3.2.2 GENERATING ASSOCIATION RULES FROM FREQUENT ITEMSETS

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where strong association rules satisfy both minimum support and minimum confidence).

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}$$

Association rules based on conditional probability can be generated as follows:

- ✧ For each frequent itemset *l*, generate all nonempty subsets of *l*.
- ✧ For every nonempty subset s of *l*, output the rule "$s \Rightarrow (l - s)$" if $\frac{support\_count(l)}{support\_count(s)} \geq$ min $\_conf$, where min_conf is the minimum confidence threshold.

Example: Generating association rules. Suppose the data contain the frequent itemset *l* = {I1, 12,15}. What are the association rules that can be generated from *l* ? The nonempty subsets of *l* are {I1, 12}, {I1, 15}, {12, 15}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:

I1 ⋀ I2 ⇒ I5, confidence = 2 / 4 = 50%

I1 ⋀ I5 ⇒ I2, confidence = 2 / 2 = 100%

I2 ⋀ I5 ⇒ I1, confidence = 2 / 2 = 100%

I1⇒ I2 ⋀ I5, confidence = 2 / 6 = 33%

I2⇒ I1 ⋀ I5, confidence = 2 / 7 = 29%

I5⇒ I1 ⋀ I2, confidence = 2 / 2 = 100%

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because these are the only ones generated that are strong. association rules can contain more than one conjunct in the right-hand side of the rule.

3.2.3 METHODS TO IMPROVE APRIORI'S EFFICIENCY

i. **A hash-based technique:** Hashing itemset counts.

A hash-based technique can be used to reduce the size of the candidate k-itemsets, Ck, for k> 1. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1, from the candidate 1-itemsets in C1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts (Figure 3.2). A 2-itemset whose corresponding bucket count in the hash table is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique

may substantially reduce the number of the candidate k-itemsets examined (especially when k = 2).

Create hash table, H_2
using hash function
$h(x, y) = ((order\ of\ x) * 10 + (order\ of\ y))\ mod\ 7$
$\longrightarrow$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	1	1	2	2	1	2	4
bucket contents	{I1,I4}	{I1,I5}	{I2,I3} {I2,I3}	{I2,I4} {I2,I4}	{I2,I5}	{I1,I2} {I1,I2}	{I3,I4} {I3,I4} {I1,I3} {I3,I4}

Figure 3.2 : Hash table, *H*2, for candidate 2-itemsets: This hash table was generated by scanning the transactions of Table 5.1 while determining *L*1 from*C*1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in *C*2.

Transaction reduction (reducing the number of transactions scanned in future iterations):

A transaction that does not contain any frequent *k*-itemsets cannot contain any frequent (*k*+1) -itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for *j*- itemsets, where *j* > *k*, will not require it.

Partitioning (partitioning the data to find candidate itemsets):

A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure 3.3). It consists of two phases. In Phase I, the algorithm subdivides the transactions of *D* into *n* nonoverlapping partitions. If the minimum support threshold for transactions in *D* is *min sup*, then the minimum support count for a partition is *min sup_the number of transactions in that partition*. For each partition, all frequent itemsets within the partition are found. These are referred to as local frequent itemsets. The procedure employs a special data structure that, for each itemset, records the TIDs of the transactions containing the items in the itemset. This allows it to find all of the local frequent *k*-itemsets, for *k* = 1, 2, : : : , in just one scan of the database.

A local frequent itemset may or may not be frequent with respect to the entire database, *D*. *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions*. Therefore, all local frequent itemsets are candidate itemsets with respect to *D*. The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to *D*. In Phase II, a second scan of *D* is conducted in which the actual support of each candidate is assessed in order to determine the global frequent

itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

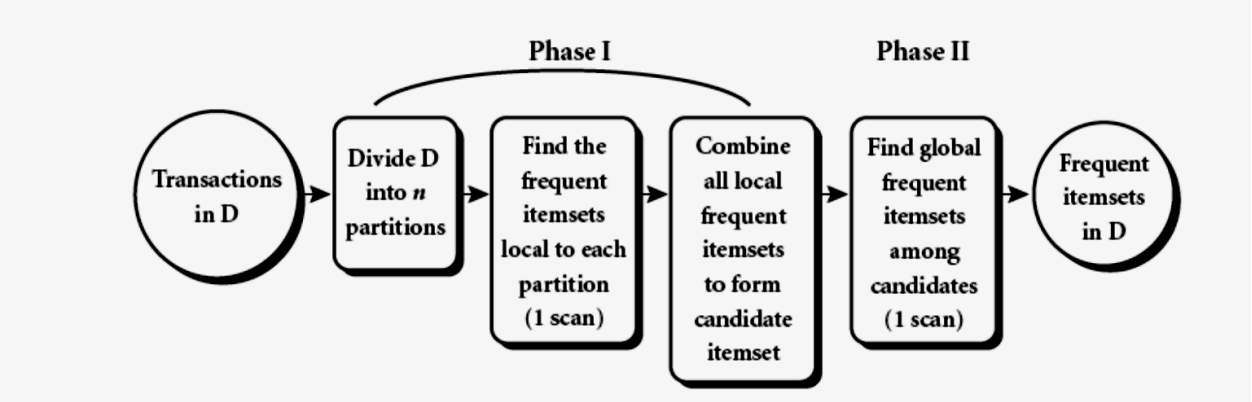

Figure 3.3 Mining by partitioning the data.

Sampling (mining on a subset of the given data):

The basic idea of the sampling approach is to pick a random sample *S* of the given data *D*, and then search for frequent itemsets in *S* instead of *D*. In this way, we trade off some degree of accuracy against efficiency. The sample size of *S* is such that the search for frequent itemsets in *S* can be done in main memory, and so only one scan of the transactions in *S* is required overall. Because we are searching for frequent itemsets in *S* rather than in *D*, it is possible that we will miss some of the global frequent itemsets. To lessen this possibility, we use a lower support threshold than minimum support to find the frequent itemsets local to *S* (denoted L^S). The rest of the database is then used to compute the actual frequencies of each itemset in L^S. A mechanism is used to determine whether all of the global frequent itemsets are included in L^S. If L^S actually contains all of the frequent itemsets in *D*, then only one scan of *D* is required. Otherwise, a second pass can be done in order to find the frequent itemsets that were missed in the first pass. The sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive applications that must be run frequently.

Dynamic itemset counting (adding candidate itemsets at different points during a scan):

A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires fewer database scans than Apriori.

3.3 MINING DIFFERENT KINDS OF ASSOCIATION RULES

3.3.1 MINING MULTILEVEL ASSOCIATION RULES FROM TRANSACTION DATABASES

Multilevel association rule involve concepts at different levels of abstraction. We can mine multilevel association rules efficiently using concept hierarchies, which defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Data can be generalized by replacing low level concepts within the data by their higher level concepts or ancestors from a concept hierarchy. In a concept hierarchy, which is represented as a tree with the root as D i.e., Task relevant data.

TID	Items Purchased
T100	IBM-ThinkPad -T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003

Table 3.1 Task relevant data D

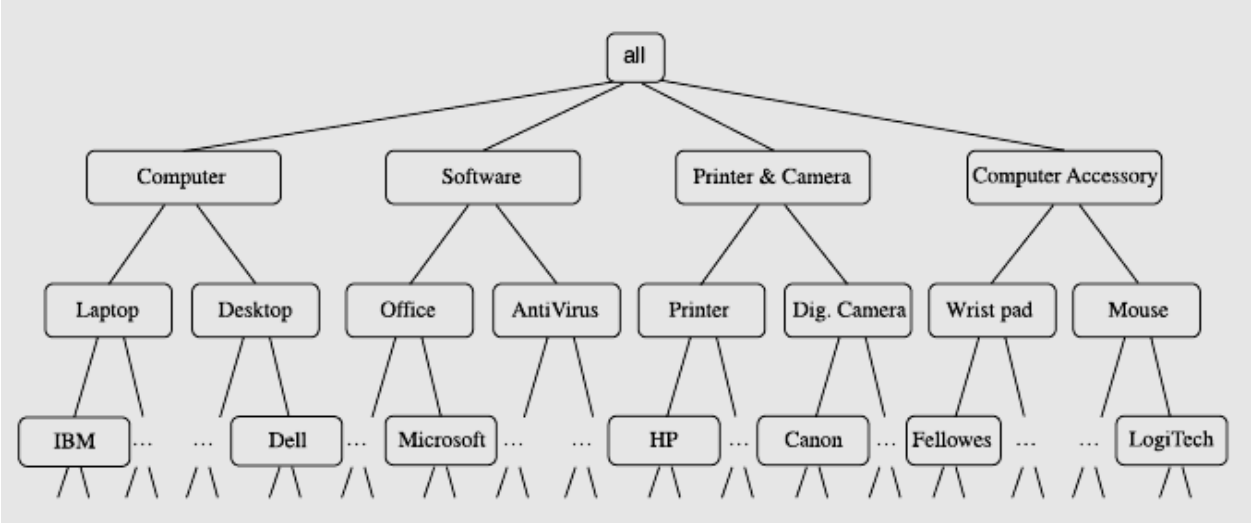

Figure 3.4 A concept hierarchy for *AllElectronics* computer items

The popular area of application for multilevel association is market basket analysis, which studies the buying habits of customers by searching for sets of items that are frequently purchased together which was presented in terms of concept hierarchy shown above. The items in Table 5.6 are at the lowest level of the concept hierarchy of Figure 3.4. It is difficult to find interesting purchase patterns at such raw or primitive-level data. For instance, if *"IBM-*

