

Q.2 a. Discuss data mining task primitives in detail.**(2x4)****Answer:** Data Mining Task Primitives

- We can specify a data mining task in the form of a data mining query.
- This query is input to the system.
- A data mining query is defined in terms of data mining task primitives.

Note – These primitives allow us to communicate in an interactive manner with the data mining system. Here is the list of Data Mining Task Primitives –

- Set of task relevant data to be mined.
- Kind of knowledge to be mined.
- Background knowledge to be used in discovery process.
- Interestingness measures and thresholds for pattern evaluation.
- Representation for visualizing the discovered patterns.

Set of task relevant data to be mined

This is the portion of database in which the user is interested. This portion includes the following –

- Database Attributes
- Data Warehouse dimensions of interest

Kind of knowledge to be mined

It refers to the kind of functions to be performed. These functions are –

- Characterization
- Discrimination
- Association and Correlation Analysis
- Classification
- Prediction
- Clustering
- Outlier Analysis
- Evolution Analysis

Background knowledge

The background knowledge allows data to be mined at multiple levels of abstraction. For example, the Concept hierarchies are one of the background knowledge that allows data to be mined at multiple levels of abstraction.

Interestingness measures and thresholds for pattern evaluation

This is used to evaluate the patterns that are discovered by the process of knowledge discovery. There are different interesting measures for different kind of knowledge.

Representation for visualizing the discovered patterns

This refers to the form in which discovered patterns are to be displayed. These representations may include the following –

- Rules
- Tables
- Charts
- Graphs
- Decision Trees
- Cubes

b. Describe briefly the Data Mining functionalities, highlighting the kind of patterns that can be mined.**(8)**

Answer:

1.4 Data Mining Functionalities—What Kinds of Patterns Can Be Mined?

We have observed various types of databases and information repositories on which data mining can be performed. Let us now examine the kinds of data patterns that can be mined.

Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. In general, data mining tasks can be classified into two categories: descriptive and predictive. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks perform inference on the current data in order to make predictions.

In some cases, users may have no idea regarding what kinds of patterns in their data may be interesting, and hence may like to search for several different kinds of patterns in parallel. Thus it is important to have a data mining system that can mine multiple kinds of patterns to accommodate different user expectations or applications. Furthermore, data mining systems should be able to discover patterns at various granularity (i.e., different levels of abstraction). Data mining systems should also allow users to specify hints to guide or focus the search for interesting patterns. Because some patterns may not hold for all of the data in the database, a measure of certainty or "trustworthiness" is usually associated with each discovered pattern.

Data mining functionalities, and the kinds of patterns they can discover, are described below.

1.4.1 Concept/Class Description: Characterization and Discrimination

Data can be associated with classes or concepts. For example, in the *AllElectronics* store, classes of items for sale include *computers* and *printers*, and concepts of customers include *bigSpenders* and *budgetSpenders*. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms. Such descriptions of a class or a concept are called class/concept descriptions. These descriptions can be derived via (1) *data characterization*, by summarizing the data of the class under study (often called

the target class) in general terms, or (2) *data discrimination*, by comparison of the target class with one or a set of comparative classes (often called the contrasting classes), or (3) both data characterization and discrimination.

Data characterization is a summarization of the general characteristics or features of a target class of data. The data corresponding to the user-specified class are typically collected by a database query. For example, to study the characteristics of software products whose sales increased by 10% in the last year, the data related to such products can be collected by executing an SQL query.

There are several methods for effective data summarization and characterization. Simple data summaries based on statistical measures and plots are described in Chapter 2. The data cube-based OLAP roll-up operation (Section 1.3.2) can be used to perform user-controlled data summarization along a specified dimension. This process is further detailed in Chapters 3 and 4, which discuss data warehousing. An *attribute-oriented induction* technique can be used to perform data generalization and characterization without step-by-step user interaction. This technique is described in Chapter 4.

The output of data characterization can be presented in various forms. Examples include pie charts, bar charts, curves, multidimensional data cubes, and multidimensional tables, including crosstabs. The resulting descriptions can also be presented as generalized relations or in rule form (called characteristic rules). These different output forms and their transformations are discussed in Chapter 4.

Example 1.4 Data characterization. A data mining system should be able to produce a description summarizing the characteristics of customers who spend more than \$1,000 a year at *AllElectronics*. The result could be a general profile of the customers, such as they are 40–50 years old, employed, and have excellent credit ratings. The system should allow users to drill down on any dimension, such as on *occupation* in order to view these customers according to their type of employment. ■

Data discrimination is a comparison of the general features of target class data objects with the general features of objects from one or a set of contrasting classes. The target and contrasting classes can be specified by the user, and the corresponding data objects retrieved through database queries. For example, the user may like to compare the general features of software products whose sales increased by 10% in the last year with those whose sales decreased by at least 30% during the same period. The methods used for data discrimination are similar to those used for data characterization.

“How are discrimination descriptions output?” The forms of output presentation are similar to those for characteristic descriptions, although discrimination descriptions should include comparative measures that help distinguish between the target and contrasting classes. Discrimination descriptions expressed in rule form are referred to as *discriminant rules*.

Example 1.5 Data discrimination. A data mining system should be able to compare two groups of *AllElectronics* customers, such as those who shop for computer products regularly (more

than two times a month) versus those who rarely shop for such products (i.e., less than three times a year). The resulting description provides a general comparative profile of the customers, such as 80% of the customers who frequently purchase computer products are between 20 and 40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either seniors or youths, and have no university degree. Drilling down on a dimension, such as *occupation*, or adding new dimensions, such as *income_level*, may help in finding even more discriminative features between the two classes. ■

Concept description, including characterization and discrimination, is described in Chapter 4.

1.4.2 Mining Frequent Patterns, Associations, and Correlations

Frequent patterns, as the name suggests, are patterns that occur frequently in data. There are many kinds of frequent patterns, including itemsets, subsequences, and substructures. A *frequent itemset* typically refers to a set of items that frequently appear together in a transactional data set, such as milk and bread. A frequently occurring subsequence, such as the pattern that customers tend to purchase first a PC, followed by a digital camera, and then a memory card, is a (*frequent*) *sequential pattern*. A substructure can refer to different structural forms, such as graphs, trees, or lattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.

Example 1.6 Association analysis. Suppose, as a marketing manager of *AllElectronics*, you would like to determine which items are frequently purchased together within the same transactions. An example of such a rule, mined from the *AllElectronics* transactional database, is

$$\text{buys}(X, \text{"computer"}) \Rightarrow \text{buys}(X, \text{"software"}) \quad [\text{support} = 1\%, \text{confidence} = 50\%]$$

where X is a variable representing a customer. A confidence, or certainty, of 50% means that if a customer buys a computer, there is a 50% chance that she will buy software as well. A 1% support means that 1% of all of the transactions under analysis showed that computer and software were purchased together. This association rule involves a single attribute or predicate (i.e., *buys*) that repeats. Association rules that contain a single predicate are referred to as single-dimensional association rules. Dropping the predicate notation, the above rule can be written simply as "*computer* \Rightarrow *software* [1%, 50%]".

Suppose, instead, that we are given the *AllElectronics* relational database relating to purchases. A data mining system may find association rules like

$$\text{age}(X, \text{"20...29"}) \wedge \text{income}(X, \text{"20K...29K"}) \Rightarrow \text{buys}(X, \text{"CD player"}) \\ [\text{support} = 2\%, \text{confidence} = 60\%]$$

The rule indicates that of the *AllElectronics* customers under study, 2% are 20 to 29 years of age with an income of 20,000 to 29,000 and have purchased a CD player

at *AllElectronics*. There is a 60% probability that a customer in this age and income group will purchase a CD player. Note that this is an association between more than one attribute, or predicate (i.e., *age*, *income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a multidimensional association rule. ■

Typically, association rules are discarded as uninteresting if they do not satisfy both a minimum support threshold and a minimum confidence threshold. Additional analysis can be performed to uncover interesting statistical correlations between associated attribute-value pairs.

Frequent itemset mining is the simplest form of frequent pattern mining. The mining of frequent patterns, associations, and correlations is discussed in Chapter 5, where particular emphasis is placed on efficient algorithms for frequent itemset mining. Sequential pattern mining and structured pattern mining are considered advanced topics. They are discussed in Chapters 8 and 9, respectively.

1.4.3 Classification and Prediction

Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data (i.e., data objects whose class label is known).

"How is the derived model presented?" The derived model may be represented in various forms, such as *classification (IF-THEN) rules*, *decision trees*, *mathematical formulae*, or *neural networks* (Figure 1.10). A decision tree is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can easily be converted to classification rules. A neural network, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and *k*-nearest neighbor classification.

Whereas classification predicts categorical (discrete, unordered) labels, prediction models continuous-valued functions. That is, it is used to predict missing or unavailable *numerical data values* rather than class labels. Although the term *prediction* may refer to both numeric prediction and class label prediction, in this book we use it to refer primarily to numeric prediction. Regression analysis is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Prediction also encompasses the identification of distribution *trends* based on the available data.

Classification and prediction may need to be preceded by *relevance analysis*, which attempts to identify attributes that do not contribute to the classification or prediction process. These attributes can then be excluded.

Example 1.7 Classification and prediction. Suppose, as sales manager of *AllElectronics*, you would like to classify a large set of items in the store, based on three kinds of responses to a

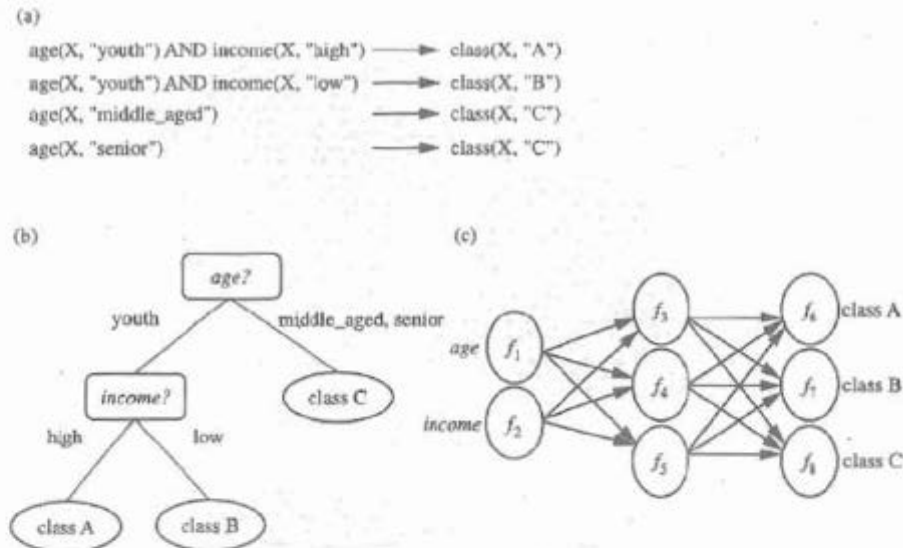


Figure 1.10 A classification model can be represented in various forms, such as (a) IF-THEN rules, (b) a decision tree, or a (c) neural network.

sales campaign: *good response*, *mild response*, and *no response*. You would like to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand*, *place_made*, *type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set. Suppose that the resulting classification is expressed in the form of a decision tree. The decision tree, for instance, may identify *price* as being the single factor that best distinguishes the three classes. The tree may reveal that, after *price*, other features that help further distinguish objects of each class from another include *brand* and *place_made*. Such a decision tree may help you understand the impact of the given sales campaign and design a more effective campaign for the future.

Suppose instead, that rather than predicting categorical response labels for each store item, you would like to predict the amount of revenue that each item will generate during an upcoming sale at *AllElectronics*, based on previous sales data. This is an example of (numeric) prediction because the model constructed will predict a continuous-valued function, or ordered value. ■

Chapter 6 discusses classification and prediction in further detail.

