chapter 28

Data Mining Concepts

Over the last three decades, many organizations have generated a large amount of machine-readable data in the form of files and databases. To process this data, we have the database technology available that supports query languages like SQL. The problem with SQL is that it is a structured language that assumes the user is aware of the database schema. SQL supports operations of relational algebra that allow a user to select rows and columns of data from tables or join-related information from tables based on common fields. In the next chapter, we will see that *data warehousing technology* affords several types of functionality: that of consolidation, aggregation, and summarization of data. Data warehouses let us view the same information along multiple dimensions. In this chapter, we will focus our attention on another very popular area of interest known as data mining. As the term connotes, **data mining** refers to the mining or discovery of new information in terms of patterns or rules from vast amounts of data. To be practically useful, data mining must be carried out efficiently on large files and databases. Although some data mining features are being provided in RDBMSs, data mining is *not* well-integrated with database management systems.

We will briefly review the state of the art of this rather extensive field of data mining, which uses techniques from such areas as machine learning, statistics, neural networks, and genetic algorithms. We will highlight the nature of the information that is discovered, the types of problems faced when trying to mine databases, and the types of applications of data mining. We will also survey the state of the art of a large number of commercial tools available (see Section 28.7) and describe a number of research advances that are needed to make this area viable.

28.1 Overview of Data Mining Technology

In reports such as the very popular Gartner Report,[1] data mining has been hailed as one of the top technologies for the near future. In this section we relate data mining to the broader area called *knowledge discovery* and contrast the two by means of an illustrative example.

28.1.1 Data Mining versus Data Warehousing

The goal of a data warehouse (see Chapter 29) is to support decision making with data. Data mining can be used in conjunction with a data warehouse to help with certain types of decisions. Data mining can be applied to operational databases with individual transactions. To make data mining more efficient, the data warehouse should have an aggregated or summarized collection of data. Data mining helps in extracting meaningful new patterns that cannot necessarily be found by merely querying or processing data or metadata in the data warehouse. Therefore, data mining applications should be strongly considered early, during the design of a data warehouse. Also, data mining tools should be designed to facilitate their use in conjunction with data warehouses. In fact, for very large databases running into terabytes and even petabytes of data, successful use of data mining applications will depend first on the construction of a data warehouse.

28.1.2 Data Mining as a Part of the Knowledge Discovery Process

Knowledge Discovery in Databases, frequently abbreviated as **KDD**, typically encompasses more than data mining. The knowledge discovery process comprises six phases:[2] data selection, data cleansing, enrichment, data transformation or encoding, data mining, and the reporting and display of the discovered information.

As an example, consider a transaction database maintained by a specialty consumer goods retailer. Suppose the client data includes a customer name, ZIP Code, phone number, date of purchase, item code, price, quantity, and total amount. A variety of new knowledge can be discovered by KDD processing on this client database. During *data selection,* data about specific items or categories of items, or from stores in a specific region or area of the country, may be selected. The *data cleansing* process then may correct invalid ZIP Codes or eliminate records with incorrect phone prefixes. *Enrichment* typically enhances the data with additional sources of information. For example, given the client names and phone numbers, the store may purchase other data about age, income, and credit rating and append them to each record. *Data transformation* and encoding may be done to reduce the amount

[1]The Gartner Report is one example of the many technology survey publications that corporate managers rely on to make their technology selection discussions.

[2]This discussion is largely based on Adriaans and Zantinge (1996).

of data. For instance, item codes may be grouped in terms of product categories into audio, video, supplies, electronic gadgets, camera, accessories, and so on. ZIP Codes may be aggregated into geographic regions, incomes may be divided into ranges, and so on. In Figure 29.1, we will show a step called *cleaning* as a precursor to the data warehouse creation. If data mining is based on an existing warehouse for this retail store chain, we would expect that the cleaning has already been applied. It is only after such preprocessing that *data mining* techniques are used to mine different rules and patterns.

The result of mining may be to discover the following type of *new* information:

- **Association rules**—for example, whenever a customer buys video equipment, he or she also buys another electronic gadget.
- **Sequential patterns**—for example, suppose a customer buys a camera, and within three months he or she buys photographic supplies, then within six months he is likely to buy an accessory item. This defines a sequential pattern of transactions. A customer who buys more than twice in lean periods may be likely to buy at least once during the Christmas period.
- **Classification trees**—for example, customers may be classified by frequency of visits, types of financing used, amount of purchase, or affinity for types of items; some revealing statistics may be generated for such classes.

We can see that many possibilities exist for discovering new knowledge about buying patterns, relating factors such as age, income group, place of residence, to what and how much the customers purchase. This information can then be utilized to plan additional store locations based on demographics, run store promotions, combine items in advertisements, or plan seasonal marketing strategies. As this retail store example shows, data mining must be preceded by significant data preparation before it can yield useful information that can directly influence business decisions.

The results of data mining may be reported in a variety of formats, such as listings, graphic outputs, summary tables, or visualizations.

28.1.3 Goals of Data Mining and Knowledge Discovery

Data mining is typically carried out with some end goals or applications. Broadly speaking, these goals fall into the following classes: prediction, identification, classification, and optimization.

- **Prediction.** Data mining can show how certain attributes within the data will behave in the future. Examples of predictive data mining include the analysis of buying transactions to predict what consumers will buy under certain discounts, how much sales volume a store will generate in a given period, and whether deleting a product line will yield more profits. In such applications, business logic is used coupled with data mining. In a scientific context, certain seismic wave patterns may predict an earthquake with high probability.

- **Identification.** Data patterns can be used to identify the existence of an item, an event, or an activity. For example, intruders trying to break a system may be identified by the programs executed, files accessed, and CPU time per session. In biological applications, existence of a gene may be identified by certain sequences of nucleotide symbols in the DNA sequence. The area known as *authentication* is a form of identification. It ascertains whether a user is indeed a specific user or one from an authorized class, and involves a comparison of parameters or images or signals against a database.
- **Classification.** Data mining can partition the data so that different classes or categories can be identified based on combinations of parameters. For example, customers in a supermarket can be categorized into discount-seeking shoppers, shoppers in a rush, loyal regular shoppers, shoppers attached to name brands, and infrequent shoppers. This classification may be used in different analyses of customer buying transactions as a post-mining activity. Sometimes classification based on common domain knowledge is used as an input to decompose the mining problem and make it simpler. For instance, health foods, party foods, or school lunch foods are distinct categories in the supermarket business. It makes sense to analyze relationships within and across categories as separate problems. Such categorization may be used to encode the data appropriately before subjecting it to further data mining.
- **Optimization.** One eventual goal of data mining may be to optimize the use of limited resources such as time, space, money, or materials and to maximize output variables such as sales or profits under a given set of constraints. As such, this goal of data mining resembles the objective function used in operations research problems that deals with optimization under constraints.

The term data mining is popularly used in a very broad sense. In some situations it includes statistical analysis and constrained optimization as well as machine learning. There is no sharp line separating data mining from these disciplines. It is beyond our scope, therefore, to discuss in detail the entire range of applications that make up this vast body of work. For a detailed understanding of the topic, readers are referred to specialized books devoted to data mining.

28.1.4 Types of Knowledge Discovered during Data Mining

The term *knowledge* is broadly interpreted as involving some degree of intelligence. There is a progression from raw data to information to knowledge as we go through additional processing. Knowledge is often classified as inductive versus deductive. **Deductive knowledge** deduces new information based on applying *prespecified* logical rules of deduction on the given data. Data mining addresses **inductive knowledge**, which discovers new rules and patterns from the supplied data. Knowledge can be represented in many forms: In an unstructured sense, it can be represented by rules or propositional logic. In a structured form, it may be represented in deci-

sion trees, semantic networks, neural networks, or hierarchies of classes or frames. It is common to describe the knowledge discovered during data mining as follows:

- **Association rules.** These rules correlate the presence of a set of items with another range of values for another set of variables. Examples: (1) When a female retail shopper buys a handbag, she is likely to buy shoes. (2) An X-ray image containing characteristics a and b is likely to also exhibit characteristic c.
- **Classification hierarchies.** The goal is to work from an existing set of events or transactions to create a hierarchy of classes. Examples: (1) A population may be divided into five ranges of credit worthiness based on a history of previous credit transactions. (2) A model may be developed for the factors that determine the desirability of a store location on a 1–10 scale. (3) Mutual funds may be classified based on performance data using characteristics such as growth, income, and stability.
- **Sequential patterns.** A sequence of actions or events is sought. Example: If a patient underwent cardiac bypass surgery for blocked arteries and an aneurysm and later developed high blood urea within a year of surgery, he or she is likely to suffer from kidney failure within the next 18 months. Detection of sequential patterns is equivalent to detecting associations among events with certain temporal relationships.
- **Patterns within time series.** Similarities can be detected within positions of a **time series** of data, which is a sequence of data taken at regular intervals, such as daily sales or daily closing stock prices. Examples: (1) Stocks of a utility company, ABC Power, and a financial company, XYZ Securities, showed the same pattern during 2009 in terms of closing stock prices. (2) Two products show the same selling pattern in summer but a different one in winter. (3) A pattern in solar magnetic wind may be used to predict changes in Earth's atmospheric conditions.
- **Clustering.** A given population of events or items can be partitioned (segmented) into sets of "similar" elements. Examples: (1) An entire population of treatment data on a disease may be divided into groups based on the similarity of side effects produced. (2) The adult population in the United States may be categorized into five groups from *most likely to buy* to *least likely to buy* a new product. (3) The Web accesses made by a collection of users against a set of documents (say, in a digital library) may be analyzed in terms of the keywords of documents to reveal clusters or categories of users.