ThinkPad-R40/P4M" or *"Symantec-Norton-Antivirus-2003"* each occurs in a very small fraction of the transactions, then it can be difficult to find strong associations involving these specific items. Few people may buy these items together, making it unlikely that the itemset will satisfy minimum support. However, we would expect that it is easier to find strong associations between generalized abstractions of these items, such as between *"IBM laptop computer"* and *"antivirus software."*

Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations. Following are the various algorithm that can be used:

- **Using uniform minimum support for all levels** (referred to as **uniform support**):

  The same minimum support threshold is used when mining at each level of abstraction. For example, in Figure 3.5, a minimum support threshold of 5% is used throughout (e.g., for mining from *"computer"* down to *"laptop computer"*). Both *"computer"* and *"laptop computer"* are found to be frequent, while *"desktop computer"* is not.

  When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining itemsets containing any item whose ancestors do not have minimum support.

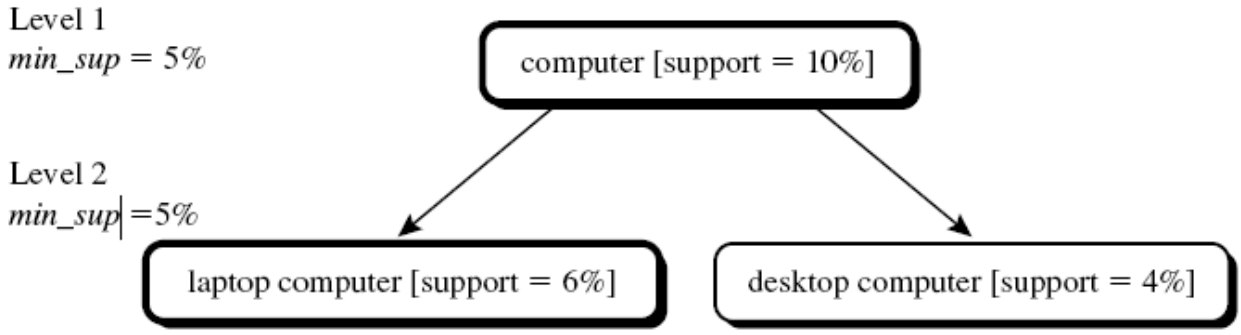

Figure 3.5 Multilevel mining with uniform support

The uniform support approach has some difficulties. It is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction. If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the following approach.

- **Using reduced minimum support at lower levels** (referred to as **reduced support**):

  Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in Figure 3.6, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, *"computer," "laptop computer,"* and *"desktop computer"* are all considered frequent.

- **Using item or group-based minimum support** (referred to as **group-based support**):

  Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price, or on items of interest, such as by setting particularly low support thresholds for *laptop computers* and *flash drives* in order to pay particular attention to the association patterns containing items in these categories.

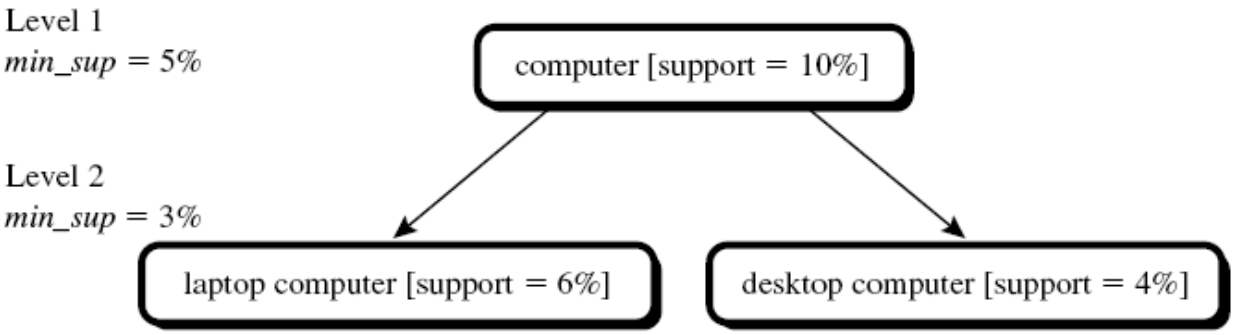

Figure 3.6 Multilevel mining with reduced support.

3.3.2 MINING MULTIDIMENSIONAL ASSOCIATION RULES FROM RELATIONAL DATABASES AND DATAWAREHOUSES

The association rule that have a single predicate is called as a single dimensional or intradimensional association rule because it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule).

Example: $buys(X, "sony\ laptop") \Rightarrow buys(X, "HP\ printer")$

But rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Additional relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates, such as:

$$age(X, "20 \ldots 29") \wedge occupation(X, "student") \Rightarrow buys(X, "laptop")$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Above rule contains three predicates (*age*, *occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has no repeated predicates. Multidimensional association rules with no repeated predicates are called inter-dimensional association rules. We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimensional association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X,20\ldots29) \wedge buys(X,\text{laptop}) \Longrightarrow buys(X,\text{HP printer})$$

Database attributes can be categorical or quantitative. Categorical attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). Categorical attributes are also called nominal attributes, because their values are "names of things." Quantitative attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*). Techniques for mining multidimensional association rules can be categorized into two basic approaches regarding the treatment of quantitative attributes.

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels, such as "0. . . 20K", "21K . . . 30K", "31K . . . 40K", and so on. Here, discretization is *static* and predetermined that is, here attributes are statically discretized by using predefined concept hierarchies.

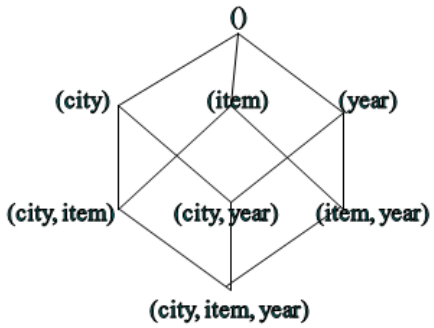

Figure 3.7: Concept Hierarchy

In the second approach, *quantitative attributes are discretized or clustered into "bins" based on the distribution of the data*. These bins may be further combined during the mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as (dynamic) quantitative association rules.

3.4 FROM ASSOCIATION MINING TO CORRELATION ANALYSIS

Most association rule mining algorithms employ a support-confidence framework. But it has one major disadvantage: If we choose low support threshold, then uninteresting pattern may also be mined.

For e.g.: Consider a transaction at *AllElectronics* with respect to the purchase of computer games and videos. Let *game* refer to the transactions containing computer games, and *video* refer to those containing videos. *Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, while 7500 included videos, and 4000 included both computer games and videos and* suppose *minimum support is 30% and minimum confidence is 60%.*

The following association rule is discovered:

***buys(X, "computer games")*⟹buys(X, "videos") [support=40%, confidence=66%]**

is a strong association rule ; since its support value=$\frac{4000}{10000}$ = 40% and Confidence value = $\frac{4000}{6000}$ = 66% satisfy minimum support and minimum confidence thresholds.

But this rule misleading because probability of purchasing videos is 75%, which is even larger than 66%.

3.4.1 FROM ASSOCIATION ANALYSIS TO CORRELATION ANALYSIS

To tackle this weakness, a correlation measure can be used to augment the support-confidence framework for association rules.

A⟹B [support, confidence, correlation]

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets *A* and *B*. There are many different correlation measures, out of these we choose lift.

Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is independent of the occurrence of itemset B if P(A∪B) = P(A)P(B); Otherwise, itemset A and B are dependent and correlated events.