1.4.4 Cluster Analysis

"What is cluster analysis?" Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label.

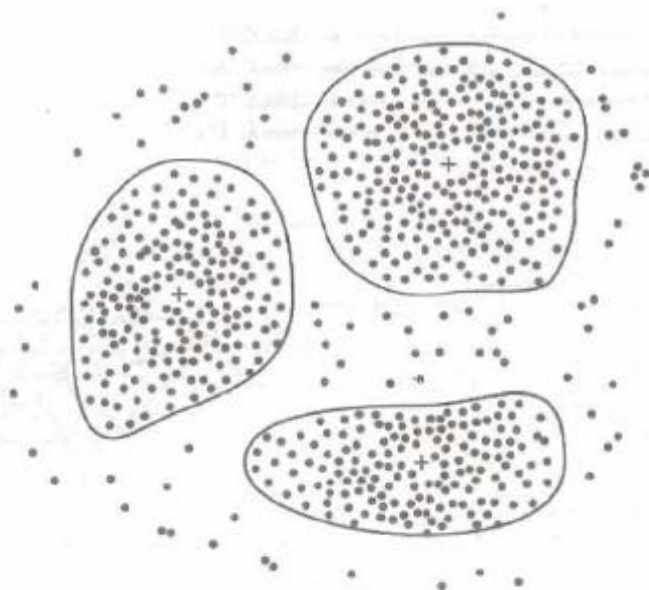


Figure 1.11 A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster “center” is marked with a “+”.

In general, the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped based on the principle of *maximizing the intraclass similarity and minimizing the interclass similarity*. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.

Example 1.8 Cluster analysis. Cluster analysis can be performed on *AllElectronics* customer data in order to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. Figure 1.11 shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident. ■

Cluster analysis forms the topic of Chapter 7.

1.4.5 Outlier Analysis

A database may contain data objects that do not comply with the general behavior or model of the data. These data objects are outliers. Most data mining methods discard

outliers as noise or exceptions. However, in some applications such as fraud detection, the rare events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as outlier mining.

Outliers may be detected using statistical tests that assume a distribution or probability model for the data, or using distance measures where objects that are a substantial distance from any other cluster are considered outliers. Rather than using statistical or distance measures, deviation-based methods identify outliers by examining differences in the main characteristics of objects in a group.

Example 1.9 Outlier analysis. Outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of extremely large amounts for a given account number in comparison to regular charges incurred by the same account. Outlier values may also be detected with respect to the location and type of purchase, or the purchase frequency. ■

Outlier analysis is also discussed in Chapter 7.

1.4.6 Evolution Analysis

Data evolution analysis describes and models regularities or trends for objects whose behavior changes over time. Although this may include characterization, discrimination, association and correlation analysis, classification, prediction, or clustering of *time-related* data, distinct features of such an analysis include time-series data analysis, sequence or periodicity pattern matching, and similarity-based data analysis.

Example 1.10 Evolution analysis. Suppose that you have the major stock market (time-series) data of the last several years available from the New York Stock Exchange and you would like to invest in shares of high-tech industrial companies. A data mining study of stock exchange data may identify stock evolution regularities for overall stocks and for the stocks of particular companies. Such regularities may help predict future trends in stock market prices, contributing to your decision making regarding stock investments. ■

Data evolution analysis is discussed in Chapter 8.

Q.3 a. Describe the process of data cleansing.

(8)

Answer: The process of data cleansing

- **Data auditing:** The data is audited with the use of statistical and database methods to detect anomalies and contradictions: this eventually gives an indication of the characteristics of the anomalies and their locations. Several commercial software packages will let you specify constraints of various kinds (using a grammar that conforms to that of a standard programming language, e.g., JavaScript or Visual Basic) and then generate code that checks the data for violation of these constraints. This process is referred to below in the bullets "workflow specification" and "workflow execution." For users who lack access to high-end cleansing software, Microcomputer database packages such as Microsoft Access or File Maker Pro will also let you perform such checks, on a constraint-by-constraint basis, interactively with little or no programming required in many cases.
- **Workflow specification:** The detection and removal of anomalies is performed by a sequence of operations on the data known as the workflow. It is specified after the process of auditing the data and is crucial in achieving the end product of high-quality data. In order to achieve a proper workflow, the causes of the anomalies and errors in the data have to be closely considered.
- **Workflow execution:** In this stage, the workflow is executed after its specification is complete and its correctness is verified. The implementation of the workflow should be

efficient, even on large sets of data, which inevitably poses a trade-off because the execution of a data-cleansing operation can be computationally expensive.

- **Post-processing and controlling:** After executing the cleansing workflow, the results are inspected to verify correctness. Data that could not be corrected during execution of the workflow is manually corrected, if possible. The result is a new cycle in the data-cleansing process where the data is audited again to allow the specification of an additional workflow to further cleanse the data by automatic processing.

b. Explain the need of data discretization. How is it performed? In the same context explain concept hierarchy generation. (2+3+3)

Answer:

2.6 Data Discretization and Concept Hierarchy Generation

Data discretization techniques can be used to reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. Replacing numerous values of a continuous attribute by a small number of interval labels thereby reduces and simplifies the original data. This leads to a concise, easy-to-use, knowledge-level representation of mining results.

Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals. Discretization can be performed recursively on an attribute to provide a hierarchical or multiresolution partitioning of the attribute values, known as a concept hierarchy. Concept hierarchies are useful for mining at multiple levels of abstraction.

A concept hierarchy for a given numerical attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low-level concepts (such as numerical values for the attribute *age*) with higher-level concepts (such as *youth*, *middle-aged*, or *senior*). Although detail is lost by such data generalization, the generalized data may be more meaningful and easier to interpret. This contributes to a consistent representation of data mining results among multiple mining tasks, which is a common requirement. In addition, mining on a reduced data set requires fewer input/output operations and is more efficient than mining on a larger, ungeneralized data set. Because of these benefits, discretization techniques and concept hierarchies are typically applied before data mining as a preprocessing step, rather than during mining. An example of a concept hierarchy for the attribute *price* is given in Figure 2.22. More than one concept hierarchy can be defined for the same attribute in order to accommodate the needs of various users.

Manual definition of concept hierarchies can be a tedious and time-consuming task for a user or a domain expert. Fortunately, several discretization methods can be used to automatically generate or dynamically refine concept hierarchies for numerical attributes. Furthermore, many hierarchies for categorical attributes are

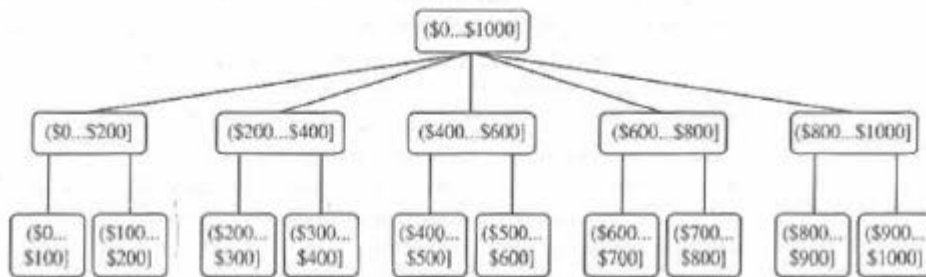


Figure 2.22 A concept hierarchy for the attribute *price*, where an interval $(\$X \dots \$Y]$ denotes the range from $\$X$ (exclusive) to $\$Y$ (inclusive).

implicit within the database schema and can be automatically defined at the schema definition level.

Let's look at the generation of concept hierarchies for numerical and categorical data.

2.6.1 Discretization and Concept Hierarchy Generation for Numerical Data

It is difficult and laborious to specify concept hierarchies for numerical attributes because of the wide diversity of possible data ranges and the frequent updates of data values. Such manual specification can also be quite arbitrary.

Concept hierarchies for numerical attributes can be constructed automatically based on data discretization. We examine the following methods: *binning*, *histogram analysis*, *entropy-based discretization*, χ^2 -*merging*, *cluster analysis*, and *discretization by intuitive partitioning*. In general, each method assumes that the values to be discretized are sorted in ascending order.

Binning

Binning is a top-down splitting technique based on a specified number of bins. Section 2.3.2 discussed binning methods for data smoothing. These methods are also used as discretization methods for numerosity reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning, and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions in order to generate concept hierarchies. Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

Histogram Analysis

Like binning, histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms partition the values for an attribute, A , into disjoint ranges called *buckets*. Histograms were introduced in Section 2.2.3. Partitioning rules for defining histograms were described in Section 2.5.4. In an *equal-width* histogram, for example, the values are partitioned into equal-sized partitions or ranges (such as in Figure 2.19 for *price*, where each bucket has a width of \$10). With an *equal-frequency* histogram, the values are partitioned so that, ideally, each partition contains the same number of data tuples. The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level. Histograms can also be partitioned based on cluster analysis of the data distribution, as described below.

Entropy-Based Discretization

Entropy is one of the most commonly used discretization measures. It was first introduced by Claude Shannon in pioneering work on information theory and the concept of information gain. Entropy-based discretization is a supervised, top-down splitting technique. It explores class distribution information in its calculation and determination of split-points (data values for partitioning an attribute range). To discretize a numerical attribute, A , the method selects the value of A that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. Such discretization forms a concept hierarchy for A .

Let D consist of data tuples defined by a set of attributes and a class-label attribute. The class-label attribute provides the class information per tuple. The basic method for entropy-based discretization of an attribute A within the set is as follows:

1. Each value of A can be considered as a potential interval boundary or split-point (denoted *split_point*) to partition the range of A . That is, a split-point for A can partition the tuples in D into two subsets satisfying the conditions $A \leq \textit{split_point}$ and $A > \textit{split_point}$, respectively, thereby creating a binary discretization.
2. Entropy-based discretization, as mentioned above, uses information regarding the class label of tuples. To explain the intuition behind entropy-based discretization, we must take a glimpse at classification. Suppose we want to classify the tuples in D by partitioning on attribute A and some split-point. Ideally, we would like this partitioning to result in an exact classification of the tuples. For example, if we had two classes, we would hope that all of the tuples of, say, class C_1 will fall into one partition, and all of the tuples of class C_2 will fall into the other partition. However, this is unlikely. For example, the first partition may contain many tuples of C_1 , but also some of C_2 . How much more information would we still need for a perfect classification, after this partitioning? This amount is called the *expected information requirement* for classifying a tuple in D based on partitioning by A . It is given by

$$\textit{Info}_A(D) = \frac{|D_1|}{|D|} \textit{Entropy}(D_1) + \frac{|D_2|}{|D|} \textit{Entropy}(D_2), \quad (2.15)$$

where D_1 and D_2 correspond to the tuples in D satisfying the conditions $A \leq \textit{split_point}$ and $A > \textit{split_point}$, respectively; $|D|$ is the number of tuples in D , and so on. The entropy function for a given set is calculated based on the class distribution of the tuples in the set. For example, given m classes, C_1, C_2, \dots, C_m , the entropy of D_1 is

$$\textit{Entropy}(D_1) = - \sum_{i=1}^m p_i \log_2(p_i), \quad (2.16)$$

where p_i is the probability of class C_i in D_1 , determined by dividing the number of tuples of class C_i in D_1 by $|D_1|$, the total number of tuples in D_1 . Therefore, when selecting a split-point for attribute A , we want to pick the attribute value that gives the minimum expected information requirement (i.e., $\min(\textit{Info}_A(D))$). This would result

in the minimum amount of expected information (still) required to perfectly classify the tuples after partitioning by $A \leq \textit{split_point}$ and $A > \textit{split_point}$. This is equivalent to the attribute-value pair with the maximum information gain (the further details of which are given in Chapter 6 on classification.) Note that the value of $\textit{Entropy}(D_T)$ can be computed similarly as in Equation (2.16).