For most applications, the desired knowledge is a combination of the above types. We expand on each of the above knowledge types in the following sections.

28.2 Association Rules

28.2.1 Market-Basket Model, Support, and Confidence

One of the major technologies in data mining involves the discovery of association rules. The database is regarded as a collection of transactions, each involving a set of

items. A common example is that of **market-basket data**. Here the market basket corresponds to the sets of items a consumer buys in a supermarket during one visit. Consider four such transactions in a random sample shown in Figure 28.1.

An **association rule** is of the form $X => Y$, where $X = \{x_1, x_2, ..., x_n\}$, and $Y = \{y_1, y_2, ..., y_m\}$ are sets of items, with x_i and y_j being distinct items for all i and all j. This association states that if a customer buys X, he or she is also likely to buy Y. In general, any association rule has the form LHS (left-hand side) => RHS (right-hand side), where LHS and RHS are sets of items. The set LHS $\cup$ RHS is called an **itemset**, the set of items purchased by customers. For an association rule to be of interest to a data miner, the rule should satisfy some interest measure. Two common interest measures are support and confidence.

The **support** for a rule LHS => RHS is with respect to the itemset; it refers to how frequently a specific itemset occurs in the database. That is, the support is the percentage of transactions that contain all of the items in the itemset LHS $\cup$ RHS. If the support is low, it implies that there is no overwhelming evidence that items in LHS $\cup$ RHS occur together because the itemset occurs in only a small fraction of transactions. Another term for support is *prevalence* of the rule.

The **confidence** is with regard to the implication shown in the rule. The confidence of the rule LHS => RHS is computed as the support(LHS $\cup$ RHS)/support(LHS). We can think of it as the probability that the items in RHS will be purchased given that the items in LHS are purchased by a customer. Another term for confidence is *strength* of the rule.

As an example of support and confidence, consider the following two rules: milk => juice and bread => juice. Looking at our four sample transactions in Figure 28.1, we see that the support of {milk, juice} is 50 percent and the support of {bread, juice} is only 25 percent. The confidence of milk => juice is 66.7 percent (meaning that, of three transactions in which milk occurs, two contain juice) and the confidence of bread => juice is 50 percent (meaning that one of two transactions containing bread also contains juice).

As we can see, support and confidence do not necessarily go hand in hand. The goal of mining association rules, then, is to generate all possible rules that exceed some minimum user-specified support and confidence thresholds. The problem is thus decomposed into two subproblems:

1. Generate all itemsets that have a support that exceeds the threshold. These sets of items are called **large** (or **frequent**) **itemsets**. Note that large here means large support.

Transaction_id	Time	Items_bought
101	6:35	milk, bread, cookies, juice
792	7:38	milk, juice
1130	8:05	milk, eggs
1735	8:40	bread, cookies, coffee

Figure 28.1
Sample transactions in market-basket model.

2. For each large itemset, all the rules that have a minimum confidence are generated as follows: For a large itemset X and $Y \subset X$, let $Z = X - Y$; then if support(X)/support(Z) > minimum confidence, the rule $Z \Rightarrow Y$ (that is, $X - Y \Rightarrow Y$) is a valid rule.

Generating rules by using all large itemsets and their supports is relatively straightforward. However, discovering all large itemsets together with the value for their support is a major problem if the cardinality of the set of items is very high. A typical supermarket has thousands of items. The number of distinct itemsets is 2^m, where m is the number of items, and counting support for all possible itemsets becomes very computation intensive. To reduce the combinatorial search space, algorithms for finding association rules utilize the following properties:

- A subset of a large itemset must also be large (that is, each subset of a large itemset exceeds the minimum required support).
- Conversely, a superset of a small itemset is also small (implying that it does not have enough support).

The first property is referred to as **downward closure**. The second property, called the **antimonotonicity** property, helps to reduce the search space of possible solutions. That is, once an itemset is found to be small (not a large itemset), then any extension to that itemset, formed by adding one or more items to the set, will also yield a small itemset.

28.2.2 Apriori Algorithm

The first algorithm to use the downward closure and antimontonicity properties was the **Apriori algorithm**, shown as Algorithm 28.1.

We illustrate Algorithm 28.1 using the transaction data in Figure 28.1 using a minimum support of 0.5. The candidate 1-itemsets are {milk, bread, juice, cookies, eggs, coffee} and their respective supports are 0.75, 0.5, 0.5, 0.5, 0.25, and 0.25. The first four items qualify for L_1 since each support is greater than or equal to 0.5. In the first iteration of the repeat-loop, we extend the frequent 1-itemsets to create the candidate frequent 2-itemsets, C_2. C_2 contains {milk, bread}, {milk, juice}, {bread, juice}, {milk, cookies}, {bread, cookies}, and {juice, cookies}. Notice, for example, that {milk, eggs} does not appear in C_2 since {eggs} is small (by the antimonotonicity property) and does not appear in L_1. The supports for the six sets contained in C_2 are 0.25, 0.5, 0.25, 0.25, 0.5, and 0.25 and are computed by scanning the set of transactions. Only the second 2-itemset {milk, juice} and the fifth 2-itemset {bread, cookies} have support greater than or equal to 0.5. These two 2-itemsets form the frequent 2-itemsets, L_2.

Algorithm 28.1. Apriori Algorithm for Finding Frequent (Large) Itemsets

Input: Database of m transactions, D, and a minimum support, *mins*, represented as a fraction of m.

Output: Frequent itemsets, $L_1, L_2, \ldots, L_k$

Begin /* steps or statements are numbered for better readability */

1. Compute support(i_j) = count(i_j)/m for each individual item, $i_1, i_2, \ldots, i_n$ by scanning the database once and counting the number of transactions that item i_j appears in (that is, count(i_j));
2. The candidate frequent 1-itemset, C_1, will be the set of items $i_1, i_2, \ldots, i_n$;
3. The subset of items containing i_j from C_1 where support(i_j) >= mins becomes the frequent

   1-itemset, L_1;
4. $k = 1$;

   termination = false;

   repeat

1. $L_{k+1} =$;
2. Create the candidate frequent (k+1)-itemset, C_{k+1}, by combining members of L_k that have k–1 items in common (this forms candidate frequent (k+1)-itemsets by selectively extending frequent k-itemsets by one item);
3. In addition, only consider as elements of C_{k+1} those k+1 items such that every subset of size k appears in L_k;
4. Scan the database once and compute the support for each member of C_{k+1}; if the support for a member of C_{k+1} >= mins then add that member to L_{k+1};
5. If L_{k+1} is empty then termination = true

   else $k = k + 1$;

   until termination;

End;

In the next iteration of the repeat-loop, we construct candidate frequent 3-itemsets by adding additional items to sets in L_2. However, for no extension of itemsets in L_2 will all 2-item subsets be contained in L_2. For example, consider {milk, juice, bread}; the 2-itemset {milk, bread} is not in L_2, hence {milk, juice, bread} cannot be a frequent 3-itemset by the downward closure property. At this point the algorithm terminates with L_1 equal to {{milk}, {bread}, {juice}, {cookies}} and L_2 equal to {{milk, juice}, {bread, cookies}}.

Several other algorithms have been proposed to mine association rules. They vary mainly in terms of how the candidate itemsets are generated, and how the supports for the candidate itemsets are counted. Some algorithms use such data structures as bitmaps and hashtrees to keep information about itemsets. Several algorithms have been proposed that use multiple scans of the database because the potential number of itemsets, 2^m, can be too large to set up counters during a single scan. We will examine three improved algorithms (compared to the Apriori algorithm) for association rule mining: the Sampling algorithm, the Frequent-Pattern Tree algorithm, and the Partition algorithm.

28.2.3 Sampling Algorithm

The main idea for the **Sampling algorithm** is to select a small sample, one that fits in main memory, of the database of transactions and to determine the frequent itemsets from that sample. If those frequent itemsets form a superset of the frequent itemsets for the entire database, then we can determine the real frequent itemsets by scanning the remainder of the database in order to compute the exact support values for the superset itemsets. A superset of the frequent itemsets can usually be found from the sample by using, for example, the Apriori algorithm, with a lowered minimum support.

In some rare cases, some frequent itemsets may be missed and a second scan of the database is needed. To decide whether any frequent itemsets have been missed, the concept of the *negative border* is used. The negative border with respect to a frequent itemset, S, and set of items, I, is the minimal itemsets contained in PowerSet(I) and not in S. The basic idea is that the negative border of a set of frequent itemsets contains the closest itemsets that could also be frequent. Consider the case where a set X is not contained in the frequent itemsets. If all subsets of X are contained in the set of frequent itemsets, then X would be in the negative border.