The lift between the occurrence of A and B can be measured as:

$$\textbf{Lift(A,B)} = \frac{P(A \cup B)}{P(A)P(B)}$$

If the resulting value is less than 1, then occurrence of A is negatively correlated with the occurrence of B. If greater than 1, then A and B are positively correlated. This means that the occurrence of one implies the occurrence of the other. If the resulting value is 1, then A and B are independent and there is no correlation between them.

Example: To filter out misleading association rule, we use correlation.

Let $\overline{game}$ refer to the transactions which do not contain computer games, and $\overline{video}$ refer to those that do not contain videos. The transaction can be summarized using contingency table.

	game	$\overline{game}$	Σ_{row}
video	4,000	3,500	7,500
$\overline{video}$	2,000	500	2,500
Σ_{col}	6,000	4,000	10,000

P({game})=$\frac{6000}{10000}$ = .6

P({video})=$\frac{7500}{10000}$ =.75

P(game, video)=$\frac{4000}{10000}$ = .4

Lift(game, video)= .89 which is less than 1. so negative correlation between two itemsets video and game.

3.5 CLUSTERING

Clustering is the process of grouping the data into classes or *clusters*, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters. Dissimilarities are assessed based on the attribute values describing the objects. The process of grouping a set of physical or abstract objects into classes of *similar* objects is called clustering.

The goal of clustering is to discover both the dense and the sparse regions in a data set. Data clustering has been studied in the statistics, machine learning, and the database communities with the diverse emphases.

Cluster Analysis

- Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters.
- The process of grouping a set of physical or abstract objects into classes of similar objects is called **clustering.**
- Clustering is an example of **unsupervised learning**.
- Clustering is a form of **learning by observation**, rather than **learning by examples**.

By automated clustering, we can identify dense and sparse regions in object space and, therefore, discover overall distribution patterns and interesting correlations among data attributes. Cluster analysis has been widely used in numerous applications, including market

research, pattern recognition, data analysis, and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value, and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used to help classify documents on the Web for information discovery.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their *similarity*. Clustering can also be used for outlier detection, where outliers (values that are "far away" from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

The following are typical *requirements* of clustering in data mining:

- **Scalability:**
  Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a *sample* of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

- **Ability to deal with different types of attributes:**

  Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

- **Discovery of clusters with arbitrary shape**:

  Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.

- **Minimal requirements for domain knowledge to determine input parameters:**

  Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.

- **Ability to deal with noisy data:**

  Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

- **Incremental clustering and insensitivity to the order of input records:**

  Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clustering depending on the order of presentation of the input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.

- **High dimensionality:**

  A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

- **Constraint-based clustering:**

  Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

- **Interpretability and usability:**

  Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

3.5.1 TYPES OF DATA IN CLUSTER ANALYSIS

Main memory-based clustering algorithms typically operate on either of the following two data structures.

- **Data matrix (or object by variable structure):**

  Objects are usually represented as points in a multi-dimensional space, where each dimension represents a distinct attribute describing object. Thus, a set of objects is represented as an m by n matrix, where there are m rows, one for each object and n columns, one for each attribute.

$$\begin{matrix} x_{11} \dots & x_{1f} \dots & x_{1n} \\ \dots & \dots & \dots \\ x_{i1} \dots & x_{if} \dots & x_{in} \\ \dots & \dots & \dots \\ x_{m1} \dots & x_{mf} \dots & x_{mn} \end{matrix}$$

✧ **Dissimilarity matrix (or object-by-object structure):**

This stores a collection of proximities that are available for all pairs of n objects. A dissimilarity matrix, P, is an n by n matrix containing all the pairwise dissimilarities between the objects being considered. Here d(i, j) is the dissimilarity between objects i and j. If d(i, j) < 0 and if it is close to 0, then we say that object i and j are highly similar. Also, the dissimilarity matrix holds the following properties.

- d(i, j) = d(j, i)
- d(i, i) = 0

$$\begin{matrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{matrix}$$

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

Example: Consider a data set consisting of measurements of the variables : temperature and yield, for 8 different geographical locations. The data set used is represented by the 8 by 2 data matrix.

$$\begin{bmatrix} 24.0 & 113 \\ 25.1 & 109 \\ 24.7 & 124 \\ 25.5 & 106 \\ 31.9 & 156 \\ 30.3 & 167 \\ 29.6 & 159 \\ 30.8 & 168 \end{bmatrix}.$$

The dissimilarity matrix derived from the data matrix using Euclidean distance as:

$$\begin{bmatrix} 0 & & & & & & & \\ 4.15 & 0 & & & & & & \\ 11.02 & 15.01 & 0 & & & & & \\ 7.16 & 3.03 & 18.02 & 0 & & & & \\ 43.72 & 47.49 & 32.80 & 50.41 & 0 & & & \\ 54.37 & 58.23 & 43.36 & 61.19 & 11.12 & 0 & & \\ 46.34 & 50.20 & 35.34 & 53.16 & 3.78 & 8.03 & 0 & \\ 55.42 & 59.27 & 44.42 & 62.23 & 12.05 & 1.12 & 9.08 & 0 \end{bmatrix}.$$

The object dissimilarity can be computed for objects described by *interval-scaled* variables; by *binary* variables; by *categorical*, *ordinal*, and *ratio-scaled* variables; or combinations of these variable types.

1. **Interval-Scaled Variables:**

Interval-scaled variables are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature. The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. So in order to avoid confusion we can standardize the measurements to give all variables an equal weight.

To standardize measurements, one choice is to convert the original measurements to unit less variables. Given measurements for a variable *f*, this can be performed as follows.

i. Calculate The Mean Absolute Deviation, S_f,

$$S_f = \frac{1}{n}\left(|x_{1f} - m_f| + |x_{2f} - m_f| + \ldots + |x_{nf} - m_f|\right)$$

Where $x_{1f},\ldots,x_{nf}$ are n measurements of f, and m_f is the mean value of f. i.e.,

$$m_f = \frac{1}{n}\left(x_{1f} + x_{2f} + \ldots + x_{nf}\right)$$

ii. Calculate the standardized measurement, or z-score:

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

After standardization, the dissimilarity between the objects can be computed using either

- Euclidean distance
- Manhattan Distance
- Minkowski Distance

The dissimilarity is computed based on the distance between pair of objects. The Euclidean distance can be computed as:

$$d(i, j) = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \ldots + \left(x_{in} - x_{jn}\right)^2}$$

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

The Manhattan Distance can be computed using the formula below:

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

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

- d(i, j) >= 0 ; Distance is a non negative number.
- d(i, i) = 0; The distance of an object to itself is 0.
- d(i, j) = d(j, i) ; distance is a symmetric function
- d(i, j) <= d(i, h) + d(h, j) ; Going directly from object *i* to object *j* in space is no more than making a detour over any other object *h* ((*triangle inequality*).

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

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

2. **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. The dissimilarity between objects of type binary can be described using either symmetric or asymmetric binary variables.

If all binary variables are having the same weight, then we can use 2-by-2 contingency table of Table 3.2, where *q* is the number of variables that equal 1 for both objects *i* and *j*, *r* is the number of variables that equal 1 for object *i* but that are 0 for object *j*, *s* is the number of variables that equal 0 for object *i* but equal 1 for object *j*, and *t* is the number of variables that equal 0 for both objects *i* and *j*. The total number of variables is *p*, where *p* = *q*+*r*+*s*+*t*.

		Object *j*		
		1	0	*sum*
Object *i*	1	*a*	*b*	*a*+*b*
	0	*c*	*d*	*c*+*d*
	sum	*a*+*c*	*b*+*d*	*p*

Table 3.2 A contingency table for binary variables.

A binary variable is symmetric if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1.

One such example could be the attribute *gender* having the states *male* and *female*. Dissimilarity that is based on symmetric binary variables is called symmetric binary dissimilarity. Its dissimilarity (or distance) measure, defined in following equation used to assess the dissimilarity between objects *i* and *j*.

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

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

3. **Categorical, Ordinal, and Ratio-Scaled Variables**

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

Let the number of states of a categorical variable be *M*. The dissimilarity between two objects *i* and *j* can be computed based on the ratio of mismatches:

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

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.

4. **Ordinal Variables**

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

Ordinal variables can be treated like interval-scaled and then do the following steps on the data

- The value of *f* for the *i*th object is $x_{i\,f}$, and *f* has M_f ordered states, representing the ranking 1, : : : , M_f .Replace x_{if} by their rank
- Map the range of each variable onto [0.0, 1.0] by replacing *i*-th object in the *f*-th variable by

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

- Compute the dissimilarity using methods for interval-scaled variables

5. **Ratio-Scaled Variables**

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

$$Ae^{Bt} \text{ or } Ae^{-Bt}$$

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

There are **three methods** to handle ratio-scaled variables for computing the dissimilarity between objects.