"But our task is discretization, not classification!", you may exclaim. This is true. We use the split-point to partition the range of A into two intervals, corresponding to $A \leq \textit{split_point}$ and $A > \textit{split_point}$.

3. The process of determining a split-point is recursively applied to each partition obtained, until some stopping criterion is met, such as when the minimum information requirement on all candidate split-points is less than a small threshold, ϵ , or when the number of intervals is greater than a threshold, $\textit{max_interval}$.

Entropy-based discretization can reduce data size. Unlike the other methods mentioned here so far, entropy-based discretization uses class information. This makes it more likely that the interval boundaries (split-points) are defined to occur in places that may help improve classification accuracy. The entropy and information gain measures described here are also used for decision tree induction. These measures are revisited in greater detail in Section 6.3.2.

Interval Merging by χ^2 Analysis

ChiMerge is a χ^2 -based discretization method. The discretization methods that we have studied up to this point have all employed a top-down, splitting strategy. This contrasts with *ChiMerge*, which employs a bottom-up approach by finding the best neighboring intervals and then merging these to form larger intervals, recursively. The method is supervised in that it uses class information. The basic notion is that for accurate discretization, the relative class frequencies should be fairly consistent within an interval. Therefore, if two adjacent intervals have a very similar distribution of classes, then the intervals can be merged. Otherwise, they should remain separate.

ChiMerge proceeds as follows. Initially, each distinct value of a numerical attribute A is considered to be one interval. χ^2 tests are performed for every pair of adjacent intervals. Adjacent intervals with the least χ^2 values are merged together, because low χ^2 values for a pair indicate similar class distributions. This merging process proceeds recursively until a predefined stopping criterion is met.

The χ^2 statistic was introduced in Section 2.4.1 on data integration, where we explained its use to detect a correlation relationship between two categorical attributes (Equation (2.9)). Because *ChiMerge* treats intervals as discrete categories, Equation (2.9) can be applied. The χ^2 statistic tests the hypothesis that two adjacent intervals for a given attribute are independent of the class. Following the method in Example 2.1, we can construct a contingency table for our data. The contingency table has two columns (representing the two adjacent intervals) and m rows, where m is the number of distinct classes. Applying Equation (2.9) here, the cell value o_{ij} is the count of tuples in the i^{th} interval and j^{th} class. Similarly, the expected frequency of o_{ij} is $e_{ij} = (\text{number of tuples in interval$

- $i) \times (\text{number of tuples in class } j)/N$, where N is the total number of data tuples. Low χ^2 values for an interval pair indicate that the intervals are independent of the class and can, therefore, be merged.

The stopping criterion is typically determined by three conditions. First, merging stops when χ^2 values of all pairs of adjacent intervals exceed some threshold, which is determined by a specified significance level. A too (or very) high value of significance level for the χ^2 test may cause overdiscretization, whereas a too (or very) low value may lead to underdiscretization. Typically, the significance level is set between 0.10 and 0.01. Second, the number of intervals cannot be over a prespecified *max-interval*, such as 10 to 15. Finally, recall that the premise behind ChiMerge is that the relative class frequencies should be fairly consistent within an interval. In practice, some inconsistency is allowed, although this should be no more than a prespecified threshold, such as 3%, which may be estimated from the training data. This last condition can be used to remove irrelevant attributes from the data set.

Cluster Analysis

Cluster analysis is a popular data discretization method. A clustering algorithm can be applied to discretize a numerical attribute, A , by partitioning the values of A into clusters or groups. Clustering takes the distribution of A into consideration, as well as the closeness of data points, and therefore is able to produce high-quality discretization results. Clustering can be used to generate a concept hierarchy for A by following either a top-down splitting strategy or a bottom-up merging strategy, where each cluster forms a node of the concept hierarchy. In the former, each initial cluster or partition may be further decomposed into several subclusters, forming a lower level of the hierarchy. In the latter, clusters are formed by repeatedly grouping neighboring clusters in order to form higher-level concepts. Clustering methods for data mining are studied in Chapter 7.

Discretization by Intuitive Partitioning

Although the above discretization methods are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or “natural.” For example, annual salaries broken into ranges like $(\$50,000, \$60,000]$ are often more desirable than ranges like $(\$51,263.98, \$60,872.34]$, obtained by, say, some sophisticated clustering analysis.

The 3-4-5 rule can be used to segment numerical data into relatively uniform, natural-seeming intervals. In general, the rule partitions a given range of data into 3, 4, or 5 relatively equal-width intervals, recursively and level by level, based on the value range at the most significant digit. We will illustrate the use of the rule with an example further below. The rule is as follows:

- If an interval covers 3, 6, 7, or 9 distinct values at the most significant digit, then partition the range into 3 intervals (3 equal-width intervals for 3, 6, and 9; and 3 intervals in the grouping of 2-3-2 for 7).

- ▀ If it covers 2, 4, or 8 distinct values at the most significant digit, then partition the range into 4 equal-width intervals.
- ▀ If it covers 1, 5, or 10 distinct values at the most significant digit, then partition the range into 5 equal-width intervals.

The rule can be recursively applied to each interval, creating a concept hierarchy for the given numerical attribute. Real-world data often contain extremely large positive and/or negative outlier values, which could distort any top-down discretization method based on minimum and maximum data values. For example, the assets of a few people could be several orders of magnitude higher than those of others in the same data set. Discretization based on the maximal asset values may lead to a highly biased hierarchy. Thus the top-level discretization can be performed based on the range of data values representing the majority (e.g., 5th percentile to 95th percentile) of the given data. The extremely high or low values beyond the top-level discretization will form distinct interval(s) that can be handled separately, but in a similar manner.

The following example illustrates the use of the 3-4-5 rule for the automatic construction of a numerical hierarchy.

Q.4 a. Discuss the three-tier data warehouse architecture.

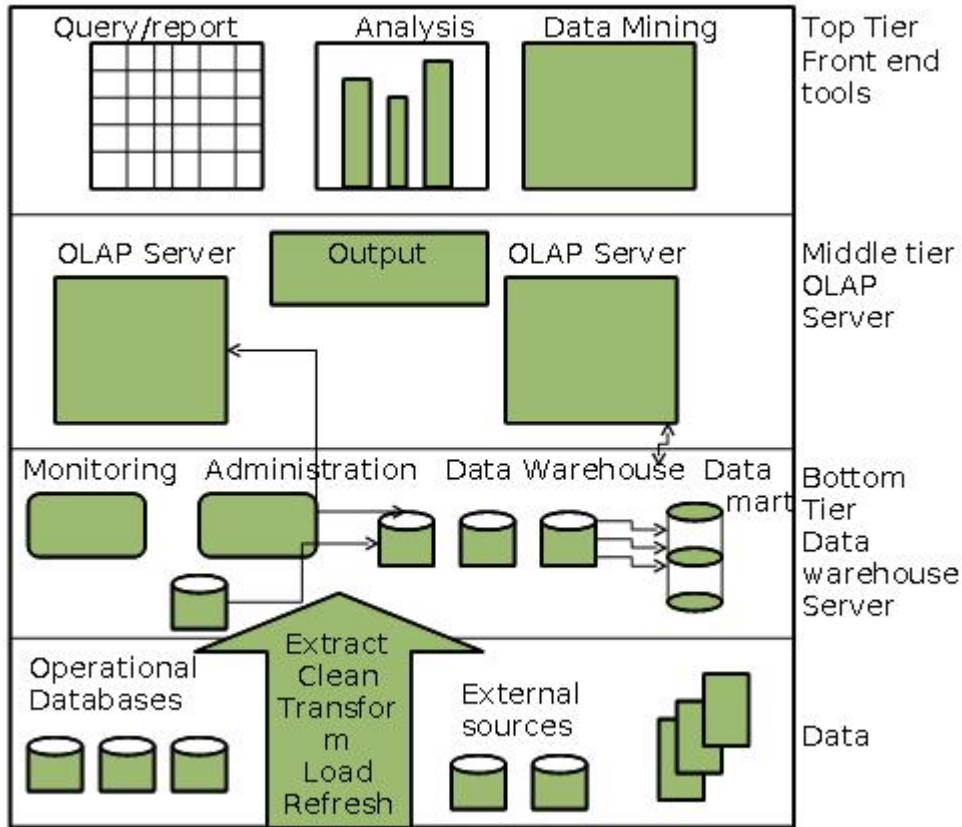
(8)

Answer: Three-Tier Data Warehouse Architecture

Generally a data warehouses adopts a three-tier architecture. Following are the three tiers of the data warehouse architecture.

- **Bottom Tier** - The bottom tier of the architecture is the data warehouse database server. It is the relational database system. We use the back end tools and utilities to feed data into the bottom tier. These back end tools and utilities perform the Extract, Clean, Load, and refresh functions.
- **Middle Tier** - In the middle tier, we have the OLAP Server that can be implemented in either of the following ways.
 - By Relational OLAP (ROLAP), which is an extended relational database management system. The ROLAP maps the operations on multidimensional data to standard relational operations.
 - By Multidimensional OLAP (MOLAP) model, which directly implements the multidimensional data and operations.
- **Top-Tier** - This tier is the front-end client layer. This layer holds the query tools and reporting tools, analysis tools and data mining tools.

The following diagram depicts the three-tier architecture of data warehouse:



b. Explain how data mining system can be integrated with database/datawarehouse system. (8)

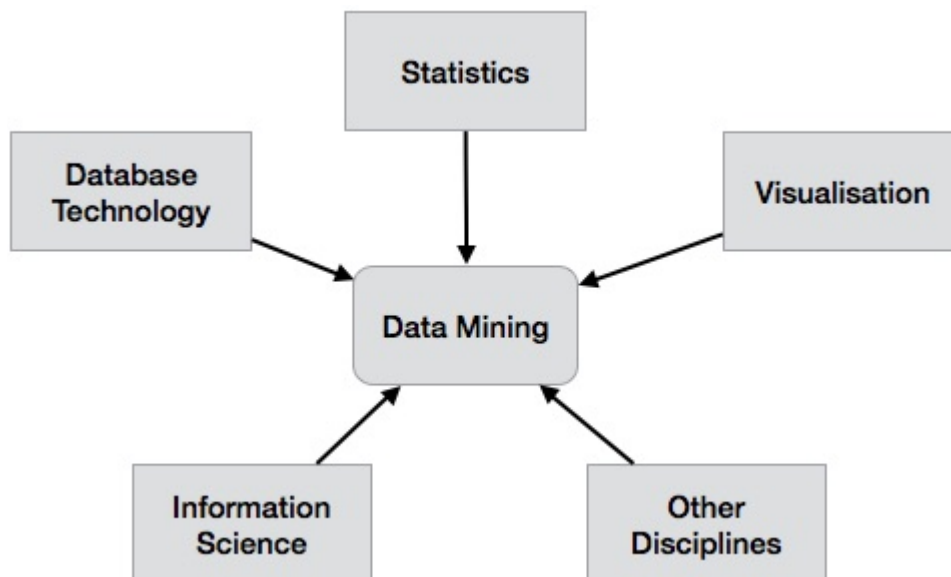
Answer: There is a LARGE variety of data mining systems available. Data mining systems may integrate techniques from the following –

- Spatial Data Analysis
- INFORMATION Retrieval
- Pattern Recognition
- IMAGE Analysis
- Signal Processing
- Computer Graphics
- Web Technology
- BUSINESS
- Bioinformatics

Data Mining System Classification

A data mining SYSTEM can be classified according to the following criteria –

- Database Technology
- Statistics
- Machine Learning
- INFORMATION Science
- Visualization
- OTHER Disciplines



Apart from these, a data mining system can also be classified based on the kind of (a) databases mined, (b) knowledge mined, (c) techniques utilized, and (d) APPLICATIONS adapted.

Classification Based on the Databases Mined

We can classify a data mining system according to the kind of databases mined. Database system can be classified according to different criteria such as data models, TYPES of data, etc. And the data mining system can be classified accordingly.