We illustrate this with the following example. Consider the set of items I = {A, B, C, D, E} and let the combined frequent itemsets of size 1 to 3 be S = {{A}, {B}, {C}, {D}, {AB}, {AC}, {BC}, {AD}, {CD}, {ABC}}. The negative border is {{E}, {BD}, {ACD}}. The set {E} is the only 1-itemset not contained in S, {BD} is the only 2-itemset not in S but whose 1-itemset subsets are, and {ACD} is the only 3-itemset whose 2-itemset subsets are all in S. The negative border is important since it is necessary to determine the support for those itemsets in the negative border to ensure that no large itemsets are missed from analyzing the sample data.

Support for the negative border is determined when the remainder of the database is scanned. If we find that an itemset, X, in the negative border belongs in the set of all frequent itemsets, then there is a potential for a superset of X to also be frequent. If this happens, then a second pass over the database is needed to make sure that all frequent itemsets are found.

28.2.4 Frequent-Pattern (FP) Tree and FP-Growth Algorithm

The **Frequent-Pattern Tree** (**FP-tree**) is motivated by the fact that Apriori-based algorithms may generate and test a very large number of candidate itemsets. For example, with 1000 frequent 1-itemsets, the Apriori algorithm would have to generate

$$\binom{1000}{2}$$

or 499,500 candidate 2-itemsets. The **FP-Growth algorithm** is one approach that eliminates the generation of a large number of candidate itemsets.

The algorithm first produces a compressed version of the database in terms of an FP-tree (frequent-pattern tree). The FP-tree stores relevant itemset information and allows for the efficient discovery of frequent itemsets. The actual mining process adopts a divide-and-conquer strategy where the mining process is decomposed into a set of smaller tasks that each operates on a conditional FP-tree, a subset (projection) of the original tree. To start with, we examine how the FP-tree is constructed. The database is first scanned and the frequent 1-itemsets along with their support are computed. With this algorithm, the support is the *count* of transactions containing the item rather than the fraction of transactions containing the item. The frequent 1-itemsets are then sorted in nonincreasing order of their support. Next, the root of the FP-tree is created with a NULL label. The database is scanned a second time and for each transaction T in the database, the frequent 1-itemsets in T are placed in order as was done with the frequent 1-itemsets. We can designate this sorted list for T as consisting of a first item, the head, and the remaining items, the tail. The itemset information (head, tail) is inserted into the FP-tree recursively, starting at the root node, as follows:

1. If the current node, N, of the FP-tree has a child with an item name = head, then increment the count associated with node N by 1, else create a new node, N, with a count of 1, link N to its parent and link N with the item header table (used for efficient tree traversal).
2. If the tail is nonempty, then repeat step (1) using as the sorted list only the tail, that is, the old head is removed and the new head is the first item from the tail and the remaining items become the new tail.

The item header table, created during the process of building the FP-tree, contains three fields per entry for each frequent item: item identifier, support count, and node link. The item identifier and support count are self-explanatory. The node link is a pointer to an occurrence of that item in the FP-tree. Since multiple occurrences of a single item may appear in the FP-tree, these items are linked together as a list where the start of the list is pointed to by the node link in the item header table. We illustrate the building of the FP-tree using the transaction data in Figure 28.1. Let us use a minimum support of 2. One pass over the four transactions yields the following frequent 1-itemsets with associated support: {{(milk, 3)}, {(bread, 2)}, {(cookies, 2)}, {(juice, 2)}}. The database is scanned a second time and each transaction will be processed again.

For the first transaction, we create the sorted list, T = {milk, bread, cookies, juice}. The items in T are the frequent 1-itemsets from the first transaction. The items are ordered based on the nonincreasing ordering of the count of the 1-itemsets found in pass 1 (that is, milk first, bread second, and so on). We create a NULL root node for the FP-tree and insert *milk* as a child of the root, *bread* as a child of *milk*, *cookies* as a child of *bread*, and *juice* as a child of *cookies*. We adjust the entries for the frequent items in the item header table.

For the second transaction, we have the sorted list {milk, juice}. Starting at the root, we see that a child node with label *milk* exists, so we move to that node and update

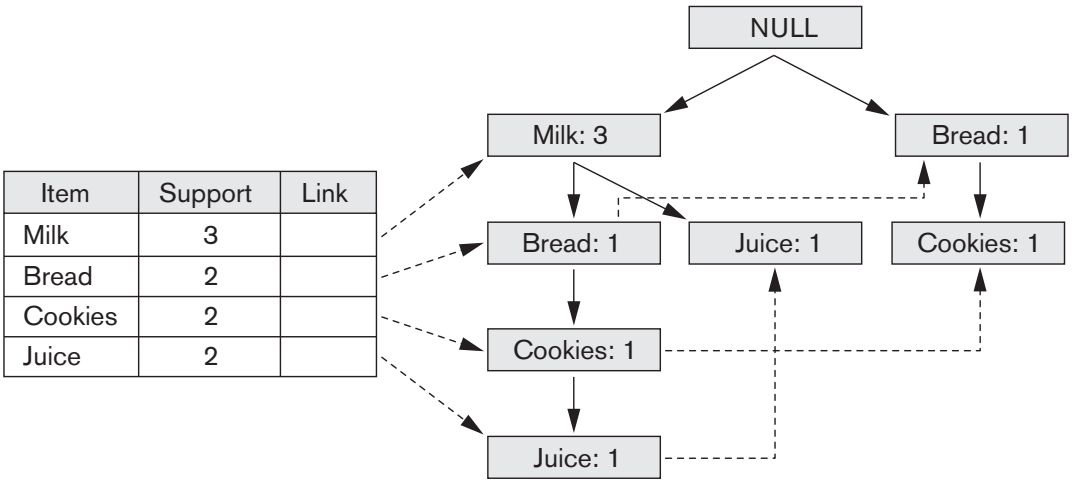

Figure 28.2
FP-tree and item header table.

its count (to account for the second transaction that contains milk). We see that there is no child of the current node with label *juice*, so we create a new node with label *juice*. The item header table is adjusted.

The third transaction only has 1-frequent item, {milk}. Again, starting at the root, we see that the node with label *milk* exists, so we move to that node, increment its count, and adjust the item header table. The final transaction contains frequent items, {bread, cookies}. At the root node, we see that a child with label *bread* does not exist. Thus, we create a new child of the root, initialize its counter, and then insert *cookies* as a child of this node and initialize its count. After the item header table is updated, we end up with the FP-tree and item header table as shown in Figure 28.2. If we examine this FP-tree, we see that it indeed represents the original transactions in a compressed format (that is, only showing the items from each transaction that are large 1-itemsets).

Algorithm 28.2 is used for mining the FP-tree for frequent patterns. With the FP-tree, it is possible to find all frequent patterns that contain a given frequent item by starting from the item header table for that item and traversing the node links in the FP-tree. The algorithm starts with a frequent 1-itemset (suffix pattern) and constructs its conditional pattern base and then its conditional FP-tree. The conditional pattern base is made up of a set of prefix paths, that is, where the frequent item is a suffix. For example, if we consider the item juice, we see from Figure 28.2 that there are two paths in the FP-tree that end with juice: (milk, bread, cookies, juice) and (milk, juice). The two associated prefix paths are (milk, bread, cookies) and (milk). The conditional FP-tree is constructed from the patterns in the conditional pattern base. The mining is recursively performed on this FP-tree. The frequent patterns are formed by concatenating the suffix pattern with the frequent patterns produced from a conditional FP-tree.

Algorithm 28.2. FP-Growth Algorithm for Finding Frequent Itemsets

Input: FP-tree and a minimum support, mins

Output: frequent patterns (itemsets)

procedure FP-growth (tree, alpha);

Begin
    if tree contains a single path *P* then
        for each combination, beta, of the nodes in the path
            generate pattern (beta ∪ alpha)
            with support = minimum support of nodes in beta
    else
        for each item, *i*, in the header of the tree do
            begin
                generate pattern beta = (*i* ∪ alpha) with support = *i*.support;
                construct beta's conditional pattern base;
                construct beta's conditional FP-tree, beta_tree;
                if beta_tree is not empty then
                    FP-growth(beta_tree, beta);
            end;
End;

We illustrate the algorithm using the data in Figure 28.1 and the tree in Figure 28.2. The procedure FP-growth is called with the two parameters: the original FP-tree and NULL for the variable alpha. Since the original FP-tree has more than a single path, we execute the else part of the first if statement. We start with the frequent item, juice. We will examine the frequent items in order of lowest support (that is, from the last entry in the table to the first). The variable beta is set to juice with support equal to 2.

Following the node link in the item header table, we construct the conditional pattern base consisting of two paths (with juice as suffix). These are (milk, bread, cookies: 1) and (milk: 1). The conditional FP-tree consists of only a single node, milk: 2. This is due to a support of only 1 for node bread and cookies, which is below the minimal support of 2. The algorithm is called recursively with an FP-tree of only a single node (that is, milk: 2) and a beta value of juice. Since this FP-tree only has one path, all combinations of beta and nodes in the path are generated—that is, {milk, juice}—with support of 2.

Next, the frequent item, cookies, is used. The variable beta is set to cookies with support = 2. Following the node link in the item header table, we construct the conditional pattern base consisting of two paths. These are (milk, bread: 1) and (bread: 1). The conditional FP-tree is only a single node, bread: 2. The algorithm is called recursively with an FP-tree of only a single node (that is, bread: 2) and a beta value of cookies. Since this FP-tree only has one path, all combinations of beta and nodes in the path are generated, that is, {bread, cookies} with support of 2. The frequent item, bread, is considered next. The variable beta is set to bread with support = 2. Following the node link in the item header table, we construct the conditional

pattern base consisting of one path, which is (milk: 1). The conditional FP-tree is empty since the count is less than the minimum support. Since the conditional FP-tree is empty, no frequent patterns will be generated.