- Treat ratio-scaled variables like interval-scaled variables. This, however, is not usually a good choice since it is likely that the scale may be distorted.
- Apply logarithmic transformation to a ratio-scaled variable *f* having value *xi f* for object *i* by using the formula $y_{i\,f} = \log(x_{i\,f})$. The $y_{i\,f}$ values can be treated as interval valued.
- Treat *xi f* as continuous ordinal data and treat their ranks as interval-valued.

6. **Variables of Mixed Types**

If the databases contain objects of *mixed* variable types: *interval-scaled*, *symmetric binary*, *asymmetric binary*, *categorical*, *ordinal*, or *ratio-scaled* then any of the following approaches will be followed.

One approach is to group each kind of variable together, performing a separate cluster analysis for each variable type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate cluster analysis per variable type will generate compatible results.

A more preferable approach is to group all the variables of different types into a single cluster using dissimilarity matrix. Now make a set of common scale of the interval [0.0,1.0].

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

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

If either X_{if} or X_{jf} is missing or if $X_{if} = X_{jf} = 0$ and the variable f is assymmetric variable then the indicator is $\delta_{ij}^{(f)} = 0$

otherwise, $\delta_{ij}^{(f)} = 1$

3.6 A CATEGORIZATION OF MAJOR CLUSTERING METHODS

The major clustering methods can be classified into the following categories:

✧ **Partitioning Methods:**

A partitioning method constructs *k* partitions of the data, where each partition represents a cluster and $k \leq n$, *where* n is the number of objects or data tuples. The partitions that formed should satisfy the following requirements:

(1) Each group must contain at least one object, and

(2) Each object must belong to exactly one group.

Typical methods: k-means, k-medoids, CLARANS

✧ **Hierarchical Methods:**

A hierarchical method creates a hierarchical decomposition of the given set of data objects i.e., it creates a tree of clusters. A hierarchical method can be classified as either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the *bottom-up* approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The *divisive approach*, also called the *top-down* approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds. Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone.

Typical methods: Diana, Agnes, BIRCH, ROCK, CAMELEON

- **Density-based Methods:**

  This density based algorithms are developed based on the notion of *density*. Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers) and discover clusters of arbitrary shape.

  Typical methods: DBSACN, OPTICS, DenClue

- **Grid-based Methods:**

  In this class of methods, the object space is divided into a grid structure. Clustering techniques are then applied using cells of the grid as the basic units instead of individual data points. This has the advantage of improving the processing time significantly.

  Typical methods: STING, WaveCluster, CLIQUE

- **Model-based Methods:**

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

  Typical methods: EM, SOM, COBWEB

- **Constraint-based Clustering:**

  Constraint-based clustering is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints. A constraint expresses a user's expectation or describes "properties" of the desired clustering results, and provides an effective means for communicating with the clustering process. Various kinds of constraints can be specified, either by a user or as per application requirements.

  Typical methods: COD (obstacles), constrained clustering

3.7 PARTITIONING METHODS

Partitioning algorithms construct partitions of a database of *n* objects into a set of *k* clusters. The construction involves determining the optimal partition with respect to an objective function. In general, partitioning algorithm is:

- Non-hierarchical, each instance is placed in exactly one of *k* non overlapping clusters
- Since only one set of clusters is output, the user normally has to input the desired number of clusters *k.*

Given *k*, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or very different. There are two main categories of partitioning algorithms:

I. K-means algorithm

II. K – medoid algorithm.

3.7.1 **The *k*-Means Method**

K – means is an iterative clustering algorithm in which it partitions all the n objects into k clusters, so that each cluster has maxim um intra cluster similarity and minimum inter-cluster similarity. Here cluster similarity is measured by the center of gravity of a cluster. The k-means algorithm works on the following steps:

- **Selects k objects randomly that represent cluster mean or center of gravity of a cluster**
- **Assign an object to the cluster for which cluster mean is most similar, based on distance between object and cluster mean**
- **Update cluster mean of each cluster, if necessary**
- **Repeat this process until the criterion function converges.**

The k-means algorithm takes n objects and the number of cluster k as its input and returns the cluster that minimizes the squared-error criterion. The squared-error criterion function is defined as follows:

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

Where E is the sum of the square error for all objects in the data set; p is the point in space representing a given object; and m_i is the mean of the cluster C_i.

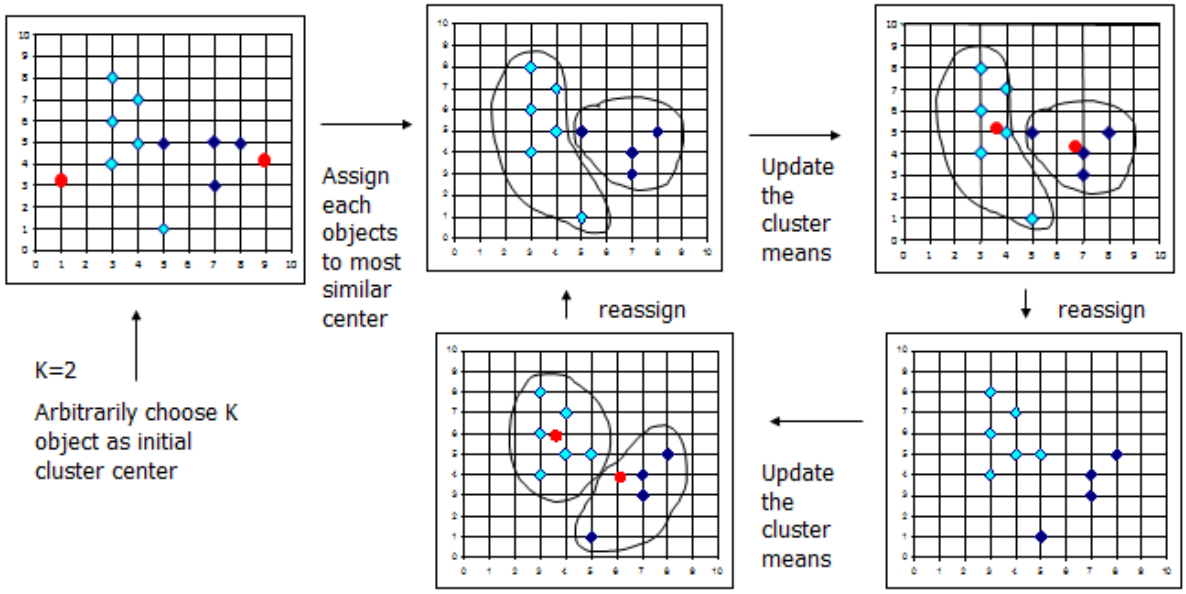

Figure 3.8 :k-means clustering

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

Input : k : no. of clusters

D : a data set containing n objects

Output: A set of k clusters

Method:

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

Scaling the k-means algorithm

For achieving scalability the following can be done:

- Identify three kinds of regions in the data: regions that are compressible, regions that must be maintained in main memory, and regions that are discardable. If an object is neither discardable nor compressible, then it should be *retained in main memory*.

- To achieve scalability, the iterative clustering algorithm only includes the clustering features of the compressible objects and the objects that must be retained in main memory, thereby turning a secondary-memory based algorithm into a main-memory-based algorithm.
- Another approach is to perform micro clustering , which first groups nearby objects into "micro clusters" and then performs *k*-means clustering on the micro clusters.