For example, if we classify a database according to the data model, then we may have a relational, TRANSACTIONAL, object-relational, or data warehouse mining system.

Classification Based on the kind of Knowledge Mined

We can classify a data mining system according to the kind of knowledge mined. It means the data mining system is classified on the BASIS of functionalities such as –

- Characterization
- Discrimination
- Association and Correlation Analysis
- Classification
- Prediction
- Prediction
- Outlier Analysis
- Evolution Analysis

Classification Based on the Techniques Utilized

We can classify a data mining system according to the kind of techniques used. We can describe these techniques according to the degree of USER interaction involved or the methods of analysis employed.

Classification Based on the Applications Adapted

We can classify a data mining system according to the APPLICATIONS adapted. These applications are as follows –

- Finance
- Telecommunications
- DNA
- Stock Markets
- E-MAIL

Integrating a Data Mining System with a DB/DW System

If a data mining system is not integrated with a database or a data warehouse system, then there will be no system to COMMUNICATE with. This scheme is known as the non-coupling scheme. In this scheme, the main focus is on data mining design and on developing efficient and effective algorithms for mining the available data sets.

The list of Integration Schemes is as follows –

- **No Coupling** – In this scheme, the data mining system does not utilize any of the database or data warehouse functions. It fetches the data from a particular source and processes that data using some data mining algorithms. The data mining RESULT is stored in another file.
- **Loose Coupling** – In this scheme, the data mining system may use some of the functions of database and data warehouse system. It fetches the data from the data respiratory managed by these systems and performs data mining on that data. It then stores the mining RESULT either in a file or in a designated place in a database or in a data warehouse.
- **Semi-tight Coupling** - In this scheme, the data mining system is LINKED with a database or a data warehouse system and in addition to that, efficient implementations of a few data mining primitives can be provided in the database.
- **Tight coupling** – In this coupling scheme, the data mining system is smoothly integrated into the database or data warehouse system. The data mining subsystem is treated as one functional component of an INFORMATION system.

Q.5 a. What do you understand by “data cube”? Explain the terms Full cube, Iceberg cube, Closed cube and Shell cube. (2+6)

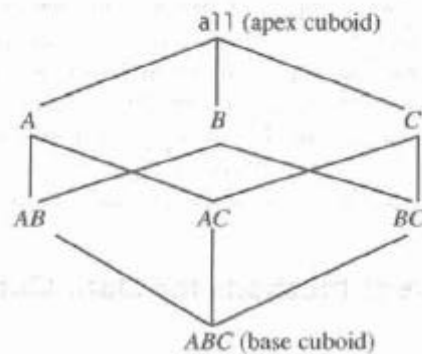
Answer:

4.1.1 A Road Map for the Materialization of Different Kinds of Cubes

// Data cubes facilitate the on-line analytical processing of multidimensional data. "But how can we compute data cubes in advance, so that they are handy and readily available for query processing?" This section contrasts full cube materialization (i.e., precomputation) versus various strategies for partial cube materialization. For completeness, we begin with a review of the basic terminology involving data cubes. We also introduce a cube cell notation that is useful for describing data cube computation methods.

Cube Materialization: Full Cube, Iceberg Cube, Closed Cube, and Shell Cube

Figure 4.1 shows a 3-D data cube for the dimensions A , B , and C , and an aggregate measure, M . A data cube is a lattice of cuboids. Each cuboid represents a group-by. ABC is the base cuboid, containing all three of the dimensions. Here, the aggregate measure, M , is computed for each possible combination of the three dimensions. The base cuboid is the least generalized of all of the cuboids in the data cube. The most generalized cuboid is the apex cuboid, commonly represented as *all*. It contains one value—it aggregates measure M for all of the tuples stored in the base cuboid. To drill down in the data cube, we move from the apex cuboid, downward in the lattice. To



re 4.1 Lattice of cuboids, making up a 3-D data cube with the dimensions A , B , and C for some aggregate measure, M .

roll up, we move from the base cuboid, upward. For the purposes of our discussion in this chapter, we will always use the term data cube to refer to a lattice of cuboids rather than an individual cuboid.

A cell in the base cuboid is a base cell. A cell from a nonbase cuboid is an aggregate cell. An aggregate cell aggregates over one or more dimensions, where each aggregated dimension is indicated by a "*" in the cell notation. Suppose we have an n -dimensional data cube. Let $a = (a_1, a_2, \dots, a_n, \text{measures})$ be a cell from one of the cuboids making up the data cube. We say that a is an m -dimensional cell (that is, from an m -dimensional cuboid) if exactly m ($m \leq n$) values among $\{a_1, a_2, \dots, a_n\}$ are not "*". If $m = n$, then a is a base cell; otherwise, it is an aggregate cell (i.e., where $m < n$).

Example 4.1 Base and aggregate cells. Consider a data cube with the dimensions *month*, *city*, and *customer_group*, and the measure *price*. $(Jan, *, *, 2800)$ and $(*, Toronto, *, 1200)$ are 1-D cells, $(Jan, *, Business, 150)$ is a 2-D cell, and $(Jan, Toronto, Business, 45)$ is a 3-D cell. Here, all base cells are 3-D, whereas 1-D and 2-D cells are aggregate cells. ■

An ancestor-descendant relationship may exist between cells. In an n -dimensional data cube, an i -D cell $a = (a_1, a_2, \dots, a_n, \text{measures}_a)$ is an ancestor of a j -D cell $b = (b_1, b_2, \dots, b_n, \text{measures}_b)$, and b is a descendant of a , if and only if (1) $i < j$, and (2) for $1 \leq m \leq n$, $a_m = b_m$ whenever $a_m \neq *$. In particular, cell a is called a parent of cell b , and b is a child of a , if and only if $j = i + 1$ and b is a descendant of a .

Example 4.2 Ancestor and descendant cells. Referring to our previous example, 1-D cell $a = (Jan, *, *, 2800)$, and 2-D cell $b = (Jan, *, Business, 150)$, are ancestors of 3-D cell $c = (Jan, Toronto, Business, 45)$; c is a descendant of both a and b ; b is a parent of c , and c is a child of b . ■

In order to ensure fast on-line analytical processing, it is sometimes desirable to precompute the full cube (i.e., all the cells of all of the cuboids for a given data cube). This, however, is exponential to the number of dimensions. That is, a data cube of n dimensions contains 2^n cuboids. There are even more cuboids if we consider concept hierarchies for each dimension.¹ In addition, the size of each cuboid depends on the cardinality of its dimensions. Thus, precomputation of the full cube can require huge and often excessive amounts of memory.

Nonetheless, full cube computation algorithms are important. Individual cuboids may be stored on secondary storage and accessed when necessary. Alternatively, we can use such algorithms to compute smaller cubes, consisting of a subset of the given set of dimensions, or a smaller range of possible values for some of the dimensions. In such cases, the smaller cube is a full cube for the given subset of dimensions and/or dimension values. A thorough understanding of full cube computation methods will

¹Equation (3.1) gives the total number of cuboids in a data cube where each dimension has an associated concept hierarchy.

help us develop efficient methods for computing partial cubes. Hence, it is important to explore scalable methods for computing all of the cuboids making up a data cube, that is, for full materialization. These methods must take into consideration the limited amount of main memory available for cuboid computation, the total size of the computed data cube, as well as the time required for such computation.

Partial materialization of data cubes offers an interesting trade-off between storage space and response time for OLAP. Instead of computing the full cube, we can compute only a subset of the data cube's cuboids, or subcubes consisting of subsets of cells from the various cuboids.

Many cells in a cuboid may actually be of little or no interest to the data analyst. Recall that each cell in a full cube records an aggregate value. Measures such as *count*, *sum*, or *sales.in.dollars* are commonly used. For many cells in a cuboid, the measure value will be zero. When the product of the cardinalities for the dimensions in a cuboid is large relative to the number of nonzero-valued tuples that are stored in the cuboid, then we say that the cuboid is *sparse*. If a cube contains many sparse cuboids, we say that the cube is *sparse*.

In many cases, a substantial amount of the cube's space could be taken up by a large number of cells with very low measure values. This is because the cube cells are often quite sparsely distributed within a multiple dimensional space. For example, a customer may only buy a few items in a store at a time. Such an event will generate only a few nonempty cells, leaving most other cube cells empty. In such situations, it is useful to materialize only those cells in a cuboid (group-by) whose measure value is above some minimum threshold. In a data cube for sales, say, we may wish to materialize only those cells for which $count \geq 10$ (i.e., where at least 10 tuples exist for the cell's given combination of dimensions), or only those cells representing $sales \geq \$100$. This not only saves processing time and disk space, but also leads to a more focused analysis. The cells that cannot pass the threshold are likely to be too trivial to warrant further analysis. Such partially materialized cubes are known as *iceberg cubes*. The minimum threshold is called the *minimum support threshold*, or *minimum support (min_sup)*, for short. By materializing only a fraction of the cells in a data cube, the result is seen as the "tip of the iceberg," where the "iceberg" is the potential full cube including all cells. An iceberg cube can be specified with an SQL query, as shown in the following example.

Example 4.3 Iceberg cube.

```
compute cube sales_iceberg as
select month, city, customer_group, count(*)
from salesInfo
cube by month, city, customer_group
having count(*) >= min_sup
```

The `compute cube` statement specifies the precomputation of the iceberg cube, *sales_iceberg*, with the dimensions *month*, *city*, and *customer_group*, and the aggregate measure `count()`. The input tuples are in the *salesInfo* relation. The `cube by` clause specifies that aggregates (group-by's) are to be formed for each of the possible subsets of the given

dimensions. If we were computing the full cube, each group-by would correspond to a cuboid in the data cube lattice. The constraint specified in the having clause is known as the iceberg condition. Here, the iceberg measure is *count*. Note that the iceberg cube computed for Example 4.3 could be used to answer group-by queries on any combination of the specified dimensions of the form *having count*(*) $\geq v$, where $v \geq \text{min_sup}$. Instead of *count*, the iceberg condition could specify more complex measures, such as *average*.

If we were to omit the having clause of our example, we would end up with the full cube. Let's call this cube *sales_cube*. The iceberg cube, *sales_iceberg*, excludes all the cells of *sales_cube* whose count is less than *min_sup*. Obviously, if we were to set the minimum support to 1 in *sales_iceberg*, the resulting cube would be the full cube, *sales_cube*. ■

A naïve approach to computing an iceberg cube would be to first compute the full cube and then prune the cells that do not satisfy the iceberg condition. However, this is still prohibitively expensive. An efficient approach is to compute only the iceberg cube directly without computing the full cube. Sections 4.1.3 and 4.1.4 discuss methods for efficient iceberg cube computation.

Introducing iceberg cubes will lessen the burden of computing trivial aggregate cells in a data cube. However, we could still end up with a large number of uninteresting cells to compute. For example, suppose that there are 2 base cells for a database of 100 dimensions, denoted as $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10\}$, where each has a cell count of 10. If the minimum support is set to 10, there will still be an impermissible number of cells to compute and store, although most of them are not interesting. For example, there are $2^{101} - 6$ distinct aggregate cells,² like $\{(a_1, a_2, a_3, a_4, \dots, a_{99}, *) : 10, \dots, (a_1, a_2, *, a_4, \dots, a_{99}, a_{100}) : 10, \dots, (a_1, a_2, a_3, *, \dots, *, *) : 10\}$, but most of them do not contain much new information. If we ignore all of the aggregate cells that can be obtained by replacing some constants by *'s while keeping the same measure value, there are only three distinct cells left: $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10, (a_1, a_2, *, \dots, *) : 20\}$. That is, out of $2^{101} - 6$ distinct aggregate cells, only 3 really offer new information.