The last frequent item to consider is milk. This is the top item in the item header table and as such has an empty conditional pattern base and empty conditional FP-tree. As a result, no frequent patterns are added. The result of executing the algorithm is the following frequent patterns (or itemsets) with their support: {{milk: 3}, {bread: 2}, {cookies: 2}, {juice: 2}, {milk, juice: 2}, {bread, cookies: 2}}.

28.2.5 Partition Algorithm

Another algorithm, called the **Partition algorithm**,[3] is summarized below. If we are given a database with a small number of potential large itemsets, say, a few thousand, then the support for all of them can be tested in one scan by using a partitioning technique. Partitioning divides the database into nonoverlapping subsets; these are individually considered as separate databases and all large itemsets for that partition, called *local frequent itemsets*, are generated in one pass. The Apriori algorithm can then be used efficiently on each partition if it fits entirely in main memory. Partitions are chosen in such a way that each partition can be accommodated in main memory. As such, a partition is read only once in each pass. The only caveat with the partition method is that the minimum support used for each partition has a slightly different meaning from the original value. The minimum support is based on the size of the partition rather than the size of the database for determining local frequent (large) itemsets. The actual support threshold value is the same as given earlier, but the support is computed only for a partition.

At the end of pass one, we take the union of all frequent itemsets from each partition. This forms the global candidate frequent itemsets for the entire database. When these lists are merged, they may contain some false positives. That is, some of the itemsets that are frequent (large) in one partition may not qualify in several other partitions and hence may not exceed the minimum support when the original database is considered. Note that there are no false negatives; no large itemsets will be missed. The global candidate large itemsets identified in pass one are verified in pass two; that is, their actual support is measured for the *entire* database. At the end of phase two, all global large itemsets are identified. The Partition algorithm lends itself naturally to a parallel or distributed implementation for better efficiency. Further improvements to this algorithm have been suggested.[4]

28.2.6 Other Types of Association Rules

Association Rules among Hierarchies. There are certain types of associations that are particularly interesting for a special reason. These associations occur among

[3]See Savasere et al. (1995) for details of the algorithm, the data structures used to implement it, and its performance comparisons.

[4]See Cheung et al. (1996) and Lin and Dunham (1998).

hierarchies of items. Typically, it is possible to divide items among disjoint hierarchies based on the nature of the domain. For example, foods in a supermarket, items in a department store, or articles in a sports shop can be categorized into classes and subclasses that give rise to hierarchies. Consider Figure 28.3, which shows the taxonomy of items in a supermarket. The figure shows two hierarchies—beverages and desserts, respectively. The entire groups may not produce associations of the form beverages => desserts, or desserts => beverages. However, associations of the type Healthy-brand frozen yogurt => bottled water, or Rich cream-brand ice cream => wine cooler may produce enough confidence and support to be valid association rules of interest.

Therefore, if the application area has a natural classification of the itemsets into hierarchies, discovering associations *within* the hierarchies is of no particular interest. The ones of specific interest are associations *across* hierarchies. They may occur among item groupings at different levels.

Multidimensional Associations. Discovering association rules involves searching for patterns in a file. In Figure 28.1, we have an example of a file of customer transactions with three dimensions: Transaction_id, Time, and Items_bought. However, our data mining tasks and algorithms introduced up to this point only involve one dimension: Items_bought. The following rule is an example of including the label of the single dimension: Items_bought(milk) => Items_bought(juice). It may be of interest to find association rules that involve multiple dimensions, for

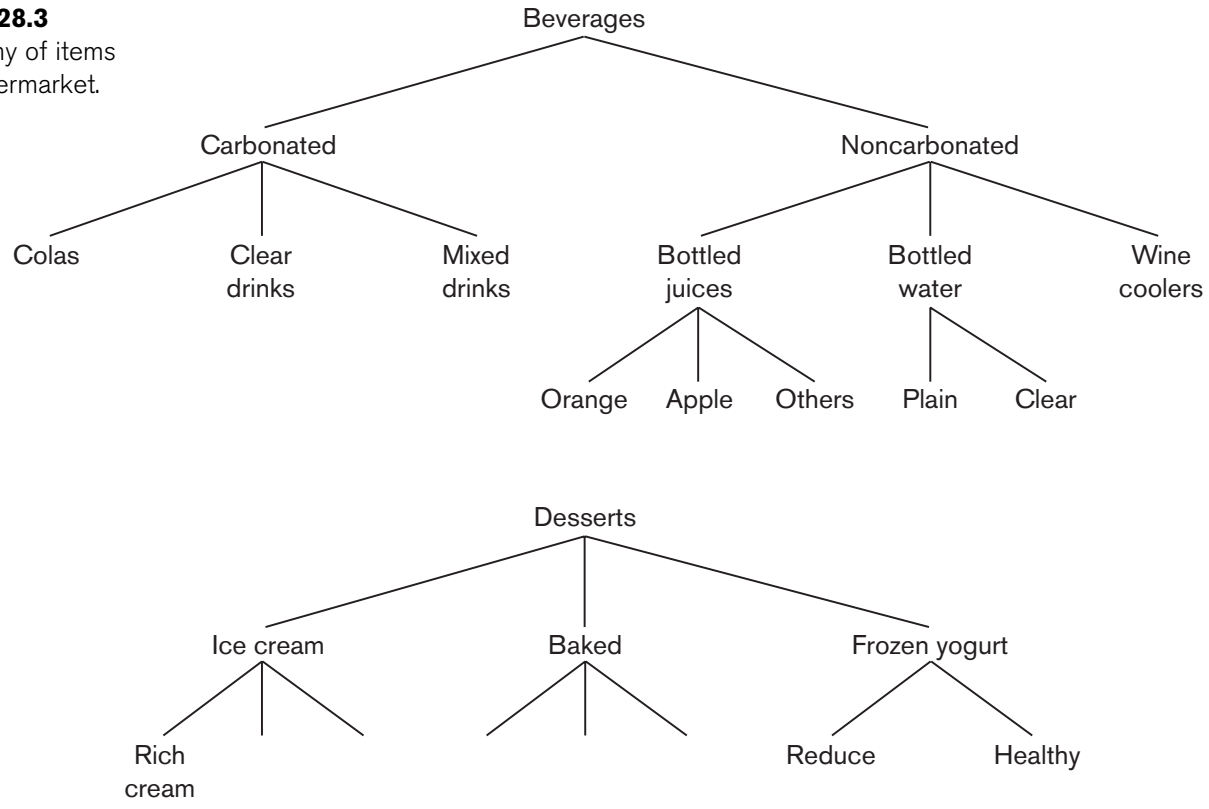

Figure 28.3
Taxonomy of items in a supermarket.

example, Time(6:30…8:00) => Items_bought(milk). Rules like these are called *multidimensional association rules*. The dimensions represent attributes of records of a file or, in terms of relations, columns of rows of a relation, and can be categorical or quantitative. Categorical attributes have a finite set of values that display no ordering relationship. Quantitative attributes are numeric and their values display an ordering relationship, for example, <. Items_bought is an example of a categorical attribute and Transaction_id and Time are quantitative.

One approach to handling a quantitative attribute is to partition its values into nonoverlapping intervals that are assigned labels. This can be done in a static manner based on domain-specific knowledge. For example, a concept hierarchy may group values for Salary into three distinct classes: low income ($0 < \text{Salary} < 29{,}999$), middle income ($30{,}000 < \text{Salary} < 74{,}999$), and high income ($\text{Salary} > 75{,}000$). From here, the typical Apriori-type algorithm or one of its variants can be used for the rule mining since the quantitative attributes now look like categorical attributes. Another approach to partitioning is to group attribute values based on data distribution, for example, equi-depth partitioning, and to assign integer values to each partition. The partitioning at this stage may be relatively fine, that is, a larger number of intervals. Then during the mining process, these partitions may combine with other adjacent partitions if their support is less than some predefined maximum value. An Apriori-type algorithm can be used here as well for the data mining.

Negative Associations. The problem of discovering a negative association is harder than that of discovering a positive association. A negative association is of the following type: *60 percent of customers who buy potato chips do not buy bottled water*. (Here, the 60 percent refers to the confidence for the negative association rule.) In a database with 10,000 items, there are 210,000 possible combinations of items, a majority of which do not appear even once in the database. If the absence of a certain item combination is taken to mean a negative association, then we potentially have millions and millions of negative association rules with RHSs that are of no interest at all. The problem, then, is to find only *interesting* negative rules. In general, we are interested in cases in which two specific sets of items appear very rarely in the same transaction. This poses two problems.

1. For a total item inventory of 10,000 items, the probability of any two being bought together is $(1/10{,}000) * (1/10{,}000) = 10^{-8}$. If we find the actual support for these two occurring together to be zero, that does not represent a significant departure from expectation and hence is not an interesting (negative) association.
2. The other problem is more serious. We are looking for item combinations with very low support, and there are millions and millions with low or even zero support. For example, a data set of 10 million transactions has most of the 2.5 billion pairwise combinations of 10,000 items missing. This would generate billions of useless rules.

