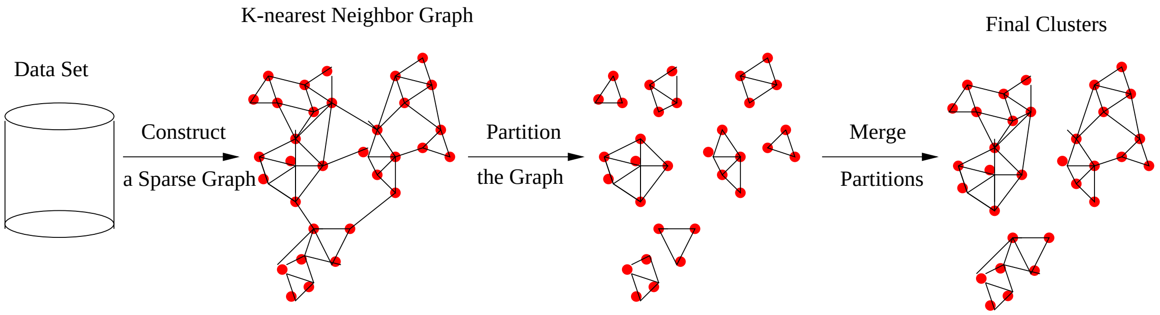

Figure 8.7: CHAMELEON: Hierarchical clustering based on k-nearest neighbors and dynamic modeling

Another agglomerative hierarchical clustering algorithm developed by (Guha, Rastogi and Shim 1999), call ROCK, 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 are defined by the number of cross links between the two clusters, and $link(p_i, p_j)$ are the number of common neighbors between two points p_i and p_j.

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

8.5.4 CHAMELEON: A hierarchical clustering algorithm using dynamic modeling

Another interesting clustering algorithm, called CHAMELEON, which explores dynamic modeling in hierarchical clustering, has been developed by Karypis, Han and Kumar (1999). 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 it 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 the information about the aggregate inter-connectivity of objects in two clusters; whereas ROCK, the group averaging method, and related schemes ignore the information about the closeness of two clusters while emphasizing on their inter-connectivity.

CHAMELEON first uses a graph partitioning algorithm to cluster the data items into a large number of relatively small subclusters. Then it uses an agglomerative hierarchical clustering algorithm to find the genuine clusters by repeatedly combining together 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.

As shown in Figure 8.7, 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 a data point is determined by the density of the region in which this object resides. In a dense region, the neighborhood is defined narrowly and in a sparse region, the neighborhood is defined more widely. Comparing with the model defined by density-based method like DBSCAN, which will be introduced in Section 8.6 and which uses a global neighborhood density, G_k captures more natural 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 weight 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,

$$RI(C_i, C_j) \quad = \quad \frac{|EC_{\{C_i,C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)} \tag{8.24}$$

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 between a pair of clusters C_i and C_j, $RC(C_i, C_j)$, is defined as 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. That is,

$$RC(C_i, C_j) \quad = \quad \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}} \tag{8.25}$$

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 arbitrary shaped clusters in high quality than DBSCAN and CURE. However, the processing cost for high dimensional data may take $O(n^2)$ time for n objects in the worst case.

8.6 Density-based clustering methods

To discover clusters with arbitrary shape, density-based clustering methods have been developed, which either connects regions with sufficiently high density into clusters or clusters objects based on density function distribution.

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, developed by Ester, Kriegel, Sander and Xu (1996). The algorithm grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in spatial database with noise. A cluster is defined as a maximal set of density-connected points.

The basic idea of density-based clustering is as follows: For each object of a cluster, the neighborhood of a given radius (ϵ) (called *ϵ-neighborhood*) has to contain at least a minimum number of objects (*MinPts*).

An object, within a given radius (ϵ) containing no less than a minimum number of neighborhood objects ($MinPts$), is called a **core object** (with respect to a radius ϵ and a minimum number of points $MinPts$).

An object p is **directly density-reachable** from object q with respect to a radius ϵ and a minimum number of points $MinPts$ in a set of objects D if p is within the ϵ-neighborhood of q which contains at least a minimum number of points, $MinPts$.

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 for $1 \leq i \leq n$, $p_i \in D$ and p_{i+1} is directly density-reachable from p_i with respect to ϵ and $MinPts$.

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 $MinPts$.

Example 8.6 In Figure 8.8, based on a given ϵ represented by the radius of the circles, and let $MinPts = 3$, M is directly density-reachable from P, and Q is (indirectly) density-reachable from P. However, P is not density-reachable from Q. Similarly, R and S are density-reachable from O; and O, R and S are all density-connected. ■

Notice that 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.

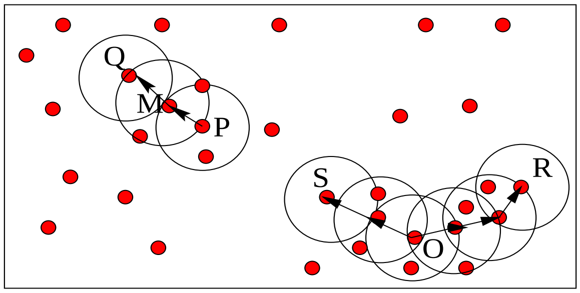

Figure 8.8: Density reachability and density connectivity in density-based clustering

A **density-based cluster** is a set of density-connected objects which is maximal with respect to density-reachability, and every object not contained in any cluster is *noise*.

Based on the notion of density-reachability, a density-based clustering algorithm, DBSCAN, is developed for clustering data in a database. It 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. It 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.