To systematically compress a data cube, we need to introduce the concept of *closed coverage*. A cell, *c*, is a *closed cell* if there exists no cell, *d*, such that *d* is a specialization (descendant) of cell *c* (that is, where *d* is obtained by replacing a * in *c* with a non-* value), and *d* has the same measure value as *c*. A *closed cube* is a data cube consisting of only closed cells. For example, the three cells derived above are the three closed cells of the data cube for the data set: $\{(a_1, a_2, a_3, \dots, a_{100}) : 10, (a_1, a_2, b_3, \dots, b_{100}) : 10\}$. They form the lattice of a closed cube as shown in Figure 4.2. Other nonclosed cells can be derived from their corresponding closed cells in this lattice. For example, $(a_1, *, *, \dots, *) : 20$ can be derived from $(a_1, a_2, *, \dots, *) : 20$ because the former is a generalized nonclosed cell of the latter. Similarly, we have $(a_1, a_2, b_3, *, \dots, *) : 10$.

Another strategy for partial materialization is to precompute only the cuboids involving a small number of dimensions, such as 3 to 5. These cuboids form a cube

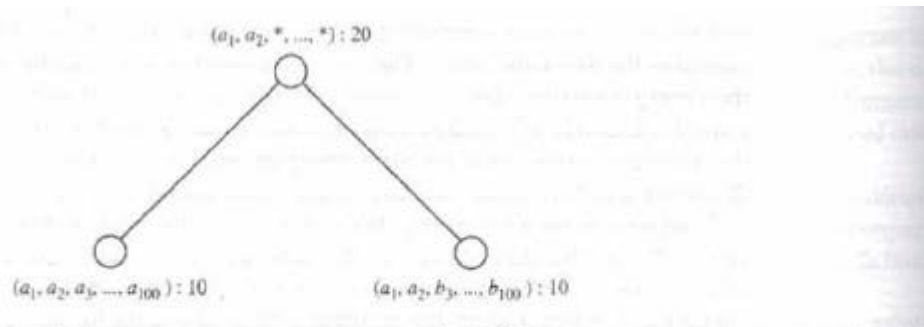


Figure 4.2 Three closed cells forming the lattice of a closed cube.

shell for the corresponding data cube. Queries on additional combinations of the dimensions will have to be computed on the fly. For example, we could compute all cuboids with 3 dimensions or less in an n -dimensional data cube, resulting in a cube shell of size 3. This, however, can still result in a large number of cuboids to compute, particularly when n is large. Alternatively, we can choose to precompute only portions or *fragments* of the cube shell, based on cuboids of interest. Section 4.1.5 discusses a method for computing such shell fragments and explores how they can be used for efficient OLAP query processing.

- b. What is attribute oriented induction? What is its use in data characterization, explain with examples? (3+5)**

Answer:

4.3 Attribute-Oriented Induction—An Alternative Method for Data Generalization and Concept Description

Data generalization summarizes data by replacing relatively low-level values (such as numeric values for an attribute *age*) with higher-level concepts (such as *young*, *middle-aged*, and *senior*). Given the large amount of data stored in databases, it is useful to be able to describe concepts in concise and succinct terms at generalized (rather than low) levels of abstraction. Allowing data sets to be generalized at multiple levels of abstraction facilitates users in examining the general behavior of the data. Given the *AllElectronics* database, for example, instead of examining individual customer transactions, sales managers may prefer to view the data generalized to higher levels, such as summarized by customer groups according to geographic regions, frequency of purchases per group, and customer income.

This leads us to the notion of *concept description*, which is a form of data generalization. A concept typically refers to a collection of data such as *frequent buyers*, *graduate students*, and so on. As a data mining task, concept description is not a simple enumeration of the data. Instead, concept description generates descriptions for the *characterization* and *comparison* of the data. It is sometimes called *class description*, when the concept to be described refers to a class of objects. Characterization provides a concise and succinct summarization of the given collection of data, while concept or class comparison (also known as *discrimination*) provides descriptions comparing two or more collections of data.

Up to this point, we have studied data cube (or OLAP) approaches to concept description using multidimensional, multilevel data generalization in data warehouses. “*Is data cube technology sufficient to accomplish all kinds of concept description tasks for large data sets?*” Consider the following cases.

- **Complex data types and aggregation:** Data warehouses and OLAP tools are based on a multidimensional data model that views data in the form of a data cube, consisting of dimensions (or attributes) and measures (aggregate functions). However, many current OLAP systems confine dimensions to nonnumeric data and measures to numeric data. In reality, the database can include attributes of various data types, including numeric, nonnumeric, spatial, text, or image, which ideally should be included in the concept description. Furthermore, the aggregation of attributes in a database may include sophisticated data types, such as the collection of nonnumeric data, the merging of spatial regions, the composition of images, the integration of texts,

and the grouping of object pointers. Therefore, OLAP, with its restrictions on the possible dimension and measure types, represents a simplified model for data analysis. Concept description should handle complex data types of the attributes and their aggregations, as necessary.

► **User-control versus automation:** On-line analytical processing in data warehouses is a user-controlled process. The selection of dimensions and the application of OLAP operations, such as drill-down, roll-up, slicing, and dicing, are primarily directed and controlled by the users. Although the control in most OLAP systems is quite user-friendly, users do require a good understanding of the role of each dimension. Furthermore, in order to find a satisfactory description of the data, users may need to specify a long sequence of OLAP operations. It is often desirable to have a more automated process that helps users determine which dimensions (or attributes) should be included in the analysis, and the degree to which the given data set should be generalized in order to produce an interesting summarization of the data.

This section presents an alternative method for concept description, called *attribute-oriented induction*, which works for complex types of data and relies on a data-driven generalization process.

4.3.1 Attribute-Oriented Induction for Data Characterization

The attribute-oriented induction (AOI) approach to concept description was first proposed in 1989, a few years before the introduction of the data cube approach. The data cube approach is essentially based on *materialized views* of the data, which typically have been precomputed in a data warehouse. In general, it performs off-line aggregation before an OLAP or data mining query is submitted for processing. On the other hand, the attribute-oriented induction approach is basically a *query-oriented*, generalization-based, on-line data analysis technique. Note that there is no inherent barrier distinguishing the two approaches based on on-line aggregation versus off-line precomputation. Some aggregations in the data cube can be computed on-line, while off-line precomputation of multidimensional space can speed up attribute-oriented induction as well.

The general idea of attribute-oriented induction is to first collect the task-relevant data using a database query and then perform generalization based on the examination of the number of distinct values of each attribute in the relevant set of data. The generalization is performed by either *attribute removal* or *attribute generalization*. Aggregation is performed by merging identical generalized tuples and accumulating their respective counts. This reduces the size of the generalized data set. The resulting generalized relation can be mapped into different forms for presentation to the user, such as charts or rules.

The following examples illustrate the process of attribute-oriented induction. We first discuss its use for characterization. The method is extended for the mining of class comparisons in Section 4.3.4.

Example 4.20 A data mining query for characterization. Suppose that a user would like to describe the general characteristics of graduate students in the *Big University* database, given the attributes *name*, *gender*, *major*, *birth_place*, *birth_date*, *residence*, *phone#* (telephone number), and *gpa* (grade point average). A data mining query for this characterization can be expressed in the data mining query language, DMQL, as follows:

```

use Big_University_DB
mine characteristics as "Science_Students"
in relevance to name, gender, major, birth_place, birth_date, residence,
    phone#, gpa
from student
where status in "graduate"

```

We will see how this example of a typical data mining query can apply attribute-oriented induction for mining characteristic descriptions.

First, **data focusing** should be performed *before* attribute-oriented induction. This step corresponds to the specification of the task-relevant data (i.e., data for analysis). The data are collected based on the information provided in the data mining query. Because a data mining query is usually relevant to only a portion of the database, selecting the relevant set of data not only makes mining more efficient, but also derives more meaningful results than mining the entire database.

Specifying the set of relevant attributes (i.e., attributes for mining, as indicated in DMQL with the *in relevance to* clause) may be difficult for the user. A user may select only a few attributes that he or she feels may be important, while missing others that could also play a role in the description. For example, suppose that the dimension *birth_place* is defined by the attributes *city*, *province_or_state*, and *country*. Of these attributes, let's say that the user has only thought to specify *city*. In order to allow generalization on the *birth_place* dimension, the other attributes defining this dimension should also be included. In other words, having the system automatically include *province_or_state* and *country* as relevant attributes allows *city* to be generalized to these higher conceptual levels during the induction process.

At the other extreme, suppose that the user may have introduced too many attributes by specifying all of the possible attributes with the clause "in relevance to *". In this case, all of the attributes in the relation specified by the *from* clause would be included in the analysis. Many of these attributes are unlikely to contribute to an interesting description. A correlation-based (Section 2.4.1) or entropy-based (Section 2.6.1) analysis method can be used to perform attribute *relevance analysis* and filter out statistically irrelevant or weakly relevant attributes from the descriptive mining process. Other approaches, such as attribute subset selection, are also described in Chapter 2.

"What does the 'where status in "graduate"' clause mean?" This *where* clause implies that a concept hierarchy exists for the attribute *status*. Such a concept hierarchy organizes primitive-level data values for *status*, such as "M.Sc.", "M.A.", "M.B.A.", "Ph.D.", "B.Sc.", "B.A.", into higher conceptual levels, such as "graduate" and "undergraduate." This use

Table 4.12 Initial working relation: a collection of task-relevant data.

name	gender	major	birth_place	birth_date	residence	phone#	gpa
Jim Woodman	M	CS	Vancouver, BC, Canada	8-12-76	3511 Main St., Richmond	687-4598	3.67
Scott Lachance	M	CS	Montreal, Que, Canada	28-7-75	345 1st Ave., Richmond	253-9106	3.70
Laura Lee	F	physics	Seattle, WA, USA	25-8-70	125 Austin Ave., Burnaby	420-5232	3.83
...

of concept hierarchies does not appear in traditional relational query languages, yet is likely to become a common feature in data mining query languages.

The data mining query presented above is transformed into the following relational query for the collection of the task-relevant set of data:

```
use Big_University_DB
select name, gender, major, birth_place, birth_date, residence, phone#, gpa
from student
where status in {"M.Sc.", "M.A.", "M.B.A.", "Ph.D."}
```

The transformed query is executed against the relational database, *Big_University_DB*, and returns the data shown in Table 4.12. This table is called the (task-relevant) **initial working relation**. It is the data on which induction will be performed. Note that each tuple is, in fact, a conjunction of attribute-value pairs. Hence, we can think of a tuple within a relation as a rule of conjuncts, and of induction on the relation as the generalization of these rules. ■

"Now that the data are ready for attribute-oriented induction, how is attribute-oriented induction performed?" The essential operation of attribute-oriented induction is *data generalization*, which can be performed in either of two ways on the initial working relation: *attribute removal* and *attribute generalization*.

Attribute removal is based on the following rule: *If there is a large set of distinct values for an attribute of the initial working relation, but either (1) there is no generalization operator on the attribute (e.g., there is no concept hierarchy defined for the attribute), or (2) its higher-level concepts are expressed in terms of other attributes, then the attribute should be removed from the working relation.*

Let's examine the reasoning behind this rule. An attribute-value pair represents a conjunct in a generalized tuple, or rule. The removal of a conjunct eliminates a constraint and thus generalizes the rule. If, as in case 1, there is a large set of distinct values for an attribute but there is no generalization operator for it, the attribute should be removed because it cannot be generalized, and preserving it would imply keeping a large number of disjuncts, which contradicts the goal of generating concise rules. On the other hand, consider case 2, where the higher-level concepts of the attribute are expressed in terms of other attributes. For example, suppose that the attribute in question is *street*, whose higher-level concepts are represented by the attributes $\langle city, province_or_state, country \rangle$.

The removal of *street* is equivalent to the application of a generalization operator. This rule corresponds to the generalization rule known as *dropping conditions* in the machine learning literature on *learning from examples*.