Therefore, to make negative association rules interesting, we must use prior knowledge about the itemsets. One approach is to use hierarchies. Suppose we use the hierarchies of soft drinks and chips shown in Figure 28.4.

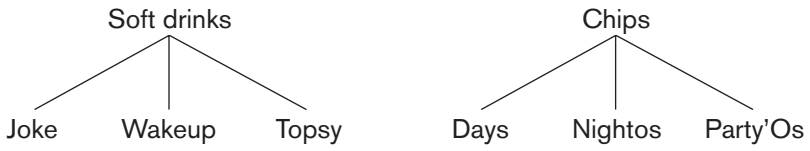

Figure 28.4
Simple hierarchy of soft drinks and chips.

A strong positive association has been shown between soft drinks and chips. If we find a large support for the fact that when customers buy Days chips they predominantly buy Topsy and *not* Joke and *not* Wakeup, that would be interesting because we would normally expect that if there is a strong association between Days and Topsy, there should also be such a strong association between Days and Joke or Days and Wakeup.[5]

In the frozen yogurt and bottled water groupings shown in Figure 28.3, suppose the Reduce versus Healthy-brand division is 80–20 and the Plain and Clear brands division is 60–40 among respective categories. This would give a joint probability of Reduce frozen yogurt being purchased with Plain bottled water as 48 percent among the transactions containing a frozen yogurt and bottled water. If this support, however, is found to be only 20 percent, it would indicate a significant negative association among Reduce yogurt and Plain bottled water; again, that would be interesting.

The problem of finding negative association is important in the above situations given the domain knowledge in the form of item generalization hierarchies (that is, the beverage given and desserts hierarchies shown in Figure 28.3), the existing positive associations (such as between the frozen yogurt and bottled water groups), and the distribution of items (such as the name brands within related groups). The scope of discovery of negative associations is limited in terms of knowing the item hierarchies and distributions. Exponential growth of negative associations remains a challenge.

28.2.7 Additional Considerations for Association Rules

Mining association rules in real-life databases is complicated by the following factors:

- The cardinality of itemsets in most situations is extremely large, and the volume of transactions is very high as well. Some operational databases in retailing and communication industries collect tens of millions of transactions per day.
- Transactions show variability in such factors as geographic location and seasons, making sampling difficult.
- Item classifications exist along multiple dimensions. Hence, driving the discovery process with domain knowledge, particularly for negative rules, is extremely difficult.

[5]For simplicity we are assuming a uniform distribution of transactions among members of a hierarchy.

- Quality of data is variable; significant problems exist with missing, erroneous, conflicting, as well as redundant data in many industries.

28.3 Classification

Classification is the process of learning a model that describes different classes of data. The classes are predetermined. For example, in a banking application, customers who apply for a credit card may be classified as a *poor risk*, *fair risk*, or *good risk*. Hence this type of activity is also called **supervised learning**. Once the model is built, it can be used to classify new data. The first step—learning the model—is accomplished by using a training set of data that has already been classified. Each record in the training data contains an attribute, called the *class* label, which indicates which class the record belongs to. The model that is produced is usually in the form of a decision tree or a set of rules. Some of the important issues with regard to the model and the algorithm that produces the model include the model's ability to predict the correct class of new data, the computational cost associated with the algorithm, and the scalability of the algorithm.

We will examine the approach where our model is in the form of a decision tree. A **decision tree** is simply a graphical representation of the description of each class or, in other words, a representation of the classification rules. A sample decision tree is pictured in Figure 28.5. We see from Figure 28.5 that if a customer is *married* and if salary >= 50K, then they are a good risk for a bank credit card. This is one of the rules that describe the class *good risk*. Traversing the decision tree from the root to each leaf node forms other rules for this class and the two other classes. Algorithm 28.3 shows the procedure for constructing a decision tree from a training data set. Initially, all training samples are at the root of the tree. The samples are partitioned

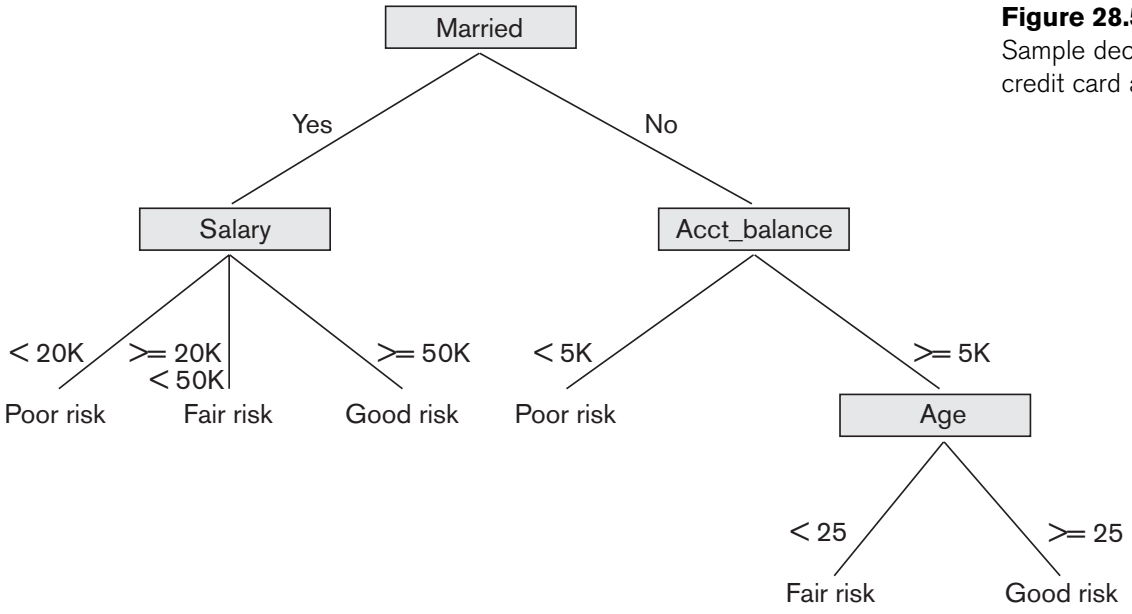

Figure 28.5
Sample decision tree for credit card applications.

recursively based on selected attributes. The attribute used at a node to partition the samples is the one with the best splitting criterion, for example, the one that maximizes the information gain measure.

Algorithm 28.3. Algorithm for Decision Tree Induction

Input: Set of training data records: $R_1, R_2, ..., R_m$ and set of attributes: $A_1, A_2, ..., A_n$

Output: Decision tree

```
procedure Build_tree (records, attributes);
Begin
    create a node N;
    if all records belong to the same class, C then
        return N as a leaf node with class label C;
    if attributes is empty then
        return N as a leaf node with class label C, such that the majority of
        records belong to it;
    select attribute A_i (with the highest information gain) from attributes;
    label node N with A_i;
    for each known value, v_j, of A_i do
        begin
            add a branch from node N for the condition A_i = v_j;
            S_j = subset of records where A_i = v_j;
            if S_j is empty then
                add a leaf, L, with class label C, such that the majority of
                        records belong to it and return L
            else add the node returned by Build_tree(S_j, attributes – A_i);
        end;
End;
```

Before we illustrate Algorithm 28.3, we will explain the **information gain** measure in more detail. The use of **entropy** as the information gain measure is motivated by the goal of minimizing the information needed to classify the sample data in the resulting partitions and thus minimizing the expected number of conditional tests needed to classify a new record. The expected information needed to classify training data of s samples, where the Class attribute has n values $(v_1, ..., v_n)$ and s_i is the number of samples belonging to class label v_i, is given by

$$I(S_1, S_2, ..., S_n) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where p_i is the probability that a random sample belongs to the class with label v_i. An estimate for p_i is s_i/s. Consider an attribute A with values $\{v_1, ..., v_m\}$ used as the test attribute for splitting in the decision tree. Attribute A partitions the samples into the subsets $S_1, ..., S_m$ where samples in each S_j have a value of v_j for attribute A. Each S_j may contain samples that belong to any of the classes. The number of

samples in S_j that belong to class i can be denoted as s_{ij}. The entropy associated with using attribute A as the test attribute is defined as

$$E(A)=\sum_{j=1}^{m}\frac{S_{1j}+\ldots+S_{nj}}{S}\times I\left(S_{1j},\ldots,S_{nj}\right)$$

$I(s_{1j}, \ldots, s_{nj})$ can be defined using the formulation for $I(s_1, \ldots, s_n)$ with p_i being replaced by p_{ij} where $p_{ij} = s_{ij}/s_j$. Now the information gain by partitioning on attribute A, Gain(A), is defined as $I(s_1, \ldots, s_n) - E(A)$. We can use the sample training data from Figure 28.6 to illustrate the algorithm.