8.6.2 OPTICS: Ordering Points To Identify the Clustering Structure

Although the density-based clustering algorithm, DBSCAN, can discover cluster objects with the selection of some input parameters, such as ϵ and $MinPts$, it still puts the responsibility to users to select good parameter values in order to find correct clusters. Actually, this is the problem associated with many other clustering algorithms. Such parameter settings are usually pretty hard 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 partitions of the data sets. Moreover, high-dimensional real data sets often have very skewed distribution, and there does not even exist a global parameter setting for which the result of a clustering algorithm may describe the intrinsic clustering structure accurately.

To overcome this difficulty, a cluster ordering method, called OPTICS (Ordering Points To Identify the Clustering Structure) has been developed by (Ankerst, Breunig, Kriegel and Sander 1999). It computes an augmented clustering-ordering for automatic and interactive cluster analysis. This cluster ordering contains information which is equivalent to density-based clustering corresponding to a broad range of parameter settings.

By examining density-based clustering algorithm, 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 with respects to a lower density. Therefore, in order to produce density-based clusters with a set of distance parameters, one need to select the objects for processing in a specific order so that the object which is density-reachable with respect to the lowest ϵ value is finished first.

Based on this idea, two value need to be stored for each object: *core-distance* and *reachability-distance*.

The **core-distance** of an object p is the smallest distance ϵ' between p and an object in its ϵ-neighborhood such that p would be a core object with respect to ϵ' if this neighbor is contained in the ϵ-neighborhood of p. Otherwise, the core-distance is UNDEFINED.

The **reachability-distance** of an object p with respect to another object o is the smallest distance such that p is directly density-reachable from o if o is a core object. If o is not a core object, even at the generating distance ϵ, the reachability-distance of an object p with respect to o is UNDEFINED.

The OPTICS algorithm creates an ordering of a database, additionally storing the core-distance and a suitable reachability-distance for each object. Such information is sufficient for extraction of all density-based clusterings with respect to any distance ϵ' smaller than the generating distance ϵ from this order.

The cluster-ordering of a data set can be presented and understood graphically. For example, Figure 8.9 is the reachability-plot for a simple 2-dimensional data set, which presents a general overview about how the data is structured and clustered. Methods are also developed for viewing clustering structures for high dimensional data.

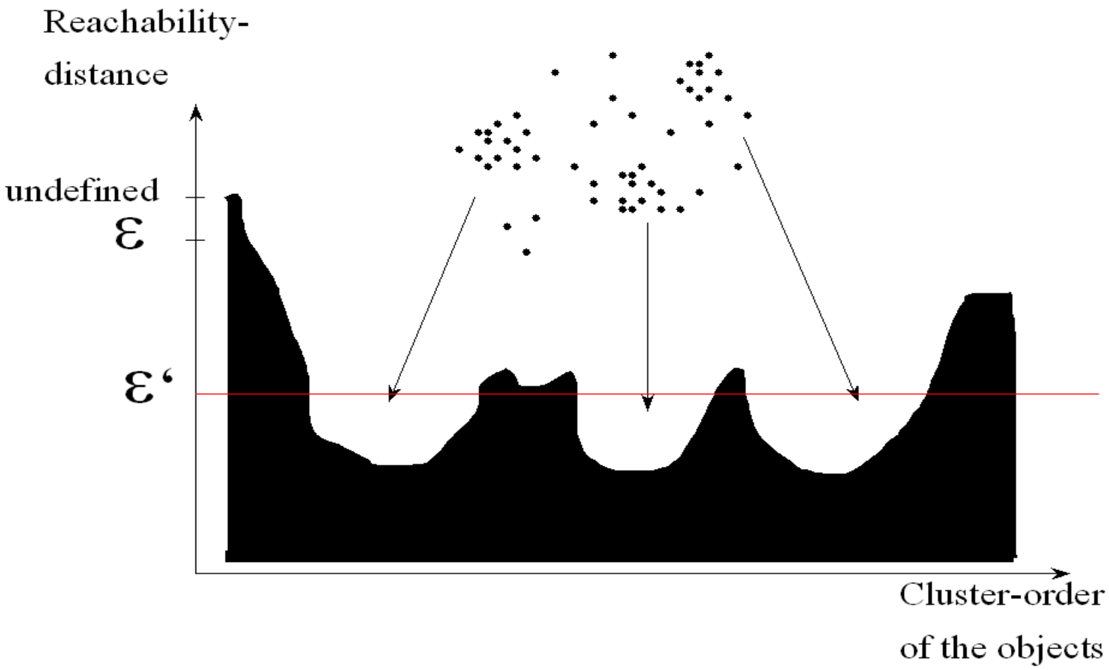

Figure 8.9: Cluster ordering in OPTICS

Because of the structural equivalence of the OPTICS algorithm to DBSCAN, the OPTICS algorithm has the same run-time complexity as that of DBSCAN. Spatial indexing structures can be used to further enhance its performance.

8.6.3 DENCLUE: Clustering based on density distribution functions

DENCLUE (an abbreviation for DENsity-based CLUstEring), a clustering method based on a set of density distribution functions, has been developed by (Hinneburg and Keim 1998).

The method is based on the following ideas: (1) the influence of each data point can be modeled formally using a mathematical function, called *influence function*, and the influence function can be seen as a 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 influence functions 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.