Attribute generalization is based on the following rule: *If there is a large set of distinct values for an attribute in the initial working relation, and there exists a set of generalization operators on the attribute, then a generalization operator should be selected and applied to the attribute.* This rule is based on the following reasoning. Use of a generalization operator to generalize an attribute value within a tuple, or rule, in the working relation will make the rule cover more of the original data tuples, thus generalizing the concept it represents. This corresponds to the generalization rule known as *climbing generalization trees* in *learning from examples*, or *concept tree ascension*.

Both rules, *attribute removal* and *attribute generalization*, claim that if there is a large set of distinct values for an attribute, further generalization should be applied. This raises the question: how large is "a large set of distinct values for an attribute" considered to be?

Depending on the attributes or application involved, a user may prefer some attributes to remain at a rather low abstraction level while others are generalized to higher levels. The control of how high an attribute should be generalized is typically quite subjective. The control of this process is called **attribute generalization control**. If the attribute is generalized "too high," it may lead to overgeneralization, and the resulting rules may not be very informative. On the other hand, if the attribute is not generalized to a "sufficiently high level," then undergeneralization may result, where the rules obtained may not be informative either. Thus, a balance should be attained in attribute-oriented generalization.

There are many possible ways to control a generalization process. We will describe two common approaches and then illustrate how they work with an example.

The first technique, called **attribute generalization threshold control**, either sets one generalization threshold for all of the attributes, or sets one threshold for each attribute. If the number of distinct values in an attribute is greater than the attribute threshold, further attribute removal or attribute generalization should be performed. Data mining systems typically have a default attribute threshold value generally ranging from 2 to 8 and should allow experts and users to modify the threshold values as well. If a user feels that the generalization reaches too high a level for a particular attribute, the threshold can be increased. This corresponds to drilling down along the attribute. Also, to further generalize a relation, the user can reduce the threshold of a particular attribute, which corresponds to rolling up along the attribute.

The second technique, called **generalized relation threshold control**, sets a threshold for the generalized relation. If the number of (distinct) tuples in the generalized relation is greater than the threshold, further generalization should be performed. Otherwise, no further generalization should be performed. Such a threshold may also be preset in the data mining system (usually within a range of 10 to 30), or set by an expert or user, and should be adjustable. For example, if a user feels that the generalized relation is too small, he or she can increase the threshold, which implies drilling down. Otherwise, to further generalize a relation, the threshold can be reduced, which implies rolling up.

These two techniques can be applied in sequence: first apply the attribute threshold control technique to generalize each attribute, and then apply relation threshold control to further reduce the size of the generalized relation. No matter which generalization control technique is applied, the user should be allowed to adjust the generalization thresholds in order to obtain interesting concept descriptions.

In many database-oriented induction processes, users are interested in obtaining quantitative or statistical information about the data at different levels of abstraction. Thus, it is important to accumulate count and other aggregate values in the induction process. Conceptually, this is performed as follows. The aggregate function, count, is associated with each database tuple. Its value for each tuple in the initial working relation is initialized to 1. Through attribute removal and attribute generalization, tuples within the initial working relation may be generalized, resulting in groups of *identical tuples*. In this case, all of the identical tuples forming a group should be merged into one tuple. The count of this new, generalized tuple is set to the total number of tuples from the initial working relation that are represented by (i.e., were merged into) the new generalized tuple. For example, suppose that by attribute-oriented induction, 52 data tuples from the initial working relation are all generalized to the same tuple, T . That is, the generalization of these 52 tuples resulted in 52 identical instances of tuple T . These 52 identical tuples are merged to form one instance of T , whose count is set to 52. Other popular aggregate functions that could also be associated with each tuple include sum and avg. For a given generalized tuple, sum contains the sum of the values of a given numeric attribute for the initial working relation tuples making up the generalized tuple. Suppose that tuple T contained $\text{sum}(\text{units_sold})$ as an aggregate function. The sum value for tuple T would then be set to the total number of units sold for each of the 52 tuples. The aggregate avg (average) is computed according to the formula, $\text{avg} = \text{sum}/\text{count}$.

Example 4.21 Attribute-oriented induction. Here we show how attribute-oriented induction is performed on the initial working relation of Table 4.12. For each attribute of the relation, the generalization proceeds as follows:

1. *name*: Since there are a large number of distinct values for *name* and there is no generalization operation defined on it, this attribute is removed.
2. *gender*: Since there are only two distinct values for *gender*, this attribute is retained and no generalization is performed on it.
3. *major*: Suppose that a concept hierarchy has been defined that allows the attribute *major* to be generalized to the values $\{\text{arts\&science}, \text{engineering}, \text{business}\}$. Suppose also that the attribute generalization threshold is set to 5, and that there are more than 20 distinct values for *major* in the initial working relation. By attribute generalization and attribute generalization control, *major* is therefore generalized by climbing the given concept hierarchy.
4. *birth_place*: This attribute has a large number of distinct values; therefore, we would like to generalize it. Suppose that a concept hierarchy exists for *birth_place*, defined

as “*city < province_or_state < country*”. If the number of distinct values for *country* in the initial working relation is greater than the attribute generalization threshold, then *birth_place* should be removed, because even though a generalization operator exists for it, the generalization threshold would not be satisfied. If instead, the number of distinct values for *country* is less than the attribute generalization threshold, then *birth_place* should be generalized to *birth_country*.

5. *birth_date*: Suppose that a hierarchy exists that can generalize *birth_date* to *age*, and *age* to *age_range*, and that the number of age ranges (or intervals) is small with respect to the attribute generalization threshold. Generalization of *birth_date* should therefore take place.
6. *residence*: Suppose that *residence* is defined by the attributes *number*, *street*, *residence_city*, *residence_province_or_state*, and *residence_country*. The number of distinct values for *number* and *street* will likely be very high, since these concepts are quite low level. The attributes *number* and *street* should therefore be removed, so that *residence* is then generalized to *residence_city*, which contains fewer distinct values.
7. *phonet#*: As with the attribute *name* above, this attribute contains too many distinct values and should therefore be removed in generalization.
8. *gpa*: Suppose that a concept hierarchy exists for *gpa* that groups values for grade point average into numerical intervals like {3.75–4.0, 3.5–3.75,...}, which in turn are grouped into descriptive values, such as {*excellent*, *very good*,...}. The attribute can therefore be generalized.

The generalization process will result in groups of identical tuples. For example, the first two tuples of Table 4.12 both generalize to the same identical tuple (namely, the first tuple shown in Table 4.13). Such identical tuples are then merged into one, with their counts accumulated. This process leads to the generalized relation shown in Table 4.13.

Based on the vocabulary used in OLAP, we may view count as a *measure*, and the remaining attributes as *dimensions*. Note that aggregate functions, such as sum, may be applied to numerical attributes, like *salary* and *sales*. These attributes are referred to as *measure attributes*.

Implementation techniques and methods of presenting the derived generalization are discussed in the following subsections.

Table 4.13 A generalized relation obtained by attribute-oriented induction on the data of Table 4.12.

<i>gender</i>	<i>major</i>	<i>birth_country</i>	<i>age_range</i>	<i>residence_city</i>	<i>gpa</i>	<i>count</i>
M	Science	Canada	20–25	Richmond	very good	16
F	Science	Foreign	25–30	Burnaby	excellent	22
...

Q.6 a. Explain Bayesian classification with suitable example. (8)

Answer: Bayesian classification is based on Bayes' Theorem. Bayesian classifiers are the statistical classifiers. Bayesian classifiers can predict class membership probabilities such as the probability that a given tuple belongs to a PARTICULAR class.

Baye's Theorem

Bayes' Theorem is NAMED after Thomas Bayes. There are two types of probabilities –

- Posterior Probability [P(H/X)]

- Prior Probability [P(H)]

where X is data tuple and H is some hypothesis.

According to Bayes' Theorem,

$$P(H/X) = P(X/H)P(H) / P(X)$$

Bayesian Belief Network

Bayesian Belief Networks specify joint CONDITIONAL probability distributions. They are also known as Belief Networks, Bayesian Networks, or Probabilistic Networks.

- A Belief Network allows CLASS conditional independencies to be defined between subsets of variables.
- It provides a graphical model of causal relationship on which learning can be performed.
- We can use a TRAINED Bayesian Network for classification.

There are two components that define a Bayesian Belief Network –

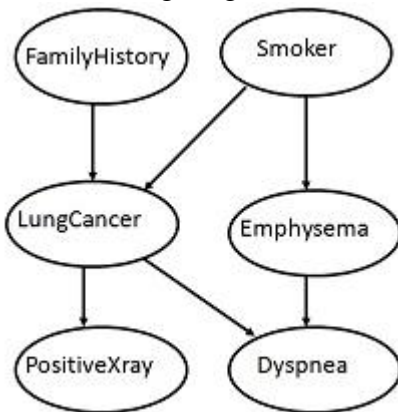
- Directed acyclic graph
- A set of conditional probability tables

Directed Acyclic Graph

- Each node in a directed acyclic graph represents a random variable.
- These variable may be discrete or CONTINUOUS valued.
- These variables may correspond to the actual attribute given in the data.

Directed Acyclic Graph Representation

The following diagram SHOWS a directed acyclic graph for six Boolean variables.



The arc in the diagram allows representation of causal knowledge. For EXAMPLE, lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. It is worth noting that the variable PositiveXray is independent of whether the patient has a family history of lung cancer or that the patient is a smoker, given that we know the patient has lung cancer.

Conditional Probability Table

The arc in the diagram allows representation of causal knowledge. For EXAMPLE, lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. It is worth noting that the variable PositiveXray is independent of whether the patient has a family history of lung cancer or that the patient is a smoker, given that we know the patient has lung cancer.

	FH,S	FH,-S	-FH,S	-FH,-S
LC	0.8	0.5	0.7	0.1
-LC	0.2	0.5	0.3	0.9

b. Give the difference between classification and prediction. (4)

Answer: Classification

- predicts categorical class labels (discrete or nominal)
- classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction
 - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
 - Credit approval
 - Target marketing
 - Medical diagnosis
 - Fraud detection

c. Discuss different types of association rules in data mining. (4)

Answer: An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item FOUND in the data. A consequent is an item that is found in combination with the antecedent.

Association rules are created by analyzing data for frequent if/then patterns and using the criteria *support* and *confidence* to identify the most IMPORTANT relationships. *Support* is an indication of how frequently the items APPEAR in the database. *Confidence* indicates the NUMBER of times the if/then statements have been found to be true.

In data mining, association rules are useful for analyzing and predicting customer behavior. They PLAY an important part in shopping basket data analysis, product clustering, catalog design and store layout.

Programmers use association rules to build PROGRAMS capable of machine learning. Machine learning is a TYPE of artificial intelligence (AI) that seeks to build programs with the ability to become more efficient without being explicitly programmed.

Q.7 a. What is back propagation? How is it used to classify patterns? (4+4)

Answer:

6.6 Classification by Backpropagation

“What is backpropagation?” Backpropagation is a neural network learning algorithm. The field of neural networks was originally kindled by psychologists and neurobiologists who sought to develop and test computational analogues of neurons. Roughly speaking, a neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. Neural network learning is also referred to as **connectionist learning** due to the connections between units.

Neural networks involve long training times and are therefore more suitable for applications where this is feasible. They require a number of parameters that are typically best determined empirically, such as the network topology or “structure.” Neural networks have been criticized for their poor interpretability. For example, it is difficult for humans to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network. These features initially made neural networks less desirable for data mining.

Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained. They can be used when you may have little knowledge of the relationships between attributes and classes. They are well-suited for continuous-valued inputs *and outputs*, unlike most decision tree algorithms. They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text. Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process. In addition, several techniques have recently been developed for the extraction of rules from trained neural networks. These factors contribute toward the usefulness of neural networks for classification and prediction in data mining.