The attribute RID represents the record identifier used for identifying an individual record and is an internal attribute. We use it to identify a particular record in our example. First, we compute the expected information needed to classify the training data of 6 records as $I(s_1, s_2)$ where there are two classes: the first class label value corresponds to *yes* and the second to *no*. So,

$$I(3,3) = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1.$$

Now, we compute the entropy for each of the four attributes as shown below. For Married = yes, we have $s_{11} = 2$, $s_{21} = 1$ and $I(s_{11}, s_{21}) = 0.92$. For Married = no, we have $s_{12} = 1$, $s_{22} = 2$ and $I(s_{12}, s_{22}) = 0.92$. So, the expected information needed to classify a sample using attribute Married as the partitioning attribute is

$$E(\text{Married}) = 3/6\ I(s_{11}, s_{21}) + 3/6\ \text{I}(s_{12}, s_{22}) = 0.92.$$

The gain in information, Gain(Married), would be 1 – 0.92 = 0.08. If we follow similar steps for computing the gain with respect to the other three attributes we end up with

E(Salary) = 0.33	and	Gain(Salary) = 0.67
E(Acct_balance) = 0.92	and	Gain(Acct_balance) = 0.08
E(Age) = 0.54	and	Gain(Age) = 0.46

Since the greatest gain occurs for attribute Salary, it is chosen as the partitioning attribute. The root of the tree is created with label *Salary* and has three branches, one for each value of Salary. For two of the three values, that is, <20K and >=50K, all the samples that are partitioned accordingly (records with RIDs 4 and 5 for <20K

RID	Married	Salary	Acct_balance	Age	Loanworthy
1	no	>=50K	<5K	>=25	yes
2	yes	>=50K	>=5K	>=25	yes
3	yes	20K. . .50K	<5K	<25	no
4	no	<20K	>=5K	<25	no
5	no	<20K	<5K	>=25	no
6	yes	20K. . .50K	>=5K	>=25	yes

Figure 28.6
Sample training data for classification algorithm.

and records with RIDs 1 and 2 for >=50K) fall within the same class *loanworthy no* and *loanworthy yes* respectively for those two values. So we create a leaf node for each. The only branch that needs to be expanded is for the value 20K…50K with two samples, records with RIDs 3 and 6 in the training data. Continuing the process using these two records, we find that Gain(Married) is 0, Gain(Acct_balance) is 1, and Gain(Age) is 1.

We can choose either Age or Acct_balance since they both have the largest gain. Let us choose Age as the partitioning attribute. We add a node with label *Age* that has two branches, less than 25, and greater or equal to 25. Each branch partitions the remaining sample data such that one sample record belongs to each branch and hence one class. Two leaf nodes are created and we are finished. The final decision tree is pictured in Figure 28.7.

28.4 Clustering

The previous data mining task of classification deals with partitioning data based on using a preclassified training sample. However, it is often useful to partition data without having a training sample; this is also known as **unsupervised learning**. For example, in business, it may be important to determine groups of customers who have similar buying patterns, or in medicine, it may be important to determine groups of patients who show similar reactions to prescribed drugs. The goal of clustering is to place records into groups, such that records in a group are similar to each other and dissimilar to records in other groups. The groups are usually *disjoint*.

An important facet of clustering is the similarity function that is used. When the data is numeric, a similarity function based on distance is typically used. For example, the Euclidean distance can be used to measure similarity. Consider two n-dimensional data points (records) r_j and r_k. We can consider the value for the ith dimension as r_{ji} and r_{ki} for the two records. The Euclidean distance between points r_j and r_k in n-dimensional space is calculated as:

$$\text{Distance}(r_j, r_k) = \sqrt{|r_{j1} - r_{k1}|^2 + |r_{j2} - r_{k2}|^2 + \ldots + |r_{jn} - r_{kn}|^2}$$

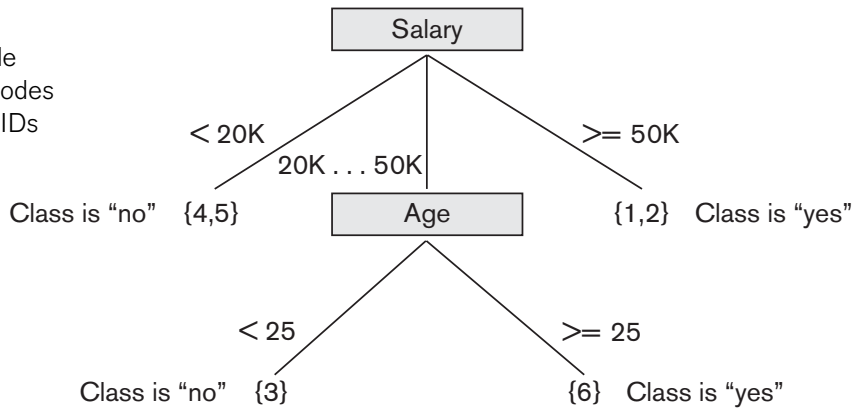

Figure 28.7
Decision tree based on sample training data where the leaf nodes are represented by a set of RIDs of the partitioned records.

The smaller the distance between two points, the greater is the similarity as we think of them. A classic clustering algorithm is the *k*-Means algorithm, Algorithm 28.4.

Algorithm 28.4. *k*-Means Clustering Algorithm

Input: a database D, of m records, $r_1, ..., r_m$ and a desired number of clusters k

Output: set of k clusters that minimizes the squared error criterion

Begin
randomly choose k records as the centroids for the k clusters;
repeat
assign each record, r_i, to a cluster such that the distance between r_i and the cluster centroid (mean) is the smallest among the k clusters;
recalculate the centroid (mean) for each cluster based on the records assigned to the cluster;
until no change;
End;

The algorithm begins by randomly choosing k records to represent the centroids (means), $m_1, ..., m_k$, of the clusters, $C_1, ..., C_k$. All the records are placed in a given cluster based on the distance between the record and the cluster mean. If the distance between m_i and record r_j is the smallest among all cluster means, then record r_j is placed in cluster C_i. Once all records have been initially placed in a cluster, the mean for each cluster is recomputed. Then the process repeats, by examining each record again and placing it in the cluster whose mean is closest. Several iterations may be needed, but the algorithm will converge, although it may terminate at a local optimum. The terminating condition is usually the squared-error criterion. For clusters $C_1, ..., C_k$ with means $m_1, ..., m_k$, the error is defined as:

$$\text{Error} = \sum_{i=1}^{k} \sum_{\forall r_j \in C_i} \text{Distance}(r_j, m_i)^2$$

We will examine how Algorithm 28.4 works with the (two-dimensional) records in Figure 28.8. Assume that the number of desired clusters k is 2. Let the algorithm choose records with RID 3 for cluster C_1 and RID 6 for cluster C_2 as the initial cluster centroids. The remaining records will be assigned to one of those clusters during the

RID	Age	Years_of_service
1	30	5
2	50	25
3	50	15
4	25	5
5	30	10
6	55	25

Figure 28.8
Sample 2-dimensional records for clustering example (the RID column is not considered).

first iteration of the repeat loop. The record with RID 1 has a distance from C_1 of 22.4 and a distance from C_2 of 32.0, so it joins cluster C_1. The record with RID 2 has a distance from C_1 of 10.0 and a distance from C_2 of 5.0, so it joins cluster C_2. The record with RID 4 has a distance from C_1 of 25.5 and a distance from C_2 of 36.6, so it joins cluster C_1. The record with RID 5 has a distance from C_1 of 20.6 and a distance from C_2 of 29.2, so it joins cluster C_1. Now, the new means (centroids) for the two clusters are computed. The mean for a cluster, C_i, with n records of m dimensions is the vector:

$$\bar{C}_i = \left(\frac{1}{n} \sum_{\forall r_j \in C_i} r_{ji}, \ldots, \frac{1}{n} \sum_{\forall r_j \in C_i} r_{jm} \right)$$

The new mean for C_1 is (33.75, 8.75) and the new mean for C_2 is (52.5, 25). A second iteration proceeds and the six records are placed into the two clusters as follows: records with RIDs 1, 4, 5 are placed in C_1 and records with RIDs 2, 3, 6 are placed in C_2. The mean for C_1 and C_2 is recomputed as (28.3, 6.7) and (51.7, 21.7), respectively. In the next iteration, all records stay in their previous clusters and the algorithm terminates.

Traditionally, clustering algorithms assume that the entire data set fits in main memory. More recently, researchers have developed algorithms that are efficient and are scalable for very large databases. One such algorithm is called BIRCH. BIRCH is a hybrid approach that uses both a hierarchical clustering approach, which builds a tree representation of the data, as well as additional clustering methods, which are applied to the leaf nodes of the tree. Two input parameters are used by the BIRCH algorithm. One specifies the amount of available main memory and the other is an initial threshold for the radius of any cluster. Main memory is used to store descriptive cluster information such as the center (mean) of a cluster and the radius of the cluster (clusters are assumed to be spherical in shape). The radius threshold affects the number of clusters that are produced. For example, if the radius threshold value is large, then few clusters of many records will be formed. The algorithm tries to maintain the number of clusters such that their radius is below the radius threshold. If available memory is insufficient, then the radius threshold is increased.

The BIRCH algorithm reads the data records sequentially and inserts them into an in-memory tree structure, which tries to preserve the clustering structure of the data. The records are inserted into the appropriate leaf nodes (potential clusters) based on the distance between the record and the cluster center. The leaf node where the insertion happens may have to split, depending upon the updated center and radius of the cluster and the radius threshold parameter. Additionally, when splitting, extra cluster information is stored, and if memory becomes insufficient, then the radius threshold will be increased. Increasing the radius threshold may actually produce a side effect of reducing the number of clusters since some nodes may be merged.