The **influence function** of a data point $y \in F^d$, where F^d is a d-dimensional feature space, is a basic function $f_B^y : F^d \rightarrow R_0^+$, which is defined in terms of a basic influence function f_B,

$$f_B^y = f_B(x, y). \tag{8.26}$$

In principle, the influence function can be an arbitrary function, but it should be reflexive and symmetric. It can be a *Euclidean distance function*, a *square wave influence function*,

$$f_{Square}(x, y) = \begin{cases} 0 & if\ d(x, y) > \sigma \\ 1 & otherwise \end{cases} \tag{8.27}$$

or a *Gaussian influence function*,

$$f_{Gauss}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}} \tag{8.28}$$

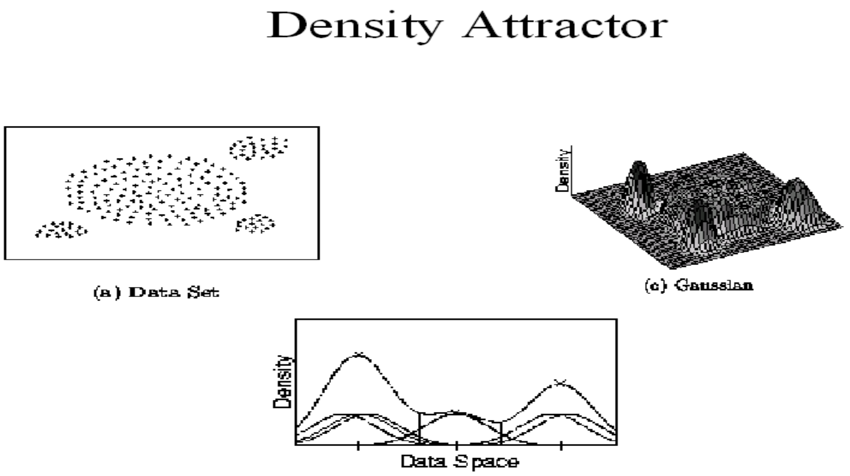

Figure 8.10: Density function and density-attractor

The **density function** is defined as the sum of influence functions of all data points. Given N data objects described by a set of feature vectors $D = \{x_1, \ldots, x_N\} \subset F^d$, the density function is defined as,

$$f_B^D = \Sigma_{i=1}^N f_B^{x_i}(x). \tag{8.29}$$

For example, the density function which results from the Gaussian influence function (8.28) is,

$$f_{Gaussian}^D(x) = \Sigma_{i=1}^N e^{-\frac{d(x,y)^2}{2\sigma^2}} \tag{8.30}$$

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.

Based on these notions, both *center-defined cluster* and *arbitrary-shape cluster* can be defined formally. A *center-defined cluster* is a subset C being density-extracted, with its density function no less than a threshold ξ; otherwise (i.e., if density function is less than ξ), it is an outlier. An *arbitrary-shape cluster* is a set of C's, each being density-extracted, with the density function no less than a threshold ξ, and there exists a path P from each region to another and the density function for each point along the path is no less than ξ.

DENCLUE has the following major advantages in comparison with many clustering algorithms: (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 but only keeps information about grid cells that do actually contain data points and manages these cells in a tree-based access structure, and thus it is significantly faster than some influential algorithms, such as it is faster than DBSCAN by a factor of up to 45. However, the method needs careful selection of the parameters, density parameter σ and noise threshold ξ, and the selection of such parameters may significantly influence the quality of clustering results.

8.7 Grid-based clustering methods

A grid-based approach uses multi-resolution grid data structure. It first quantizes the space into a finite number of cells which form a grid structure and then performs all the operations in such a grid structure. The main advantage of the approach is its fast processing time which is typically independent of the number of data objects but dependent only on the number of cells in each dimension in the quantized space.

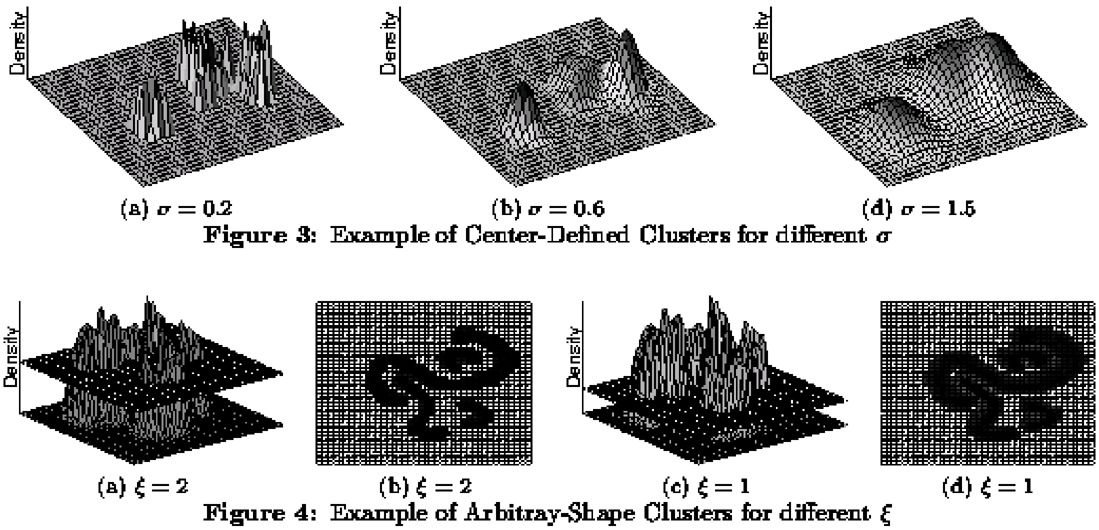

Figure 3: Example of Center-Defined Clusters for different σ

Figure 4: Example of Arbitray-Shape Clusters for different ξ

Figure 8.11: Examples of center-defined clusters and arbitrary-shaped clusters

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 multi-resolution approach developed by (Wang, Yang and Muntz 1997). In this approach, the spatial area 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. Moreover, important pieces of statistical information, such as mean, max, min, count, standard deviation, etc. associated with the attribute values in each grid cell are precomputed and stored before a query is submitted to a system.