Advantages:

- With a large number of variables, k-means may be computationally faster than hierarchical clustering(if k is small)
- k-means may produce tighter clusters than hierarchical clustering, especially if the clusters are globular

Disadvantages:

- Difficult in comparing the quality of the clusters produced
- Applicable only when mean is defined
- Need to specify k, the number of clusters in advance
- Unable to handle noisy data and outliers
- Not suitable to discover clusters with non-convex shapes

Example:

Let us consider 5 points {X1,X2,X3,X4,X5} with the following co-ordinate as a 2 dimensional samples for clustering X1=(0, 2),X2 = (0, 0), X3 = (1.5, 0), X4=(5, 0), X5 = (5, 2) with the initial cluster C1={X1, X2, X4}, C2 = {X3, X5}. Apply the k-means algorithm to find the new clusters.

step1: The centroid for these clusters are

$$M_k = \left(\frac{1}{n_k}\right)\sum_{i=1}^{n_k} X_{ik}$$

C1={X1, X2, X4} n_k =3 X1= (0, 2), X2 = (0, 0), X4=(5, 0)

M_1 ={(0+0+5) / 3, (2+0+0) / 3}

M_1={1.66,0.66}

C2 = {X3, X5} n_k =3 X3 = (1.5, 0), X5 = (5, 2)

M_2 ={(1.5+5) / 2, (0+2) / 2}

M_2 = {3.25, 1}

Step2: Within cluster variation after initial random distribution of samples are:

$$e_k^2 = \sum_{i=1}^{n_k} (X_{ik} - M_k)^2$$

C1={X1, X2, X4} M_1={1.66,0.66}

$$e_1^2 = [(0 - 1.66)^2 + (2 - 0.66)^2] + [(0 - 1.66)^2 + (0 - 0.66)^2] + [(5 - 1.66)^2 + (0 - 0.66)^2]$$

=19.36

$$e_2^2 = [(1.5 - 3.25)^2 + (0 - 1)^2] + [(5 - 3.25)^2 + (2 - 1)^2]$$

= 8.12

Step3: The total square error is

$$E_k^2 = \sum_{k=1}^{k} e_k^2$$

= 19.36 + 8.12

= 27.48

Step4: When we reassign all samples, depending on a minimum distance from centroid M1 and M2 the new redistribution of samples inside clusters will be:

d (M_1, X_1) = $\sqrt{(0 - 1.66)^2 + (2 - 0.66)^2}$ = 2.14

d (M_2, X_1) = 3.40 $\Rightarrow X_1 \in C_1$

d (M_1, X_2) = 1.79, d (M_2, X_2) =3.40 $\Rightarrow X_2 \in C_1$

d (M_1, X_3) = 0.83, d (M_2, X_3) =2.01 $\Rightarrow X_3 \in C_1$

d (M_1, X_4) = 3.41, d (M_2, X_4) =2.01 $\Rightarrow X_4 \in C_2$

d (M_1, X_5) = 3.60, d (M_2, X_5) =2.01 $\Rightarrow X_5 \in C_2$

Therefore new clusters C_1 = { X_1, X_2, X_3 } and C_2 = { X_4, X_5}

The new centroid are

M_1 ={(0 + 0 + 1.5)/3, (2 + 0 + 0/3)}

= {0.5, 0.67} M_2= {(5+ 5)/2, (0 + 2)/2}

= {5,1}

The corresponding within cluster variations are

e_1^2 = [(0 - 0.5)^2 + (2 - 0.66)^2] + [(0 - 0.5)^2 + (0 - 0.66)^2] + [(1.5 - 0.5)^2 + (0 - 0.66)^2]

= 4.17

$e_2^2 = [(5 - 5)^2 + (0 - 1)^2] + [(5 - 5)^2 + (2 - 1)^2] = 2$

The total square error are:

$E^2 = e_1^2 + e_2^2 = 4.17 + 2.0 = 6.17.$

After the first iteration, the total square error is significantly reduced from the value 27.48 *to* 6.17. The first iteration was at the same time the final one because if we analyze the distances between the new centroids and the samples, the latter will all be assigned to the same *clusters.* There is no reassignment and therefore the algorithm halts.

3.7.2 K – MEDOIDS METHOD

This algorithm is very similar to k-Means with the small exception of instead of creating an artificial point to recalculate the mean point, k-Medoids recalculates from the nearest, actual point in a data set. The reason for this is that it is not acceptable to outliers that are extremely far away from it.

The K- medoid algorithm, the object itself is considered as the reference point, this object is called representative object and then each remaining object is clustered with the representative object to which it is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. The absolute-error criterion is defined as

$$E = \sum_{j=1}^{k} \sum_{p \in c_j} |p - o_j|$$

Where E is the sum of absolute error for all objects in the dataset, P is the point in space representing a given object in cluster C_j and O_j is the representative object of C_j. In general, the algorithm iterates until, eventually, each representative object is actually the medoid, or most centrally located object, of its cluster. This is the basis of the *k*-medoids method for grouping *n* objects into *k* clusters.

PAM(partitioning Around Medoids) uses k-medoid to identify clusters. PAM selects k objects arbitrarily from the database as medoids. Each of these k objects are representatives of k-classes. Other objects or data in the database are classified based on their distances to these medoids. The algorithm starts with arbitrarily selected k-medoids and iteratively improves the medoid. To calculate the effect of swapping a medoid with a non medoid object, a cost is computed, which is related to the quality of partitioning the non-selected objects to k-clusters represented by the medoids.

The k-mediods algorithm works on following steps:

1. **Randomly select k objects that represent a reference point, medoids.**
2. **Assign remaining object to a cluster that is most similar to its medoid**

3. **Randomly select a non-medoid object O_{random}**
4. **Calculate the total cost C of swapping the medoid O_i with non-medoids object O_{random} by cost function that is used to determine the dissimilarity between O_{random} and O_i.**
5. **Swap medoid O_i with non medoid object O_{random} to take new medoids, if total cost of swapping is negative**
6. **Steps 2 to 5 is again repeated until no swapping occurs.**

The quality of swapping is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster. To determine whether a nonrepresentative object, o_{random}, is a good replacement for a current representative object, o_j, the following four cases are examined for each of the nonrepresentative objects, ***p.***

- **Case1:** p currently belongs to representative object O_j. If O_j is replaced by O_{random} as a representative object and p is closest to one of the other representative objects, O_i, i !=j , then p is reassigned to O_i.
- **Case 2:** p currently belongs to representative object O_j. If O_j is replaced by O_{random} as a representative object and p is closest to O_{random}, then p is reassigned to O_{random}.
- **Case 3:** p currently belongs to representative object O_i, i!=j. If O_j is replaced by O_{random} as a representative object and p is still closest to O_i, then assignment does not change.
- **Case 4:** p currently belongs to representative object O_i, i!=j. If O_j is replaced by O_{random} as a representative object and p is closest to O_{random}, then p is reassigned to O_{random}.

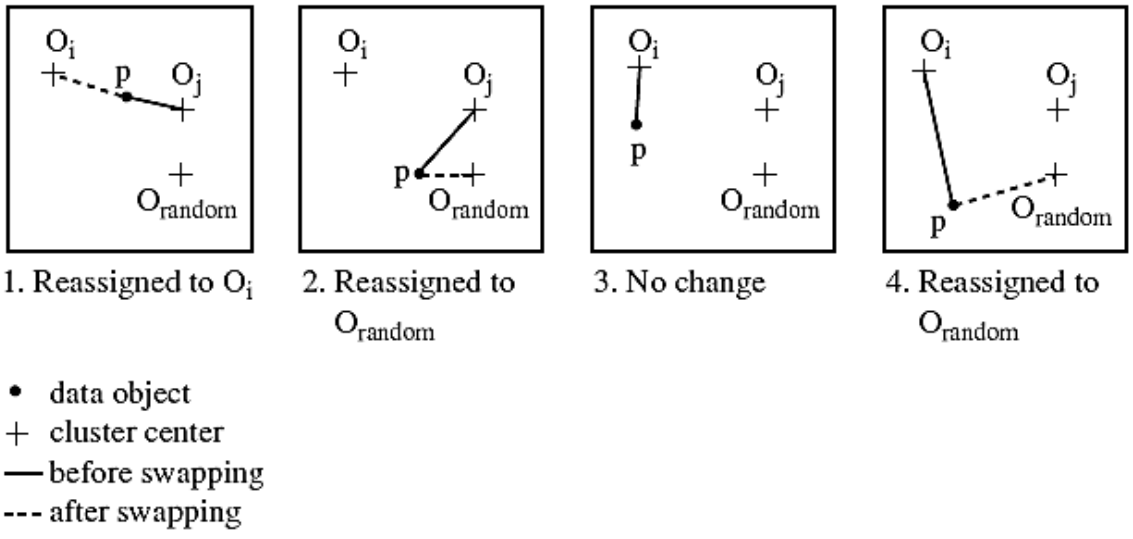

Figure 3.9 Four cases of the cost function for *k*-medoids clustering.

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

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

Input : k : No. of clusters

D: a dataset containing n objects

Output: A set of k clusters

Method:

1) Arbitrarily choose k objects in D as the initial representative objects or seeds
2) Repeat
3) assign each remaining object to the cluster with the nearest representative object;
4) Randomly select a non representative object O_{random};
5) Compute the total cost, S, of swapping representative object, O_j, with O_{random}.
6) If S<0 then swap O_j with O_{random} to form the new set of k representative objects;
7) Until no change;

PAM is ***very robust to the existence of outliers***. The ***clusters*** found this method ***do not depend on the order in which the objects are examined***. However, ***it cannot handle very large volumes of data***.