Overall, BIRCH is an efficient clustering method with a linear computational complexity in terms of the number of records to be clustered.

28.5 Approaches to Other Data Mining Problems

28.5.1 Discovery of Sequential Patterns

The discovery of sequential patterns is based on the concept of a sequence of itemsets. We assume that transactions such as the supermarket-basket transactions we discussed previously are ordered by time of purchase. That ordering yields a sequence of itemsets. For example, {milk, bread, juice}, {bread, eggs}, {cookies, milk, coffee} may be such a **sequence of itemsets** based on three visits by the same customer to the store. The **support** for a sequence S of itemsets is the percentage of the given set U of sequences of which S is a subsequence. In this example, {milk, bread, juice} {bread, eggs} and {bread, eggs} {cookies, milk, coffee} are considered **subsequences**. The problem of identifying sequential patterns, then, is to find all subsequences from the given sets of sequences that have a user-defined minimum support. The sequence S_1, S_2, S_3, ... is a **predictor** of the fact that a customer who buys itemset S_1 is likely to buy itemset S_2 and then S_3, and so on. This prediction is based on the frequency (support) of this sequence in the past. Various algorithms have been investigated for sequence detection.

28.5.2 Discovery of Patterns in Time Series

Time series are sequences of events; each event may be a given fixed type of a transaction. For example, the closing price of a stock or a fund is an event that occurs every weekday for each stock and fund. The sequence of these values per stock or fund constitutes a time series. For a time series, one may look for a variety of patterns by analyzing sequences and subsequences as we did above. For example, we might find the period during which the stock rose or held steady for n days, or we might find the longest period over which the stock had a fluctuation of no more than 1 percent over the previous closing price, or we might find the quarter during which the stock had the most percentage gain or percentage loss. Time series may be compared by establishing measures of similarity to identify companies whose stocks behave in a similar fashion. Analysis and mining of time series is an extended functionality of temporal data management (see Chapter 26).

28.5.3 Regression

Regression is a special application of the classification rule. If a classification rule is regarded as a function over the variables that maps these variables into a target class variable, the rule is called a **regression rule**. A general application of regression occurs when, instead of mapping a tuple of data from a relation to a specific class, the value of a variable is predicted based on that tuple. For example, consider a relation

LAB_TESTS (patient ID, test 1, test 2, ..., test n)

which contains values that are results from a series of n tests for one patient. The target variable that we wish to predict is P, the probability of survival of the patient. Then the rule for regression takes the form:

(test 1 in range$_1$) and (test 2 in range$_2$) and ... (test n in range$_n$) $\Rightarrow P = x$, or $x < P \leq y$

The choice depends on whether we can predict a unique value of P or a range of values for P. If we regard P as a function:

$P = f$ (test 1, test 2, ..., test n)

the function is called a **regression function** to predict P. In general, if the function appears as

$$Y = f(X_1, X_2, ..., X_n),$$

and f is linear in the domain variables x_i, the process of deriving f from a given set of tuples for $<X_1, X_2, ..., X_n, y>$ is called **linear regression**. Linear regression is a commonly used statistical technique for fitting a set of observations or points in n dimensions with the target variable y.

Regression analysis is a very common tool for analysis of data in many research domains. The discovery of the function to predict the target variable is equivalent to a data mining operation.

28.5.4 Neural Networks

A **neural network** is a technique derived from artificial intelligence research that uses generalized regression and provides an iterative method to carry it out. Neural networks use the curve-fitting approach to infer a function from a set of samples. This technique provides a *learning approach*; it is driven by a test sample that is used for the initial inference and learning. With this kind of learning method, responses to new inputs may be able to be interpolated from the known samples. This interpolation, however, depends on the world model (internal representation of the problem domain) developed by the learning method.

Neural networks can be broadly classified into two categories: supervised and unsupervised networks. Adaptive methods that attempt to reduce the output error are **supervised learning** methods, whereas those that develop internal representations without sample outputs are called **unsupervised learning** methods.

Neural networks self-adapt; that is, they learn from information about a specific problem. They perform well on classification tasks and are therefore useful in data mining. Yet, they are not without problems. Although they learn, they do not provide a good representation of *what* they have learned. Their outputs are highly quantitative and not easy to understand. As another limitation, the internal representations developed by neural networks are not unique. Also, in general, neural networks have trouble modeling time series data. Despite these shortcomings, they are popular and frequently used by several commercial vendors.

28.5.5 Genetic Algorithms

Genetic algorithms (GAs) are a class of randomized search procedures capable of adaptive and robust search over a wide range of search space topologies. Modeled after the adaptive emergence of biological species from evolutionary mechanisms, and introduced by Holland,[6] GAs have been successfully applied in such diverse fields as image analysis, scheduling, and engineering design.

Genetic algorithms extend the idea from human genetics of the four-letter alphabet (based on the A,C,T,G nucleotides) of the human DNA code. The construction of a genetic algorithm involves devising an alphabet that encodes the solutions to the decision problem in terms of strings of that alphabet. Strings are equivalent to individuals. A fitness function defines which solutions can survive and which cannot. The ways in which solutions can be combined are patterned after the cross-over operation of cutting and combining strings from a father and a mother. An initial population of a well-varied population is provided, and a game of evolution is played in which mutations occur among strings. They combine to produce a new generation of individuals; the fittest individuals survive and mutate until a family of successful solutions develops.

The solutions produced by GAs are distinguished from most other search techniques by the following characteristics:

- A GA search uses a set of solutions during each generation rather than a single solution.
- The search in the string-space represents a much larger parallel search in the space of encoded solutions.
- The memory of the search done is represented solely by the set of solutions available for a generation.
- A genetic algorithm is a randomized algorithm since search mechanisms use probabilistic operators.
- While progressing from one generation to the next, a GA finds near-optimal balance between knowledge acquisition and exploitation by manipulating encoded solutions.

Genetic algorithms are used for problem solving and clustering problems. Their ability to solve problems in parallel provides a powerful tool for data mining. The drawbacks of GAs include the large overproduction of individual solutions, the random character of the searching process, and the high demand on computer processing. In general, substantial computing power is required to achieve anything of significance with genetic algorithms.

[6]Holland's seminal work (1975) entitled *Adaptation in Natural and Artificial Systems* introduced the idea of genetic algorithms.

28.6 Applications of Data Mining

Data mining technologies can be applied to a large variety of decision-making contexts in business. In particular, areas of significant payoffs are expected to include the following:

- **Marketing.** Applications include analysis of consumer behavior based on buying patterns; determination of marketing strategies including advertising, store location, and targeted mailing; segmentation of customers, stores, or products; and design of catalogs, store layouts, and advertising campaigns.
- **Finance.** Applications include analysis of creditworthiness of clients, segmentation of account receivables, performance analysis of finance investments like stocks, bonds, and mutual funds; evaluation of financing options; and fraud detection.
- **Manufacturing.** Applications involve optimization of resources like machines, manpower, and materials; and optimal design of manufacturing processes, shop-floor layouts, and product design, such as for automobiles based on customer requirements.
- **Health Care.** Applications include discovery of patterns in radiological images, analysis of microarray (gene-chip) experimental data to cluster genes and to relate to symptoms or diseases, analysis of side effects of drugs and effectiveness of certain treatments, optimization of processes within a hospital, and the relationship of patient wellness data with doctor qualifications.

28.7 Commercial Data Mining Tools

Currently, commercial data mining tools use several common techniques to extract knowledge. These include association rules, clustering, neural networks, sequencing, and statistical analysis. We discussed these earlier. Also used are decision trees, which are a representation of the rules used in classification or clustering, and statistical analyses, which may include regression and many other techniques. Other commercial products use advanced techniques such as genetic algorithms, case-based reasoning, Bayesian networks, nonlinear regression, combinatorial optimization, pattern matching, and fuzzy logic. In this chapter we have already discussed some of these.

Most data mining tools use the ODBC (Open Database Connectivity) interface. ODBC is an industry standard that works with databases; it enables access to data in most of the popular database programs such as Access, dBASE, Informix, Oracle, and SQL Server. Some of these software packages provide interfaces to specific database programs; the most common are Oracle, Access, and SQL Server. Most of the tools work in the Microsoft Windows environment and a few work in the UNIX operating system. The trend is for all products to operate under the Microsoft Windows environment. One tool, Data Surveyor, mentions ODMG compliance; see Chapter 11 where we discuss the ODMG object-oriented standard.

In general, these programs perform sequential processing in a single machine. Many of these products work in the client-server mode. Some products incorporate parallel processing in parallel computer architectures and work as a part of online analytical processing (OLAP) tools.

28.7.1 User Interface

Most of the tools run in a graphical user interface (GUI) environment. Some products include sophisticated visualization techniques to view data and rules (for example, SGI's MineSet), and are even able to manipulate data this way interactively. Text interfaces are rare and are more common in tools available for UNIX, such as IBM's Intelligent Miner.

28.7.2 Application Programming Interface

Usually, the application programming interface (API) is an optional tool. Most products do not permit using their internal functions. However, some of them allow the application programmer to reuse their code. The most common interfaces are C libraries and Dynamic Link Libraries (DLLs). Some tools include proprietary database command languages.