There are many different kinds of neural networks and neural network algorithms. The most popular neural network algorithm is *backpropagation*, which gained reputation in the 1980s. In Section 6.6.1 you will learn about multilayer feed-forward networks, the type of neural network on which the backpropagation algorithm performs. Section 6.6.2 discusses defining a network topology. The backpropagation algorithm is described in Section 6.6.3. Rule extraction from trained neural networks is discussed in Section 6.6.4.

6.6.1 A Multilayer Feed-Forward Neural Network

The backpropagation algorithm performs learning on a *multilayer feed-forward* neural network. It iteratively learns a set of weights for prediction of the class label of tuples. A multilayer feed-forward neural network consists of an *input layer*, one or more *hidden layers*, and an *output layer*. An example of a multilayer feed-forward network is shown in Figure 6.15.

Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the input layer. These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of "neuronlike" units, known as a hidden layer. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers is arbitrary, although in practice, usually only one is used. The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction for given tuples.

The units in the input layer are called *input units*. The units in the hidden layers and output layer are sometimes referred to as *neurodes*, due to their symbolic biological basis, or as *output units*. The multilayer neural network shown in Figure 6.15 has two layers

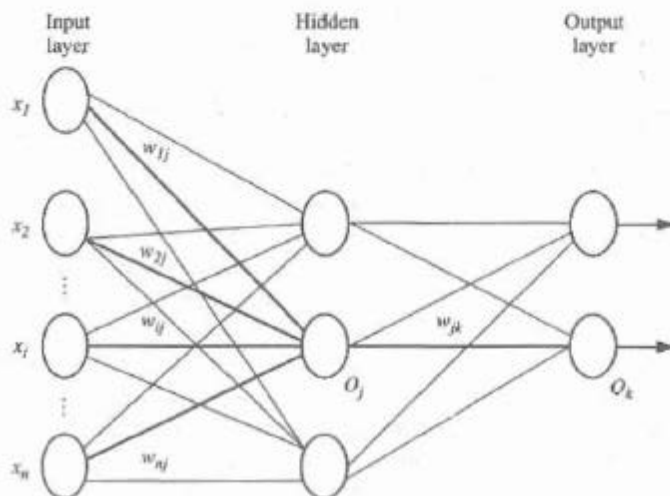


Figure 6.15 A multilayer feed-forward neural network.

of output units. Therefore, we say that it is a two-layer neural network. (The input layer is not counted because it serves only to pass the input values to the next layer.) Similarly, a network containing two hidden layers is called a *three-layer* neural network, and so on. The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer. It is fully connected in that each unit provides input to each unit in the next forward layer.

Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer (see Figure 6.17). It applies a nonlinear (activation) function to the weighted input. Multilayer feed-forward neural networks are able to model the class prediction as a nonlinear combination of the inputs. From a statistical point of view, they perform nonlinear regression. *Multilayer feed-forward networks, given enough hidden units and enough training samples, can closely approximate any function.*

6.6.2 Defining a Network Topology

“How can I design the topology of the neural network?” Before training can begin, the user must decide on the network topology by specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer.

Normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. Typically, input values are normalized so as to fall between 0.0 and 1.0. Discrete-valued attributes may be encoded such that there is one input unit per domain value. For example, if an attribute A has three possible or known values, namely $\{a_0, a_1, a_2\}$, then we may assign three input units to represent A . That is, we may have, say, I_0, I_1, I_2 as input units. Each unit is initialized to 0. If $A = a_0$, then I_0 is set to 1. If $A = a_1$, I_1 is set to 1, and so on. Neural networks can be used for both classification (to predict the class label of a given tuple) or prediction (to predict a continuous-valued output). For classification, one output unit may be used to represent two classes (where the value 1 represents one class, and the value 0 represents the other). If there are more than two classes, then one output unit per class is used.

There are no clear rules as to the “best” number of hidden layer units. Network design is a trial-and-error process and may affect the accuracy of the resulting trained network. The initial values of the weights may also affect the resulting accuracy. Once a network has been trained and its accuracy is not considered acceptable, it is common to repeat the training process with a different network topology or a different set of initial weights. Cross-validation techniques for accuracy estimation (described in Section 6.13) can be used to help decide when an acceptable network has been found. A number of automated techniques have been proposed that search for a “good” network structure. These typically use a hill-climbing approach that starts with an initial structure that is selectively modified.

6.6.3 Backpropagation

“How does backpropagation work?” Backpropagation learns by iteratively processing a data set of training tuples, comparing the network’s prediction for each tuple with the

actual known *target* value. The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for prediction). For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name *backpropagation*). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops. The algorithm is summarized in Figure 6.16. The steps involved are expressed in terms of inputs, outputs, and errors, and may seem awkward if this is your first look at neural network learning. However, once you become familiar with the process, you will see that each step is inherently simple. The steps are described below.

Algorithm: Backpropagation. Neural network learning for classification or prediction, using the backpropagation algorithm.

Input:

- D , a data set consisting of the training tuples and their associated target values;
- l , the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

```

(1) Initialize all weights and biases in network;
(2) while terminating condition is not satisfied {
(3)   for each training tuple  $X$  in  $D$  {
(4)     // Propagate the inputs forward:
(5)     for each input layer unit  $j$  {
(6)        $O_j = I_j$ ; // output of an input unit is its actual input value
(7)     for each hidden or output layer unit  $j$  {
(8)        $I_j = \sum_i w_{ij} O_i + \theta_j$ ; // compute the net input of unit  $j$  with respect to the
           previous layer,  $i$ 
(9)        $O_j = \frac{1}{1+e^{-I_j}}$ ; // compute the output of each unit  $j$ 
(10)    // Backpropagate the errors:
(11)    for each unit  $j$  in the output layer
(12)       $Err_j = O_j(1 - O_j)(T_j - O_j)$ ; // compute the error
(13)    for each unit  $j$  in the hidden layers, from the last to the first hidden layer
(14)       $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$ ; // compute the error with respect to the
           next higher layer,  $k$ 
(15)    for each weight  $w_{ij}$  in network {
(16)       $\Delta w_{ij} = (l) Err_j O_i$ ; // weight increment
(17)       $w_{ij} = w_{ij} + \Delta w_{ij}$ ; // weight update
(18)    for each bias  $\theta_j$  in network {
(19)       $\Delta \theta_j = (l) Err_j$ ; // bias increment
(20)       $\theta_j = \theta_j + \Delta \theta_j$ ; // bias update
(21)    } }

```

Figure 6.16 Backpropagation algorithm.

Initialize the weights: The weights in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0 , or -0.5 to 0.5). Each unit has a *bias* associated with it, as explained below. The biases are similarly initialized to small random numbers.

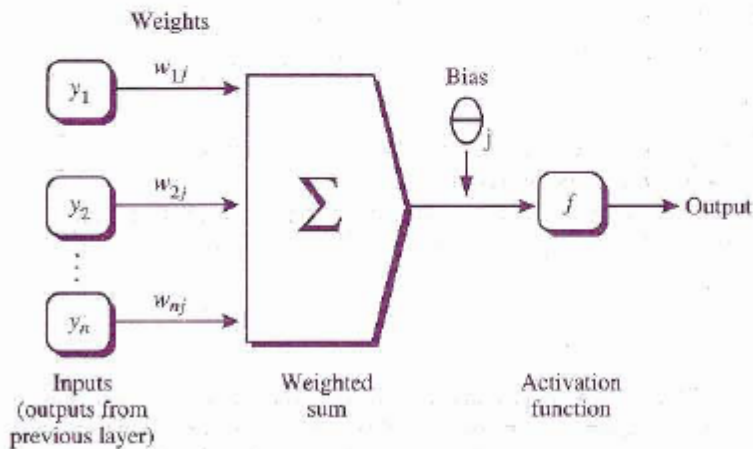
Each training tuple, X , is processed by the following steps.

Propagate the inputs forward: First, the training tuple is fed to the input layer of the network. The inputs pass through the input units, unchanged. That is, for an input unit, j , its output, O_j , is equal to its input value, I_j . Next, the net input and output of each unit in the hidden and output layers are computed. The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs. To help illustrate this point, a hidden layer or output layer unit is shown in Figure 6.17. Each such unit has a number of inputs to it that are, in fact, the outputs of the units connected to it in the previous layer. Each connection has a weight. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit j in a hidden or output layer, the net input, I_j , to unit j is

$$I_j = \sum_i w_{ij} O_i + \theta_j \quad (6.24)$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j ; O_i is the output of unit i from the previous layer; and θ_j is the bias of the unit. The bias acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input and then applies an **activation** function to it, as illustrated in Figure 6.17. The function symbolizes the activation



- 6.17** A hidden or output layer unit j : The inputs to unit j are outputs from the previous layer. These are multiplied by their corresponding weights in order to form a weighted sum, which is added to the bias associated with unit j . A nonlinear activation function is applied to the net input. (For ease of explanation, the inputs to unit j are labeled y_1, y_2, \dots, y_n . If unit j were in the first hidden layer, then these inputs would correspond to the input tuple (x_1, x_2, \dots, x_n) .)

of the neuron represented by the unit. The logistic, or sigmoid, function is used. Given the net input I_j to unit j , then O_j , the output of unit j , is computed as

$$O_j = \frac{1}{1 + e^{-I_j}} \quad (6.25)$$

This function is also referred to as a *squashing function*, because it maps a large input domain onto the smaller range of 0 to 1. The logistic function is nonlinear and differentiable, allowing the backpropagation algorithm to model classification problems that are linearly inseparable.

We compute the output values, O_j , for each hidden layer, up to and including the output layer, which gives the network's prediction. In practice, it is a good idea to cache (i.e., save) the intermediate output values at each unit as they are required again later, when backpropagating the error. This trick can substantially reduce the amount of computation required.

Backpropagate the error: The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Err_j is computed by

$$Err_j = O_j(1 - O_j)(T_j - O_j), \quad (6.26)$$

where O_j is the actual output of unit j , and T_j is the known target value of the given training tuple. Note that $O_j(1 - O_j)$ is the derivative of the logistic function.

To compute the error of a hidden layer unit j , the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}, \quad (6.27)$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k .

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where Δw_{ij} is the change in weight w_{ij} :

$$\Delta w_{ij} = (l) Err_j O_i \quad (6.28)$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \quad (6.29)$$

"What is the ' l ' in Equation (6.28)?" The variable l is the **learning rate**, a constant typically having a value between 0.0 and 1.0. Backpropagation learns using a method of gradient descent to search for a set of weights that fits the training data so as to minimize the mean squared distance between the network's class prediction and the known target value of the tuples.⁸ The learning rate helps avoid getting stuck at a local minimum

⁸ A method of gradient descent was also used for training Bayesian belief networks, as described in Section 6.4.4.

in decision space (i.e., where the weights appear to converge, but are not the optimum solution) and encourages finding the global minimum. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to $1/t$, where t is the number of iterations through the training set so far.

Biases are updated by the following equations below, where $\Delta\theta_j$ is the change in bias θ_j :

$$\Delta\theta_j = (l)Err_j \quad (6.30)$$

$$\theta_j = \theta_j + \Delta\theta_j \quad (6.31)$$

Note that here we are updating the weights and biases after the presentation of each tuple. This is referred to as *case updating*. Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all of the tuples in the training set have been presented. This latter strategy is called *epoch updating*, where one iteration through the training set is an *epoch*. In theory, the mathematical derivation of backpropagation employs epoch updating, yet in practice, case updating is more common because it tends to yield more accurate results.

Terminating condition: Training stops when

- All Δw_{ij} in the previous epoch were so small as to be below some specified threshold, or
- The percentage of tuples misclassified in the previous epoch is below some threshold, or
- A prespecified number of epochs has expired.

In practice, several hundreds of thousands of epochs may be required before the weights will converge.

"How efficient is backpropagation?" The computational efficiency depends on the time spent training the network. Given $|D|$ tuples and w weights, each epoch requires $O(|D| \times w)$ time. However, in the worst-case scenario, the number of epochs can be exponential in n , the number of inputs. In practice, the time required for the networks to converge is highly variable. A number of techniques exist that help speed up the training time. For example, a technique known as *simulated annealing* can be used, which also ensures convergence to a global optimum.