Figure 8.12 shows a hierarchical structure for STING clustering.

The set of statistics-based parameters includes the following: the attribute-independent parameter, n (*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 is loaded into the database, the set of parameters, n, m, s, *min*, *max* of the bottom level cells are calculated directly from data. The value of *distribution* could be either assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as χ^2-test. Parameters of higher level cells can be easily computed from the parameters of the lower level cells. The type of distribution of a higher level cell can be computed based on the majority distribution types of its corresponding lower level cells plus a threshold filtering process. If the distributions of the lower level cell disagree with each other and fails the threshold test, the distribution type of the high level cell is set to "*none*".

The statistical information collected will be very useful at answering queries. A top-down, statistics information grid-based query answering method can be outlined as follows. First, one can determine a layer to start with, which usually contains a small number of cells. For each cell in the current layer, we calculate the confidence interval (or estimated range) of probability that this cell is relevant to the query. The irrelevant cells will be removed from further consideration, and the deeper layer processing will examine the relevant cells only. This process repeats until it reaches the bottom layer. At this time, if the query specification is met, return the regions of relevant cells that meet the requirement of the query; otherwise, retrieve the data that fall into the relevant cells, do further processing; and return the results that meet the requirement of the query.

The approach offers several advantages over some other clustering methods: (1) the grid-based computation is

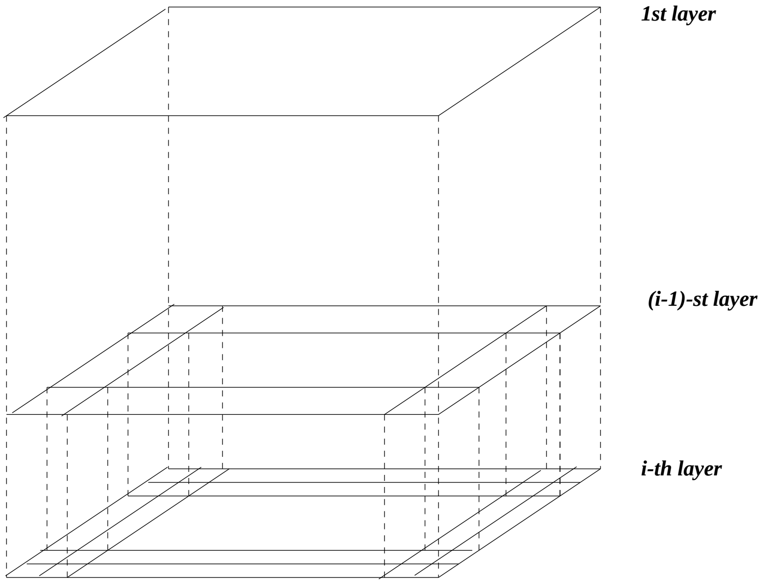

Figure 8.12: A hierarchical structure for STING clustering

query-independent since the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of query; (2) the grid structure facilitates parallel processing and incremental updating; and (3) a major advantage of the method is the efficiency of the method: 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, the total number of objects.

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

8.7.2 WaveCluster: Clustering using wavelet transformation

WaveCluster, developed by (Sheikholeslami, Chatterjee and Zhang 1998), is a multi-resolution clustering approach which first summarize the data by imposing a multidimensional grid structure on the data space, and then transforms the original feature space by wavelet transform and find dense regions in the transsformed space.

In this approach, each grid cell summarizes the ifnromation of a group of points which map into the cell, and this summary information typically fits into main memory for multi-resolution wavelet transform and the subsequent cluster analysis. In the grid structure, the numerical attributes of a spatial object can be represented by a *feature vector* where each element of the vector corresponds to one numerical attribute, or *feature*. For an object with n numerical attributes, the feature vector will be one point in the n-dimensional feature space.

Wavelet transform is a signal processing technique that decomposes a signal into different frequency subbands. The wavelet model also works on n-dimensional signals by applying one-dimensional transform n times.

In wavelet transform, spatial data are converted into the frequency domain. Convolution with an appropriate kernel function results in a transformed space where the natural clusters in the data become more distinguishable. Clusters can then be identified by finding the dense regions in the transformed domain.

Wavelet transform provides the following interesting features. First, it provides unsupervised clustering. The hat-shape filters emphasize regions where the points cluster, but simultaneously tend to suppress weaker information

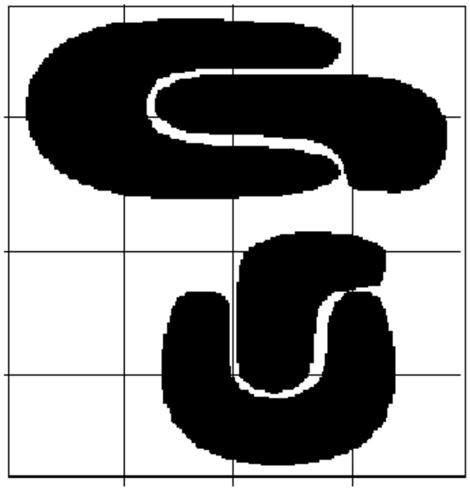

Figure 8.13: A sample of 2-dimensional feature space. Figure is from Sheikholeslami, Chatterjee and Zhang (1998).

in their boundary. Thus, dense regions in the original feature space act as attractors for nearby points and as inhibitors for the points that are not close enough. This means the clusters in the data automatically stand out and clear the regions around them. Second, the low-pass filters used in the wavelet transform will automatically remove outliers. Moreover, the multi-resolution property of wavelet transform can help detecting the clusters at different levels of accuracy. Finally, it is very fast to apply wavelet transform and such a process can also be performed in parallel.