In Table 28.1 we list 11 representative data mining tools. To date, there are almost one hundred commercial data mining products available worldwide. Non-U.S. products include Data Surveyor from the Netherlands and PolyAnalyst from Russia.

28.7.3 Future Directions

Data mining tools are continually evolving, building on ideas from the latest scientific research. Many of these tools incorporate the latest algorithms taken from artificial intelligence (AI), statistics, and optimization.

Currently, fast processing is done using modern database techniques—such as distributed processing—in client-server architectures, in parallel databases, and in data warehousing. For the future, the trend is toward developing Internet capabilities more fully. Additionally, hybrid approaches will become commonplace, and processing will be done using all resources available. Processing will take advantage of both parallel and distributed computing environments. This shift is especially important because modern databases contain very large amounts of information. Not only are multimedia databases growing, but also image storage and retrieval are slow operations. Also, the cost of secondary storage is decreasing, so massive information storage will be feasible, even for small companies. Thus, data mining programs will have to deal with larger sets of data of more companies.

Most of data mining software will use the ODBC standard to extract data from business databases; proprietary input formats can be expected to disappear. There is a definite need to include nonstandard data, including images and other multimedia data, as source data for data mining.

Table 28.1 Some Representative Data Mining Tools

Company	Product	Technique	Platform	Interface*
AcknoSoft	Kate	Decision trees, Case-based reasoning	Windows UNIX	Microsoft Access
Angoss	Knowledge SEEKER	Decision trees, Statistics	Windows	ODBC
Business Objects	Business Miner	Neural nets, Machine learning	Windows	ODBC
CrossZ	QueryObject	Statistical analysis, Optimization algorithm	Windows MVS UNIX	ODBC
Data Distilleries	Data Surveyor	Comprehensive; can mix different types of data mining	UNIX	ODBC ODMG-compliant
DBMiner Technology Inc.	DBMiner	OLAP analysis, Associations, Classification, Clustering algorithms	Windows	Microsoft 7.0 OLAP
IBM	Intelligent Miner	Classification, Association rules, Predictive models	UNIX (AIX)	IBM DB2
Megaputer Intelligence	PolyAnalyst	Symbolic knowledge acquisition, Evolutionary programming	Windows OS/2	ODBC Oracle DB2
NCR	Management Discovery Tool (MDT)	Association rules	Windows	ODBC
Purple Insight	MineSet	Decision trees, Association rules	UNIX (Irix)	Oracle Sybase Informix
SAS	Enterprise Miner	Decision trees, Association rules, Neural nets, Regression, Clustering	UNIX (Solaris) Windows Macintosh	ODBC Oracle AS/400

*ODBC: Open Data Base Connectivity

ODMG: Object Data Management Group

28.8 Summary

In this chapter we surveyed the important discipline of data mining, which uses database technology to discover additional knowledge or patterns in the data. We gave an illustrative example of knowledge discovery in databases, which has a wider scope than data mining. For data mining, among the various techniques, we focused on the details of association rule mining, classification, and clustering. We presented algorithms in each of these areas and illustrated with examples of how those algorithms work.

A variety of other techniques, including the AI-based neural networks and genetic algorithms, were also briefly discussed. Active research is ongoing in data mining and we have outlined some of the expected research directions. In the future database technology products market, a great deal of data mining activity is expected. We summarized 11 out of nearly one hundred data mining tools available; future research is expected to extend the number and functionality significantly.

Review Questions

28.1. What are the different phases of the knowledge discovery from databases? Describe a complete application scenario in which new knowledge may be mined from an existing database of transactions.

28.2. What are the goals or tasks that data mining attempts to facilitate?

28.3. What are the five types of knowledge produced from data mining?

28.4. What are association rules as a type of knowledge? Give a definition of support and confidence and use them to define an association rule.

28.5. What is the downward closure property? How does it aid in developing an efficient algorithm for finding association rules, that is, with regard to finding large itemsets?

28.6. What was the motivating factor for the development of the FP-tree algorithm for association rule mining?

28.7. Describe an association rule among hierarchies with an example.

28.8. What is a negative association rule in the context of the hierarchy in Figure 28.3?

28.9. What are the difficulties of mining association rules from large databases?

28.10. What are classification rules and how are decision trees related to them?

28.11. What is entropy and how is it used in building decision trees?

28.12. How does clustering differ from classification?

28.13. Describe neural networks and genetic algorithms as techniques for data mining. What are the main difficulties in using these techniques?

Exercises

28.14. Apply the Apriori algorithm to the following data set.

Trans_id	Items_purchased
101	milk, bread, eggs
102	milk, juice
103	juice, butter
104	milk, bread, eggs
105	coffee, eggs
106	coffee
107	coffee, juice
108	milk, bread, cookies, eggs
109	cookies, butter
110	milk, bread

The set of items is {milk, bread, cookies, eggs, butter, coffee, juice}. Use 0.2 for the minimum support value.

28.15. Show two rules that have a confidence of 0.7 or greater for an itemset containing three items from Exercise 28.14.

28.16. For the Partition algorithm, prove that any frequent itemset in the database must appear as a local frequent itemset in at least one partition.

28.17. Show the FP-tree that would be made for the data from Exercise 28.14.

28.18. Apply the FP-Growth algorithm to the FP-tree from Exercise 28.17 and show the frequent itemsets.

28.19. Apply the classification algorithm to the following set of data records. The class attribute is Repeat_customer.

RID	Age	City	Gender	Education	Repeat_customer
101	20...30	NY	F	college	YES
102	20...30	SF	M	graduate	YES
103	31...40	NY	F	college	YES
104	51...60	NY	F	college	NO
105	31...40	LA	M	high school	NO
106	41...50	NY	F	college	YES
107	41...50	NY	F	graduate	YES
108	20...30	LA	M	college	YES
109	20...30	NY	F	high school	NO
110	20...30	NY	F	college	YES

28.20. Consider the following set of two-dimensional records:

RID	Dimension1	Dimension2
1	8	4
2	5	4
3	2	4
4	2	6
5	2	8
6	8	6

Also consider two different clustering schemes: (1) where $Cluster_1$ contains records {1,2,3} and $Cluster_2$ contains records {4,5,6} and (2) where $Cluster_1$ contains records {1,6} and $Cluster_2$ contains records {2,3,4,5}. Which scheme is better and why?

28.21. Use the *k*-Means algorithm to cluster the data from Exercise 28.20. We can use a value of 3 for *K* and we can assume that the records with RIDs 1, 3, and 5 are used for the initial cluster centroids (means).

28.22. The *k*-Means algorithm uses a similarity metric of distance between a record and a cluster centroid. If the attributes of the records are not quantitative but categorical in nature, such as Income_level with values {low, medium, high} or Married with values {Yes, No} or State_of_residence with values {Alabama, Alaska, ..., Wyoming}, then the distance metric is not meaningful. Define a more suitable similarity metric that can be used for clustering data records that contain categorical data.

Selected Bibliography

Literature on data mining comes from several fields, including statistics, mathematical optimization, machine learning, and artificial intelligence. Chen et al. (1996) give a good summary of the database perspective on data mining. The book by Han and Kamber (2001) is an excellent text, describing in detail the different algorithms and techniques used in the data mining area. Work at IBM Almaden research has produced a large number of early concepts and algorithms as well as results from some performance studies. Agrawal et al. (1993) report the first major study on association rules. Their Apriori algorithm for market basket data in Agrawal and Srikant (1994) is improved by using partitioning in Savasere et al. (1995); Toivonen (1996) proposes sampling as a way to reduce the processing effort. Cheung et al. (1996) extends the partitioning to distributed environments; Lin and Dunham (1998) propose techniques to overcome problems with data skew. Agrawal et al. (1993b) discuss the performance perspective on association rules. Mannila et al. (1994), Park et al. (1995), and Amir et al. (1997) present additional efficient algorithms related to association rules. Han et al. (2000) present the FP-tree algorithm

discussed in this chapter. Srikant and Agrawal(1995) proposes mining generalized rules. Savasere et al. (1998) present the first approach to mining negative associations. Agrawal et al. (1996) describe the Quest system at IBM. Sarawagi et al. (1998) describe an implementation where association rules are integrated with a relational database management system. Piatesky-Shapiro and Frawley (1992) have contributed papers from a wide range of topics related to knowledge discovery. Zhang et al. (1996) present the BIRCH algorithm for clustering large databases. Information about decision tree learning and the classification algorithm presented in this chapter can be found in Mitchell (1997).

Adriaans and Zantinge (1996), Fayyad et al. (1997), and Weiss and Indurkhya (1998) are books devoted to the different aspects of data mining and its use in prediction. The idea of genetic algorithms was proposed by Holland (1975); a good survey of genetic algorithms appears in Srinivas and Patnaik (1994). Neural networks have a vast literature; a comprehensive introduction is available in Lippman (1987).

Tan et al. (2006) provides a comprehensive introduction to data mining and has a detailed set of references. Readers are also advised to consult proceedings of two prominent annual conferences in data mining: the Knowledge Discovery and Data Mining Conference (KDD), which has been running since 1995, and the SIAM International Conference on Data Mining (SDM), which has been running since 2001. Links to past conferences may be found at http://dblp.uni-trier.de.