- b. Explain linear and non-linear methods of prediction. In which case will you use each, explain with example. (4+4)**

Answer:

6.1.1 Prediction

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

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

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

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

6.11.1 Linear Regression

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

$$y = b + wx, \quad (6.48)$$

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

$$y = w_0 + w_1x. \quad (6.49)$$

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

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

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

$$w_0 = \bar{y} - w_1 \bar{x} \quad (6.51)$$

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

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

Table 6.7 Salary data.

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

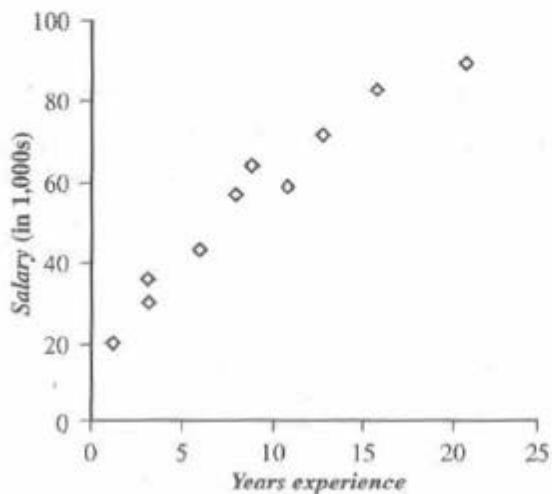


Figure 6.26 Plot of the data in Table 6.7 for Example 6.11. Although the points do not fall on a straight line, the overall pattern suggests a linear relationship between x (years experience) and y (salary).

corresponding salary of the graduate. The 2-D data can be graphed on a *scatter plot*, as in Figure 6.26. The plot suggests a linear relationship between the two variables, x and y . We model the relationship that salary may be related to the number of years of work experience with the equation $y = w_0 + w_1x$.

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

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

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

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

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

$$y = w_0 + w_1x_1 + w_2x_2, \quad (6.52)$$

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

6.11.2 Nonlinear Regression

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

Q.8 a. Describe the working of Partitioning Around Medoids (PAM) algorithm. (8)

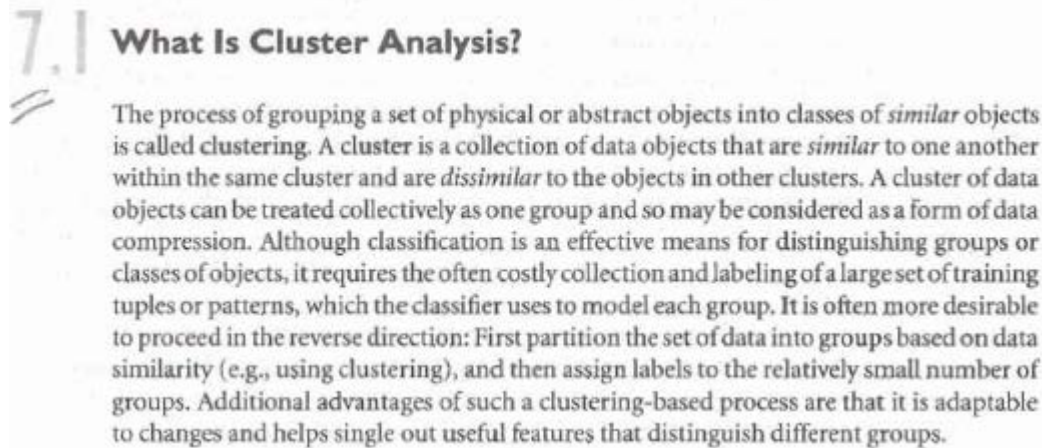
Answer: The most common realisation of k -medoid clustering is the **Partitioning Around Medoids (PAM)** algorithm and is as follows:^[2]

1. Initialize: randomly select (without replacement) k of the n data points as the medoids
2. Associate each data point to the closest medoid. ("closest" here is defined using any valid distance metric, most commonly Euclidean distance, Manhattan distance or Minkowski distance)
3. For each medoid m

1. For each non-medoid data point o
 1. Swap m and o and compute the total cost of the configuration
4. Select the configuration with the lowest cost.
5. Repeat steps 2 to 4 until there is no change in the medoid.

b. What is cluster analysis, how is it different from classification? Discuss the types of data that occur in cluster analysis. (3+5)

Answer:



7.1 What Is Cluster Analysis?

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

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

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

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

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

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

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

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

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

7.2 Types of Data in Cluster Analysis

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

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

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

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

$$\begin{bmatrix} 0 & & & & & \\ d(2,1) & 0 & & & & \\ d(3,1) & d(3,2) & 0 & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 & \end{bmatrix} \quad (7.2)$$

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

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

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

7.2.1 Interval-Scaled Variables

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

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

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

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

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

1. Calculate the mean absolute deviation, s_f :

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

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

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

$$z_{if} = \frac{x_{if} - m_f}{s_f}. \quad (7.4)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2.2 Binary Variables

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

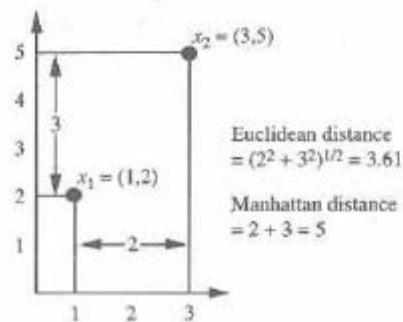


Figure 7.1 Euclidean and Manhattan distances between two objects.

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

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

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

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

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

Table 7.1 A contingency table for binary variables.

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

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

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

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

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

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

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

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

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

$$d(\text{Jack}, \text{Mary}) = \frac{0+1}{2+0+1} = 0.33$$

$$d(\text{Jack}, \text{Jim}) = \frac{1+1}{1+1+1} = 0.67$$

$$d(\text{Mary}, \text{Jim}) = \frac{1+2}{1+1+2} = 0.75$$

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

<i>name</i>	<i>gender</i>	<i>fever</i>	<i>cough</i>	<i>test-1</i>	<i>test-2</i>	<i>test-3</i>	<i>test-4</i>
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	Y	N	N	N	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

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

7.2.3 Categorical, Ordinal, and Ratio-Scaled Variables

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

Categorical Variables

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

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

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

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

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

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

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

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

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

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

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

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

Ordinal Variables

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

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

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

1. The value of f for the i th object is x_{if} , and f has M_f ordered states, representing the ranking $1, \dots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \dots, M_f\}$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto $[0.0, 1.0]$ so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i th object in the f th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \quad (7.13)$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 7.2.1 for interval-scaled variables, using z_{if} to represent the f value for the i th object.

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

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

Ratio-Scaled Variables

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

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

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

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

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

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

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

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

7.2.4 Variables of Mixed Types

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

"So, how can we compute the dissimilarity between objects of mixed variable types?"

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

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

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

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

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

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

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

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

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

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

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

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

7.2.5 Vector Objects

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

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

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

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

Q.9 a. Describe various applications of data mining. (8)

Answer: Data mining is widely used in diverse areas. There are a number of commercial data mining system available today and yet there are many challenges in this field.

Data Mining Applications

Here is the list of areas where data mining is widely used –

- Financial Data Analysis
- Retail Industry
- Telecommunication Industry
- Biological Data Analysis
- Other Scientific Applications
- Intrusion Detection

Financial Data Analysis

The financial data in banking and financial industry is generally reliable and of high quality which facilitates systematic data analysis and data mining. Some of the typical cases are as follows –

- Design and construction of data warehouses for multidimensional data analysis and data mining.
- Loan payment prediction and customer credit policy analysis.
- Classification and clustering of customers for targeted marketing.
- Detection of money laundering and other financial crimes.

Retail Industry

Data Mining has its great application in Retail Industry because it collects large amount of data from on sales, customer purchasing history, goods transportation, consumption and services. It is natural that the quantity of data collected will continue to expand rapidly because of the increasing ease, availability and popularity of the web.

Data mining in retail industry helps in identifying customer buying patterns and trends that lead to improved quality of customer service and good customer retention and satisfaction. Here is the list of examples of data mining in the retail industry –

- Design and Construction of data warehouses based on the benefits of data mining.
- Multidimensional analysis of sales, customers, products, time and region.
- Analysis of effectiveness of sales campaigns.
- Customer Retention.
- Product recommendation and cross-referencing of items.

Telecommunication Industry

Today the telecommunication industry is one of the most emerging industries providing various services such as fax, pager, cellular phone, internet messenger, images, e-mail, web data transmission, etc. Due to the development of new computer and communication technologies, the telecommunication industry is rapidly expanding. This is the reason why data mining is become very important to help and understand the business.

Data mining in telecommunication industry helps in identifying the telecommunication patterns, catch fraudulent activities, make better use of resource, and improve quality of service. Here is the list of examples for which data mining improves telecommunication services –

- Multidimensional Analysis of Telecommunication data.
- Fraudulent pattern analysis.
- Identification of unusual patterns.
- Multidimensional association and sequential patterns analysis.
- Mobile Telecommunication services.
- Use of visualization tools in telecommunication data analysis.

Biological Data Analysis

In recent times, we have seen a tremendous growth in the field of biology such as genomics, proteomics, functional Genomics and biomedical research. Biological data mining is a very important part of Bioinformatics. Following are the aspects in which data mining contributes for biological data analysis –

- Semantic integration of heterogeneous, distributed genomic and proteomic databases.
- Alignment, indexing, similarity search and comparative analysis multiple nucleotide sequences.
- Discovery of structural patterns and analysis of genetic networks and protein pathways.
- Association and path analysis.
- Visualization tools in genetic data analysis.

Other Scientific Applications

The applications discussed above tend to handle relatively small and homogeneous data sets for which the statistical techniques are appropriate. Huge amount of data have been collected from scientific domains such as geosciences, astronomy, etc. A large amount of data sets is being generated because of the fast numerical simulations in various fields such as climate and ecosystem modeling, chemical engineering, fluid dynamics, etc. Following are the applications of data mining in the field of Scientific Applications –

- Data Warehouses and data preprocessing.
- Graph-based mining.
- Visualization and domain specific knowledge.

Intrusion Detection

Intrusion refers to any kind of action that threatens integrity, confidentiality, or the availability of network resources. In this world of connectivity, security has become the major issue. With increased usage of internet and availability of the tools and tricks for intruding and attacking network prompted intrusion detection to become a critical component of network administration. Here is the list of areas in which data mining technology may be applied for intrusion detection –

- Development of data mining algorithm for intrusion detection.
- Association and correlation analysis, aggregation to help select and build discriminating attributes.
- Analysis of Stream data.
- Distributed data mining.
- Visualization and query tools.

b. What is web mining? Explain the techniques in web mining. (8)

Answer: In customer relationship management (CRM), Web mining is the integration of information gathered by traditional data mining methodologies and techniques with information gathered over the World Wide Web. (*Mining* means extracting something useful or valuable from a baser substance, such as mining gold from the earth.) Web mining is used to understand customer behavior, evaluate the effectiveness of a particular Web site, and help quantify the success of a marketing campaign.

Web mining allows you to look for patterns in data through content mining, structure mining, and usage mining. Content mining is used to examine data collected by search engines and Web spiders. Structure mining is used to examine data related to the structure of a particular Web site and usage mining is used to examine data related to a particular user's browser as well as data gathered by forms the user may have submitted during Web transactions.

The information gathered through Web mining is evaluated (sometimes with the aid of software graphing applications) by using traditional data mining parameters such as clustering and classification, association, and examination of sequential patterns.

TEXTBOOK

- I. **Data Mining, Concepts and Techniques, Jiawei Han and Micheline Kamber, Elsevier, Second Edition, 2006**