3.7.3 PARTITIONING METHODS IN LARGE DATABASES

CLARA

CLARA (Clustering Large Apllications) overcomes the problem of scalability that k-Medoids suffers from. CLARA draws a sample of a data set, and applies PAM on this sample to determine the optimal set of mediods from the sample. It then classifies the remaining objects using the partitioning principle. If the sample were drawn in a sufficiently random way, the methods of the sample would approximate the mediods of the entire data set.

CLARA Algorithm

Input : Database D, k:no. of partitions

Repeat for m times

Draw a sample S⊆ *D*randomly from D

Call PAM(S, k) to get k-medoids

Classify the entire data set D to C_1,C_2,..C_k

Calculate the quality of clustering as the average dissimilarity.

End

To improve the quality of CLARA, we go for **CLARANS**(Clustering Large Applications based on RANdomized Search). Instead of clustering on the whole dataset, CLARANS clusters on a sample of the database.

CLARANS works as follows:

- CLARANS draws sample with some randomness in each step of search.
- At each step, PAM examines all of the neighbors of the current node in its search for a minimum cost solution. The current node is then replaced by the neighbor with the largest descent in costs.
- If a better neighbor is found (i.e., having a lower error), CLARANS moves to the neighbour's node and the process starts again;
- Otherwise, the current clustering produces a local minimum. If a local minimum is found, CLARANS starts with new randomly selected nodes in search for a new local minimum.
- Once a user-specified number of local minima has been found, the algorithm outputs, as a solution, the best local minimum, that is, the local minimum having the lowest cost.

Advantages

- More efficient than k-means and k-mediods.
- Used to detect outliers.

Disadvantages

- May not find a real local minimum due to the trimming of its searching.
- It assumes that all objects fit into the main memory, and the result is sensitive to input order.

3.8 HIERARCHICAL CLUSTERING

A hierarchical clustering method works by grouping data objects into a tree of clusters. This hierarchical tree shows levels of clustering with each level having a larger number of smaller clusters. Hierarchical clustering methods can be further classified as either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top-down (splitting) fashion. In hierarchical clustering, we can't backtrack a particular merge or split.

Thus the hierarchical algorithms create a hierarchical decomposition of the set of data using some criterion. The hierarchical algorithm differs how the sub clusters are created. A tree structure called **dendrogram**, can be used to illustrate the hierarchical clustering technique and the sets of different clusters. The root in the dendrogram tree contains one cluster where all elements are together. The leaves in the dendrogram consist of sub clusters. Internal nodes in the dendrogram represents sub clusters formed by merging the clusters that appear as its children in the tree. Each level in the tree is associated with the distance measure that was used to merge the clusters. All clusters created at a particular level were combined because the children clusters had a distance between them less than the distance value associated with this level in the tree. A dendrogram is shown in Figure 3.10 .

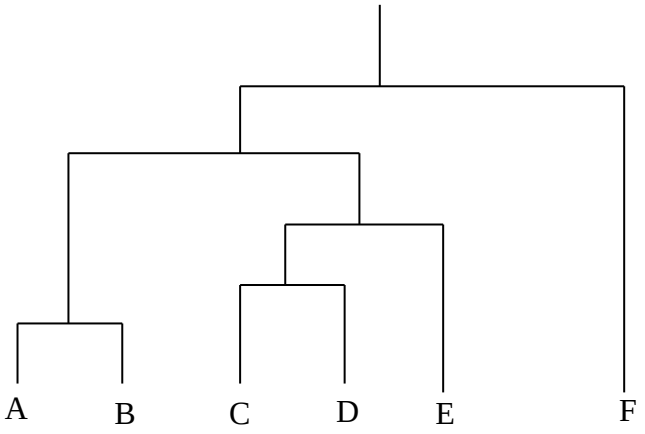

Figure 3.10 : A dendrogram

3.8.1 AGGLOMERATIVE AND DIVISIVE HIERARCHICAL CLUSTERING

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

Agglomerative hierarchical clustering:

This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering

methods belong to this category. They differ only in their definition of inter cluster similarity. The general *algorithm for agglomerative clustering* is as follows:

1. Start with as many clusters as there are records, with one record in each cluster.
2. Combine the two nearest clusters into a larger cluster.
3. Continue until only one cluster remains.

An application of agglomerative clustering is **AGNES(AGglomerative NESting).**

Divisive hierarchical clustering:

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

The general **algorithm for divisive clustering** is as follows:

1. Start with one cluster that contains all the records in the database.
2. Determine the division of the existing cluster that best maximizes similarity within clusters and dissimilarity between clusters.
3. Divide the cluster and repeat on the two smaller clusters.
4. Stop when some minimum threshold of cluster size or total number has been reached or when there is only one record in the cluster.

Example: Agglomerative versus divisive hierarchical clustering:

Figure 3.11 shows the application of **AGNES** (AGglomerative NESting), an agglomerative hierarchical clustering method, and **DIANA** (DIvisive ANAlysis), a divisive hierarchical clustering method, to a data set of five objects, {***a***, ***b***, ***c***, ***d, e*** }.

Initially, AGNES places each object into a cluster of its own. The clusters are then merged step-by-step according to some criterion. For example, clusters *C*1 and *C*2 may be merged if an object in *C*1 and an object in *C*2 form the minimum Euclidean distance between any two objects from different clusters. This is a single-linkage approach in that each cluster is represented by all of the objects in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. The cluster merging process repeats until all of the objects are eventually merged to form one cluster.

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

Major weakness of agglomerative clustering methods

- do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
- can never undo what was done previously

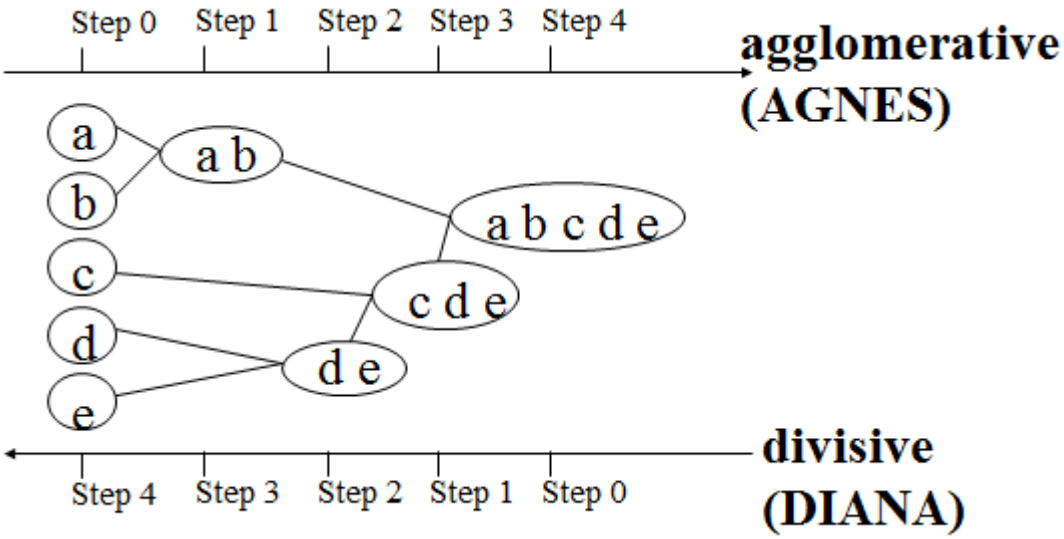

Figure 3.11 Agglomerative and divisive hierarchical clustering on data objects { ***a***, ***b***, ***c***, ***d***, ***e*** }.

3.8.2 BIRCH: BALANCED ITERATIVE REDUCING AND CLUSTERING USING HIERARCHIES

BIRCH partitions objects hierarchically using tree structures and then refines the clusters using other clustering methods. It defines a clustering feature and an associated tree structure that summarizes a cluster. The tree (CF tree) is a height-balanced tree that stores cluster information. BIRCH doesn't produce spherical clusters and may produce unintended cluster. These structures help the clustering method achieve good speed and scalability in large databases and also make it effective for incremental and dynamic clustering of incoming objects.

BIRCH applies a *multiphase* clustering technique, a single scan of the data set yields **a** basic good clustering, and one or more additional scans can (optionally) be used to further improve the quality.

The primary phases are:

Phase 1: BIRCH scans the database to build an initial in-memory CF tree, which can be viewed as a multilevel compression of the data that tries to preserve the inherent clustering structure of the data.

Phase 2: BIRC applies a (selected) clustering algorithm to cluster the leaf nodes of the CF **tree,** which removes sparse clusters as outliers and groups dense clusters into larger ones.