The wavelet-based clustering algorithm can be outlined as follows:

Algorithm 8.7.1 The wavelet-based clustering algorithm for multi-resolution clustering by wavelet transform.

Input: The feature vectors of multidimensional data objects.

Output: Clustered objects.

Method: The wavelet-based clustering algorithm is implemented as follows.

```
(1)   Quantize feature space and then assign objects to the units;
(2)   Apply wavelet transform on the feature space;
(3)   Find the connected components (clusters) in the subbands of
      transformed feature space, at different levels;
(4)   Assign labels to the units;
(5)   Make the look up tables and map the objects to the clusters.  ■
```

The computational complexity of the algorithm is $O(N)$, where N is the number of objects in the database.

For example, Fig. 8.13 shows a sample of 2-dimensional feature space, where each point in the image represents the feature values of one object in the spatial data sets. Fig. 8.14 shows the result of wavelet transforms at different scales from fine (scale 1) to coarse (scale 3). At each level, sub-band LL (average) is shown at the upper-left quadrant, sub-band LH (horizontal edges) is shown at the upper-right quadrant, sub-band HL (vertical edges) is shown at the lower-left quadrant, and sub-band HH (corners) is shown at the lower-right quadrant.

WaveCluster is a grid-based and density-based algorithm. WaveCluster conforms with all the requirements of good clustering algorithms: It handles large datasets efficiently, discovers clusters with arbitrary shape, successfully handles outliers, and is insensitive to the order of input. The study also compares WaveCluster with BIRCH, CLARANS and DBSCAN, and shows that WaveCluster outperforms these methods in both efficiency and clustering quality.

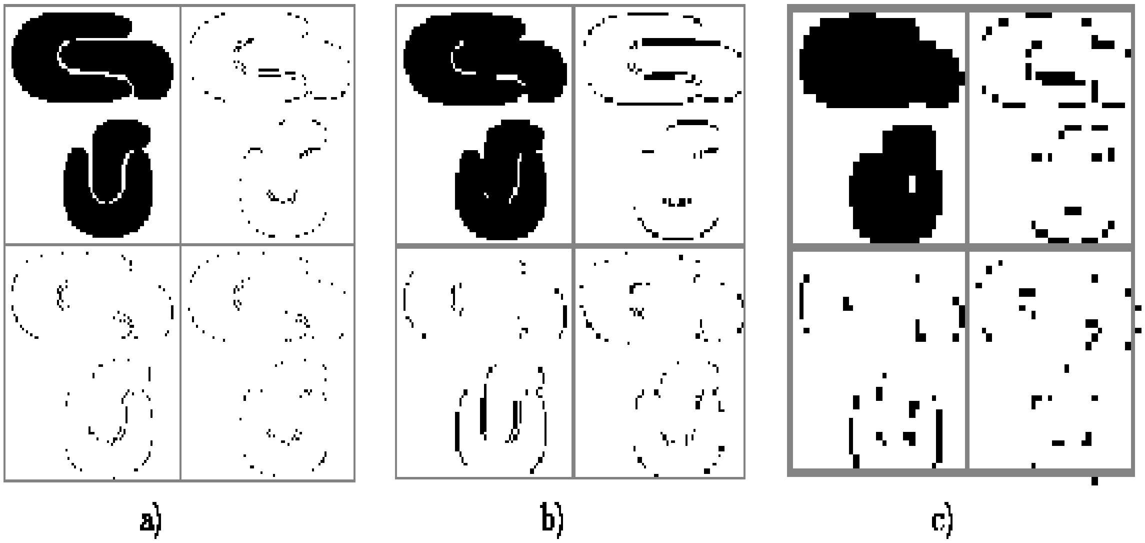

Figure 8.14: Multi-resolution of the feature space in Figure 8.13 at a) scale 1; b) scale 2; c) scale 3. Figure is from Sheikholeslami, Chatterjee and Zhang (1998).

8.7.3 CLIQUE: Clustering high-dimensional space

Another clustering algorithm, CLIQUE, developed by (Agrawal et al. 1998), is another integrated, density-based and grid-based clustering method. It is good for clustering high dimensional data in large databases.

Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. Data clustering identifies the sparse and the crowded places, and hence discovers the overall distribution patterns of the data set.

A unit is **dense** if the fraction of total data points contained in the unit exceeds an input model parameter. A cluster is a maximal set of connected dense units.

CLIQUE partitions the m-dimensional data space into non-overlapping rectangular units, identifies dense units, and finds clusters in all subspaces of the original data space, using a candidate generation method similar to the Apriori algorithm for mining association rules.

CLIQUE performs multidimensional clustering in two steps:

First, CLIQUE identifies clusters by determining dense units in all the subspaces of interests and then determining connected dense units in all subspaces of interests.

An important heuristic that CLIQUE adopts is the Apriori principle in high dimensional clustering: *If a k-dimensional unit is dense, then so are its projects in $(k-1)$-dimensional space.* That is, if any of the $(k-1)$-th units is not dense, its corresponding k-th dimensional unit cannot be a candidate dense unit. Therefore, all the candidate k-dimensional dense units can be generated from $(k-1)$-dimensional dense ones.

Second, CLIQUE generates minimal description for the clusters as follows. It first determines the maximal regions that cover a cluster of connected dense units for each cluster and then determines minimal cover for each cluster.

CLIQUE automatically finds subspaces of the highest dimensionality such that high density clusters exist in those subspaces. It is insensitive to the order of records in input and does not presume some 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 simplicity of the method.

8.8 Model-based clustering methods

Model-based clustering methods make use of certain models for the clusters and attempt to optimize the fit between the data and the model. It is often based on the assumption that the data are generated by a mixture of underlying probability distributions.

The model-based clustering methods have two major approaches: *statistical approach* and *neural network approach*.

Statistical approach

Clustering in machine learning is usually referred to as unsupervising learning or concept (clustering) formation. Most of work on concept formation adopt a probability-based approach, which uses probability measurements, such as category utility used in (Fisher 1987) and (Gennari, Langley and Fisher 1989), for clustering and represents concepts or clusters with probability descriptions.

For example, COBWEB (Fisher 1987) performs a top-down, unsupervised, incremental classification of concepts on categorical data. It uses one measure, called *category utility*, to score each newly inserted node and determine where to put in the hierarchy. The method has several limitations. First, it is based on the assumption that probability distributions on separate attributes are statistically independent of each 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 probability-based tree (such as (Fisher 1987)) that is built to identify clusters is not height-balanced for skewed input data, which may cause the time and space complexity to degrade dramatically.

Another system called CLASSIT (Gennari, Langley and Fisher 1989) 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 sum over discrete attributes as in COBWEB. However, it suffers the similar problem as COBWEB and thus it is not suitable for clustering large database data.

AutoClass (Cheeseman and Stutz 1996) implements a Bayesian clustering method based on mixture models. It uses the Bayesian statistical analysis to estimate the number of clusters.

Neural network approach

The best known neural network approach in clustering is SOM (self-organizing feature map) proposed by Kohonen in 1981. It can be viewed as a nonlinear projection from an m-dimensional input space onto a lower-order (typically 2-dimensional) regular lattice of cells. Such a mapping is used to identify clusters of elements that are similar (in an Euclidean sense) in the original space.

8.9 Outlier analysis

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

Outliers could be caused by measurement or execution error or by inherent data variability. For example, the display of a person's age as -999 could be caused by a program default setting of an unrecorded age. However, the salary of the chief executive offier of a company could naturally stand out as an outlier among the salaries of the employees in a company.

Many data mining algorithms are trying to eliminate outliers or minimizing the influence of outliers. However, "one person's noise could be another person's signal". In many cases, the outliers themselves might be of particular interest to a user. Thus, outlier detection and analysis is an interesting data mining task.

Outlier mining has wide applications. It can be used in fraud detection for finding unusual usage of credit cards or tele-communication services, in customized marketing for finding spending behavior of extremely rich or poor people, or in medical analysis for finding unusual response to certain medicine or treatment, etc.

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

The problem of defining outlier is nontrivial. If a regression model is used for data modeling, analysis of residuals can give good estimation for data "extremeness". The task become tricky when finding outlier in time series, as they might be hidden in trend, seasonal, or other cyclic changes. When multi-dimensional data is analyzed, not any particular one but a combination of dimension values might be extreme. Also, defining outliers in categorical data requires separate consideration.

We will examine in detail the issues for defining and mining outliers.

The most obvious and often quite effective ways for outlier detection are *data visualization methods*. 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. It will also encounter difficulties to detect outliers with many categorical attributes or data of high dimensionality since human eyes are only good at visualizing numerical data of two to three dimensions.

The computer-based outlier detection methods can be categorized into three approaches: *statistical approach*, *distance-based approach*, and *deviation analysis approach*. Notice also that many *clustering algorithms* discard outliers as noise, however, they can be modified to have outlier detection as a byproduct of their execution.

8.9.1 Statistical approach for outlier detection

The statistical approach assumes the model underlying distribution that generates data set (e.g., normal distribution) and then identifies outliers using a test called *discordancy test*. The test construction requires knowledge about parameters of the data set, such as data distribution and knowledge of distribution parameters, such as mean, variance, and the number of expected outliers.

A statistical discordancy test examines two hypotheses: a *working hypothesis* and an *alternative hypothesis*.

The working hypothesis is retained if there is no statistically significant evidence supports its rejection. The working hypothesis H is a statement that the whole data set comes from an initial probability model F, i.e.,

$$H : o_j \in F, where\ j = 1, 2, \ldots, n.$$

Statistical test is performed to see whether an object o_i is significantly large (or small) in relation to the distribution complying with the model F. If o_i is shown to be discordant at the level of test, then it is not reasonable to believe that o_i comes from F and an alternative model that o_i comes from another model G is adopted. The result is very much dependent on which F model is chosen because o_i might be an outlier under one model and a perfectly valid value under the other.

The form of alternative distribution is also very important in determining the power of the test, i.e. the probability that working hypothesis is rejected when o_i is really an outlier. There are several forms of alternative hypotheses.

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

  $$\overline{H} : o_j \in G, where\ j = 1, \ldots, n.$$

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

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

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

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

Different test statistics have been proposed which allow to accept or reject a working hypothesis. Choice is to be made in regard to which test to prefer in a particular situation. Assuming that some statistic T has been chosen for discordancy testing and value of the statistic for element o_i is v, distribution of T is built to find whether value v is discordant. Significant probability $SP(v) = Prob(T > v)$ is evaluated. If $SP(v)$ is sufficiently small, then o_i is discordant and the working hypothesis is rejected.

In case of multiple outliers there are two basic types of procedures for detecting outliers:

- *Block procedures.* In this case either all suspect elements treated as outliers or all accepted as consistent.
- *Consecutive (or sequential) procedures.* A procedure, called *inside-out*, is more accepted. Its main idea is that the least outlier is tested first. If it is found to be outlier, then all more extreme values are also outliers; otherwise, the next more extreme element is tested, and so on.