Clustering features

- Summary of the statistics for a given subcluster, the 0th, 1st and 2nd moments of the subcluster from the statistical point of view.
- Registers crucial measurements for computing cluster and utilizes storage efficiently.
- **Clustering Feature:** ***CF = (N, LS, SS)***

where *N*: **Number of data points,** $LS: \sum_{i=1}^{N} X_i$, $SS: \sum_{i=1}^{N} X_i^2$

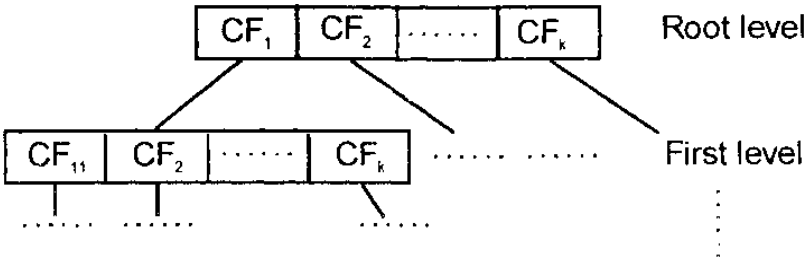

Figure 3.12: A CF tree

- A CF tree is a height balanced tree that stores the clustering features for a hierarchical clustering.
  - A non leaf node in a tree has descendents or "Children".
  - The non leaf nodes stores sums of the CFs of their children.
- A CF tree has two parameters
  - Branching factor : specify the maximum number of children
  - Threashold : Maximum diameter of sub-clusters stored at the leaf nodes.

Drawback

- Handles only numeric data, and sensitive to the order of the data record.

3.8.3 ROCK: A Hierarchical Clustering Algorithm for Categorical Attributes

ROCK (RObust Clustering using linKs) is a hierarchical clustering algorithm that explores the concept of *links* (the number of *common neighbors* between two objects) for data with categorical attributes. Most clustering algorithms assess only the similarity between points when clustering; that is, at each step, points that are the most similar are merged into a single cluster. This "localized" approach is prone to errors. For example, two distinct clusters may have a few points or outliers that are close; therefore, relying on the similarity between points to make clustering decisions could cause the two clusters to be merged. ROCK takes a more global approach to clustering by considering the *neighborhoods* of individual pairs of points. If two

similar points also have similar neighborhoods, then the two points likely belong to the same cluster and so can be merged.

Two points, ***pi*** and ***pj***, are neighbors if *sim*(***pi***, ***pj***) _ q, where *sim* is a similarity function and q is a user-specified threshold. We can choose *sim* to be a distance metric or even a nonmetric that is normalized so that its values fall between 0 and 1, with larger values indicating that the points are more similar. The number of links between ***pi*** and ***pj*** is defined as the number of common neighbors between ***pi*** and ***pj***. If the number of links between two points is large, then it is more likely that they belong to the same cluster. By considering neighboring data points in the relationship between individual pairs of points, ROCK is more robust than standard clustering methods that focus only on point similarity.

Example:

Suppose that a market basket database contains transactions regarding the items a, b, ...g. Consider 2 cluster of transactions C1 and C2. C1 references items <a, b, c, d, e> and C2 references <a, b, f, g>

- Clusters contains transactions
  - **C_1**: {a, b, c}, {a, b, d}, {a, b, e}, {a, c, d}, {a, c, e}, {a, d, e}, {b, c, d}, {b, c, e}, {b, d, e}, {c, d, e}
  - **C_2**: {a, b, f}, {a, b, g}, {a, f, g}, {b, f, g}
- Jaccard co-efficient may lead to wrong clustering result
  - Between transactions ({a, b, c}, {b, d, e}) $\frac{1}{5}$ is 0.2
  - Between transactions ({a, b, c}, {a, b, d}) of C_1 is 0.5
  - Jaccard co-efficient between C_1 & C_2: could be as high as 0.5 ({a, b, c}, {a, b, f})
- Links: Number of common neighbors
  - C_1 <a, b, c, d, e>: <u>{a, b, c},</u> {a, b, d}, {a, b, e}, {a, c, d}, {a, c, e}, {a, d, e}, {b, c, d}, {b, c, e}, {b, d, e}, <u>{c, d, e}</u>
  - C_2 <a, b, f, g>: <u>{a, b, f},</u> {a, b, g}, {a, f, g}, {b, f, g}
- Let T_1 = {a, b, c}, T_2 = {c, d, e}, T_3 = {a, b, f}
  - link(T_1, T_2) = 4, *since they have 4 common neighbors*
    - {a, c, d}, {a, c, e}, {b, c, d}, {b, c, e}
  - link(T_1, T_3) = 3, *since they have 3 common neighbors*
    - {a, b, d}, {a, b, e}, {a, b, g}
- Thus link is a better measure than Jaccard coefficient

3.8.4 CHAMELEON: A HIERARCHICAL CLUSTERING ALGORITHM USING DYNAMIC MODELING

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. It was derived based on the observed weaknesses of two hierarchical clustering algorithms: ROCK and CURE. ROCK and related schemes emphasize cluster interconnectivity while ignoring information regarding cluster proximity. In Chameleon, cluster similarity is assessed based on how well-connected objects are within a cluster *and* on the proximity of clusters. That is, two clusters are merged if their *interconnectivity* is high and they are *close together*. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all types of data as long as a similarity function can be specified.

A two-phase algorithm

1. Use a graph partitioning algorithm: cluster objects into a large number of relatively small sub-clusters
2. Use an agglomerative hierarchical clustering algorithm: find the genuine clusters by repeatedly combining these sub-clusters

The main approach of Chameleon is illustrated in Figure 3.13. Chameleon uses a *k*-nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the *k*-most-similar objects of the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the *k*-nearest-neighbor graph into a large number of relatively small subclusters. It then uses an agglomerative hierarchical clustering algorithm that repeatedly merges subclusters based on their similarity. To determine the pairs of most similar subclusters, it takes into account both the interconnectivity as well as the closeness of the clusters.

3.9 GRID-BASED METHODS

Grid based methods are based on a multiple level granularity structure. Grid based method quantize the object space into finite number of cells that form a grid structure. All the clustering operations are performed on the grid structure. The main advantage of this approach is its fast processing time, which is dependent on only on the number of cells in each dimension of the quantized space. **STING** and **CLIQUE** are the typical examples of the grid based method.

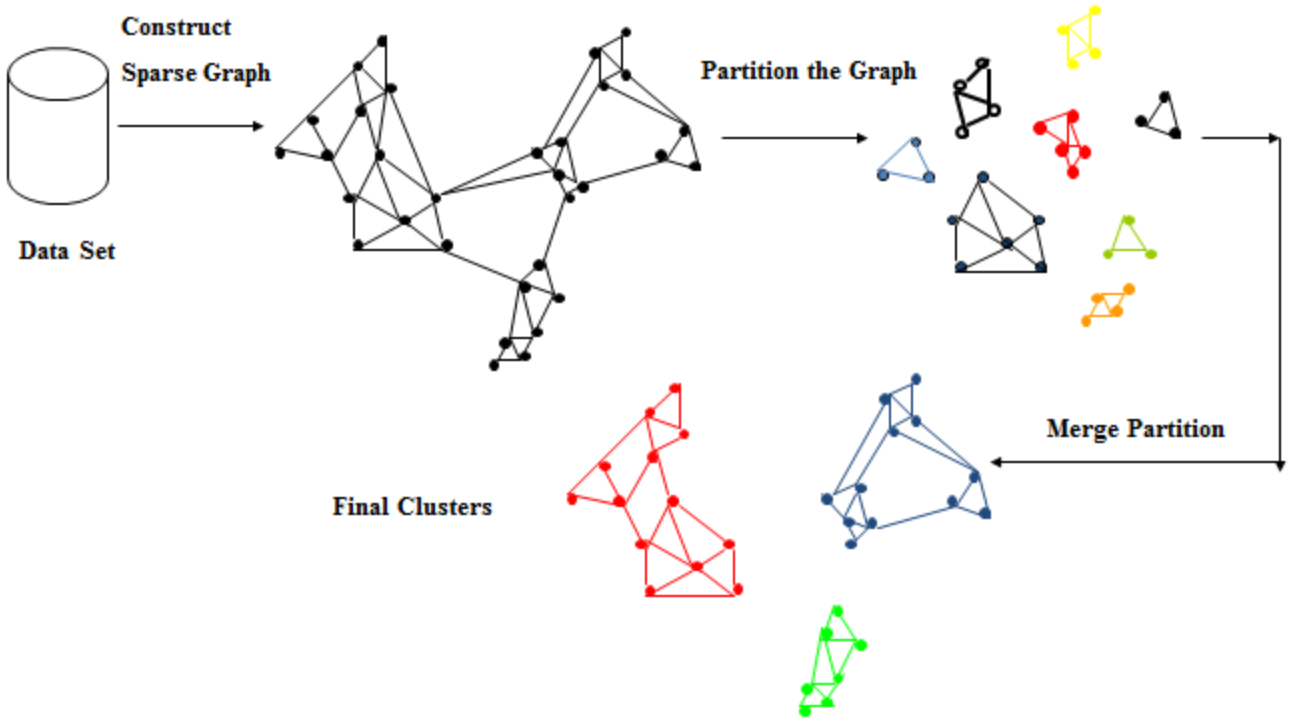

Figure 3.13 Chameleon: Hierarchical clustering based on *k*-nearest neighbors and dynamic modeling.