A major drawback of the statistical approach is that most tests are for single attributes, but many data mining problems require to find outliers at multidimensional space. Moreover, the statistical approach requires the knowledge about parameters of the data set, such as the data distribution. However, in many cases, 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 observed distribution cannot be adequately modeled with any standard distribution.

8.9.2 Distance-based outlier detection

To counter main limitations imposed by statistical methods, the notion of distance-based (DB) outlier is introduced in (Knorr and Ng 1997).

A distance-based (DB) outlier is defined as follows.

> An object o in a dataset T is a distance-based outlier with parameters p and d, i.e., $DB(p, d)$, if at least a fraction p of the objects in T lie at a distance greater than d from o.

Distance-based outlier generalizes other notions provided by numerous discordancy tests and replaces many of these tests with a single algorithm detecting only outliers of the proposed types. It avoids a lot of computation associated with fitting the observed distribution into some standard distribution and choosing discordancy tests. Therefore, a distance-based outlier is also called a *unified outlier*, or *UO-outlier*.

Based on this notion of outlier, one can show that for many discordancy tests if an object o is an outlier according to a specific discordancy test, o is also a $DB(p, d)$ outlier for some suitably defined p and d. For example, if for a normal distribution observations that lie 3 or more standard deviations from the mean considered to be outliers, 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. They are outlined as follows.

- *index-based algorithm.* The index-based algorithm uses multidimensional indexing structures, such as R-trees or k-d trees, to search for neighbors of each object o in a dataset within radius d around that object. As soon as $m + 1$ neighbors are found, where m is the maximum number of objects within the d-neighborhood of an outlier, it is clear that object o is not an outlier. This algorithm has the worst case complexity of $O(k * N^2)$, where k is the dimensionality and N is the number of items in the dataset, and it scales well as k grows. But this complexity evaluation takes into account search time only, while building an index itself can make this approach uncompetitive.
- *nested-loop algorithm.* The nested-loop algorithm has the same computational complexity as the index-based algorithm but it 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 dataset into several logical blocks. By carefully choosing the order of loading the blocks into different halves, I/O efficiency can be achieved.
- *cell-based algorithm.* To avoid N^2 computational complexity, a cell-based algorithm is developed for memory-resident datasets, with complexity $O(c^k + N)$, where c is some constant depending on the number of cells, and k is dimensionality. In this method, the data space is partitioned into cells with a side length equal to $\frac{d}{2\sqrt{k}}$. Every cell has two layers surrounding it, the first is one cell thick and 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. The

algorithm counts number of items in the current cell itself, in the cell and the first layer together, and in the cell and both layers together. Some items in the current cell can be outliers only if the total number of items in the cell and the first layer is less than or equal to the maximum number M of outliers that can be in the d-neighborhood of an outlier. If this condition does not hold, then all the items in the cell can be removed from further investigation as they cannot be outliers. If the total number of the items in the cell and both layers is less than or equal to M, then all items in a cell are outliers; and if it is more than M, then only some of the items in the cell are outliers. To detect these outliers object-by-object processing is used again and for every object in the cell objects from the second layer are checked for being d-neighbors. Only those objects in the cell having less than M neighbors in both 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 3 passes over the dataset are required. It works for large, disk-resident datasets. The cell-based algorithm does not scale well for high dimensions.

Distance-based outlier requires user to set both p and d parameters for testing and checking that every outlier discovered by the algorithm is a "real" outlier. Looking for the suitable parameters can involve many iterations.

8.9.3 Deviation-based outlier detection

(Arning, Agrawal and Raghavan 1996) introduced a notion of outliers that is different from both statistical and distance-based methods as it does not require defining distance measure between data elements. It introduces a linear algorithm for deviation (outlier) detection using implicit redundancy of the data. It defines sequential exception as an element observed in a series of similar data and disturbing this series.

The process of deviation detection described simulates the way people can distinguish unusual object after seeing a series of similar objects and learning their main characteristics. The authors introduce such terms as exception set, dissimilarity function, cardinality function, smoothing factor. An exception set is defined as the subset of items whose removal results in the greatest reduction of the dissimilarity of the residual set, which is equivalent to the greatest reduction in Kolmogorov complexity for the amount of data discarded. Dissimilarity function can be any function that returns a low value if the elements of the set are similar to each other and a higher value if the elements are dissimilar and does not require a metrical distance between the items. Cardinality function can be any function whose value is greater for a subset of the given set with a higher cardinality. The smoothing factor is defined for any subset of the given set and indicates how much the dissimilarity can be reduced by removing the subset from the set. Finding an exception set can be NP-hard and this motivated the definition of the sequential exception problem. Instead of finding dissimilarity of the current subset and the complementary set the algorithm selects a sequence of subsets from the set and for every subset it finds the dissimilarity difference with the preceding subset. Several iterations with random order of the subsets in a sequence are performed to alleviate possible impact of the input order on the results. For every iteration the element with the maximal value of dissimilarity function is selected. The maximum dissimilarity among all iterations scaled by the cardinality function gives exception (outlying) set. The paper gives an example showing that the effectiveness of the algorithm depends on the dissimilarity function used, and the task of the function definition is complicated by the fact that the nature of exception is not known in advance. The option of looking for a universal dissimilarity function was rejected by the authors after working with some real life databases. The algorithm has linear complexity $O(N)$, where N is the number of items in the input, provided that the number of iterations is not very big and dissimilarity function is such that for the next subset in the sequence it can be computed incrementally from the function used for the previous subset.

(Sarawagi, Agrawal and Megiddo 1998) uses OLAP data cubes to identify regions of anomalies in large multidimensional data. To make the process more efficient outlier detection is overlapped with cube computation, the algorithm replaces hypothesis-driven exploration, when an analyst is simply looking at the data at different levels of aggregation to discover an anomaly, with discovery-driven exploration, where indicators of exceptions are precomputed comparing real data in a cube cell with the anticipated value. The computation of the anticipated value takes into account the cell's position in the data cube and trends along different dimensions the cell belongs to. The method allows to find anomalies at different levels of aggregation what is very difficult to do manually when search space is very large because of the number of dimension and several level deep hierarchies in every dimension. The algorithm considers a value in a cell of a data cube to be an exception if it is significantly different from the value anticipated based on a statistical model.

Anticipated value in a cell is considered to be a function of all aggregates computed using the cell value. If some dimension has hierarchy defined on it, the cell value also depends on its parents along the hierarchies. A cell value is considered to be an exception, if its standardized residual, calculated as absolute difference between actual and anticipated values of the cell normalized by the normal deviation, is more than some threshold.

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. Typical computer-based outlier analysis methods include *statistical approach, distance-based approach*, and *deviation analysis approach.*

Exercises

1. What is clustering? How is it different from classification?
2. 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.

3. 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 $q = 3$.

4. 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 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 compatable pair, and why?

5. Clustering has been popularly recognized as an important data mining task with broad applications. Give one application example 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.

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

7. 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?

8. 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.

9. Data cubes and multi-dimensional 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.

10. 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 have been introduced in several textbooks, such as Jain and Dubes (1988) [JD88], Kaufman and Rousseeuw (1990) [KR90]. The methods for combining different variables into a single dissimilarity matrix was first proposed by Ducker et al. (1965?) [?] and later extended by Kaufman and Rousseeuw (1990) [KR90].

For partitioning methods, the k-means algorithm was first introduced by MacQueen (1967) [Mac67], the k-medoids algorithms such as PAM and CLARA were proposed by Kaufman and Rousseeuw (1990) [KR90]. The EM (Expectation Maximization) algorithm was introduced by Lauritzen (1995), and the k-mode algorithm was proposed by Huang (1998) [Hua98]. Ng and Han (1994) [NH94] proposed an algorithm called CLARANS which improves the quality and efficiency of the sampling-based k-medoid clustering method based upon randomized search. Ester *et al.* (1995) [EKX95] propose techniques for further improvement of the performance of the CLARANS algorithm using efficient spatial access methods, such as R*-tree and focusing techniques.

For hierarchical methods, agglomerative hierarchical clustering, represented by AGNES, and divisive hierarchical clustering, represented by DIANA, were introduced by Kaufman and Rousseeuw (1990) [KR90]. 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 BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), proposed by Zhang, Ramakrishnan, and Livny (1996) [ZRL96], CURE (Clustering Using REpresentatives), proposed by Guha, Rastogi, and Shim (1998) [GRS98], ROCK for clustering categorical attributes, proposed by Guha, Rastogi, and Shim (1999) and CHAMELEON by Karypis, Han, and Kumar (1999) [KHK99].

For density-based clustering methods, Ester et al. (1996) [EKSX96] proposed a density-based DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, and Ankerst et al. (1999) [ABKS99] developed a cluster ordering method, called Optics (Ordering Points To Identify the Clustering Structure). Hinneburg and Keim (1998) [HK98] proposed DENCLUE, a DENsity-based CLUstEring method based on a set of density distribution functions.

Grid-based clustering methods have been studied recently, with a few interesting algorithms proposed. STING (STatistical INformation Grid) is a grid-based multi-resolution approach proposed by Wang, Yang, and Muntz (1997) [WYM97]. WaveCluster is a multi-resolution clustering approach which transforms the original feature space by wavelet transform, developed by Sheikholeslami, Chatterjee and Zhang (1998) [SCZ98]. CLIQUE, developed by (Agrawal et al. 1998), is an integrated, density-based and grid-based clustering method for clustering high dimensional data.

Typical model-based methods include the statistical clustering approach and the neural network approach. Works of the statistical clustering approach include AutoClass by Cheeseman and Stutz (1996) [CS96], COBWEB by Fisher (1987) , CLASSIT by Gennari, Langley and Fisher (1989) [GLF89]. Studies of the neural network approach include SOM (self-organizing feature map) by Kohonen (1982) [Koh82].

Outlier detection and analysis can be categorized into three approaches: ***statistical approach***, ***distance-based approach***, and ***deviation analysis approach***. The statistical approach has been studied in Distance-based (DB) outlier, as introduced by Knorr and Ng (1997) [KN97, KN98]. For the deviation analysis approach, Arning, Agrawal and Raghavan (1996) introduced a linear algorithm for outlier detection using implicit redundancy of the data. Sarawagi, Agrawal, and Megiddo (1998) introduced a discovery-driven method to identify regions of anomalies in large multi-dimensional data in OLAP data cubes.

Bibliography

[AAR96] 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.

[ABKS99] 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*, Philadelphia, PA, June 1999.

[AGGR98] 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.

[BFR98] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 9–15, New York, NY, August 1998.

[BKSS90] 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.

[CS96] 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.

[EKSX96] 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.

[EKX95] 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.

[Fis87] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[Fis95] 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.

[GLF89] J. Gennari, P. Langley, and D. Fisher. Models of incremenral concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[GRS98] 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.

[GRS99] 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.

[HK98] 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.

[Hua98] 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.

[JD88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Printice Hall, 1988.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *COMPUTER*, 32:68–75, 1999.

[KN97] 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.

[KN98] 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.

[Koh82] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[Lau95] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, 1:281–297, 1967.

[NH94] 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.

[SAM98] 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.

[SCZ98] 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.

[WYM97] 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.

[ZRL96] 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.