3.9.1 STING:STatistical INformation Grid

STING is a grid-based multiresolution clustering technique in which 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. Statistical information regarding the attributes in each grid cell (such as the mean, maximum, and minimum values) is precomputed and stored before hand and is used to answer queries.

Figure 3.14 shows a hierarchical structure for STING clustering. Statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum); and the type of *distribution* that the attribute value in the cell follows, such as *normal*, *uniform*, *exponential*, or *none* (if the distribution is unknown).When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as the c2 test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If the distributions of the lower level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

The statistical parameters can be used in a top-down, grid-based method as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated range of probability) reflecting the cell's relevancy to the given query. The irrelevant cells are removed from further consideration. Processing of the next lower level examines only the remaining relevant cells. This process is repeated until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data that fall into the relevant cells are retrieved and further processed until they meet the requirements of the query.

The advantages of this approaches are:

(1) The grid-based computation is *query-independent*, because the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query;

(2) The grid structure facilitates parallel processing and incremental updating; and

(3) The method's efficiency is a major advantage.

STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters is $O(n)$, where n is the total number of objects. After generating the hierarchical structure, the query processing time is $O(g)$, where g is the total number of grid cells at the lowest level, which is usually much smaller than n.

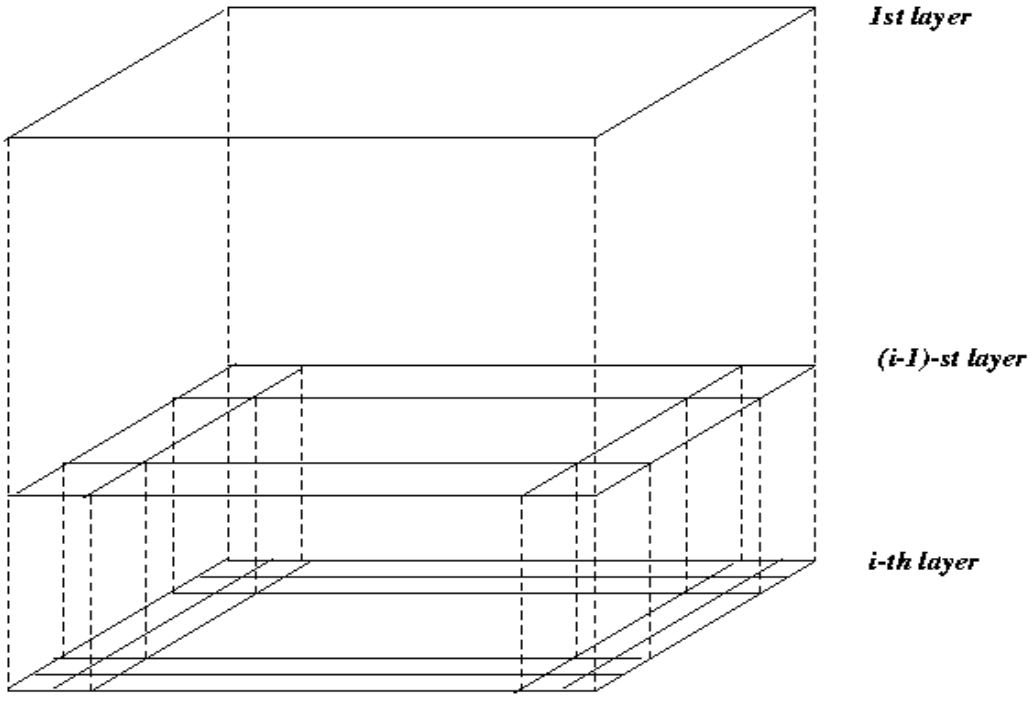

Figure 3.14 A hierarchical structure for STING clustering.

3.9.2 WAVE CLUSTER: CLUSTERING USING WAVELET TRANSFORMATION

Wave Cluster is a multi-resolution clustering algorithm that first summarizes the data by imposing a multidimensional grid structure onto the data space. It then uses a *wavelet transformation* to transform the original feature space, finding dense regions in the transformed space.

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

A wavelet transform is a signal processing technique that decomposes a signal into different frequency sub bands. The wavelet model can be applied to *d*-dimensional signals by applying a one-dimensional wavelet transform *d* times. In applying a wavelet transform, data are transformed so as to preserve the relative distance between objects at different levels of resolution. This allows the natural clusters in the data to become more distinguishable. Clusters can then be identified by searching for dense regions in the new domain.

It offers the following advantages:

- ***It provides unsupervised clustering:*** It uses hat-shaped filters that emphasize regions where the points cluster, while suppressing weaker information outside of the cluster boundaries. Thus, dense regions in the original feature space act as attractors for nearby points and as inhibitors for points that are further away. This means that the clusters in the data automatically stand out and "clear" the regions around them. Thus, another advantage is that wavelet transformation can automatically result in the removal of outliers
- ***The multiresolution property of wavelet transformations can help detect clusters at varying levels of accuracy:*** For example, Figure 3.15 shows a sample of two dimensional feature space, where each point in the image represents the attribute or feature values of one object in the spatial data set. Figure 3.16 shows the resulting wavelet transformation at different resolutions, from a fine scale (scale 1) to a coarse scale (scale 3). At each level, the four subbands into which the original data are decomposed are shown. The subband shown in the upper-left quadrant emphasizes the average neighborhood around each data point. The subband in the upper-right quadrant emphasizes the horizontal edges of the data. The subband in the lower-left quadrant emphasizes the vertical edges, while the subband in the lower-right quadrant emphasizes the corners.
- ***Wavelet-based clustering is very fast:*** with a computational complexity of $O(n)$, where n is the number of objects in the database. The algorithm implementation can be made parallel.

Wave Cluster is a grid-based and density-based algorithm. It conforms with many of the requirements of a good clustering algorithm: It handles large data sets efficiently, discovers clusters with arbitrary shape, successfully handles outliers, is insensitive to the order of input, and does not require the specification of input parameters such as the number of clusters or a neighborhood radius. .

Figure 3.15 A sample of two-dimensional feature space

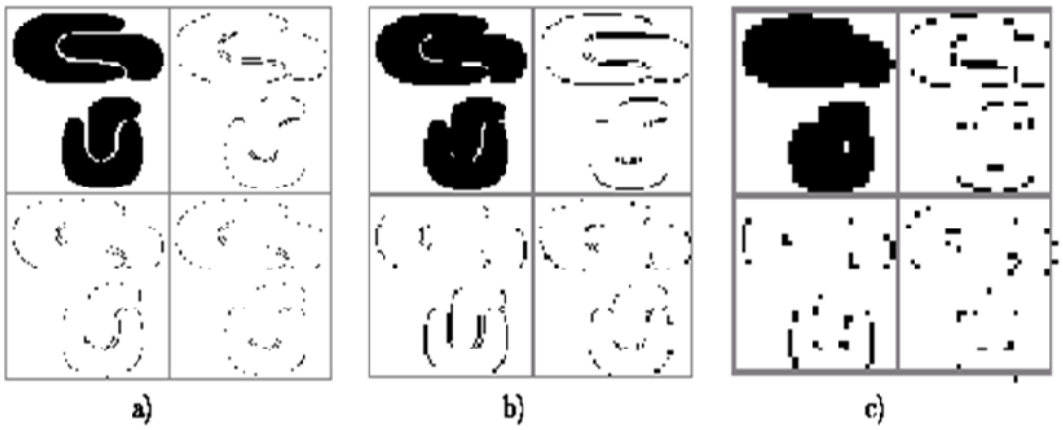

Figure 3.16 Multiresolution of the feature space in Figure 7.16 at (a) scale 1 (high resolution); (b) scale 2 (medium resolution); and (c) scale 3 (low resolution).