10

ASSOCIATION RULES

Chapter Objectives

- Explain the local modeling character of association-rule techniques.
- Analyze the basic characteristics of large transactional databases.
- Describe the Apriori algorithm and explain all its phases through illustrative examples.
- Compare the frequent pattern (FP) growth method with the Apriori algorithm.
- Outline the solution for association-rule generation from frequent itemsets.
- Explain the discovery of multidimensional associations.
- Introduce the extension of FP growth methodology for classification problems.

When we talk about machine-learning methods applied to data mining, we classify them as either parametric or nonparametric methods. In *parametric methods* used for density estimation, classification, or regression, we assume that a final model is valid over the entire input space. In regression, for example, when we derive a linear model, we apply it for all future inputs. In classification, we assume that all samples (training, but also new, testing) are drawn from the same density distribution. Models in these

Data Mining: Concepts, Models, Methods, and Algorithms, Second Edition. Mehmed Kantardzic.
© 2011 by Institute of Electrical and Electronics Engineers. Published 2011 by John Wiley & Sons, Inc.

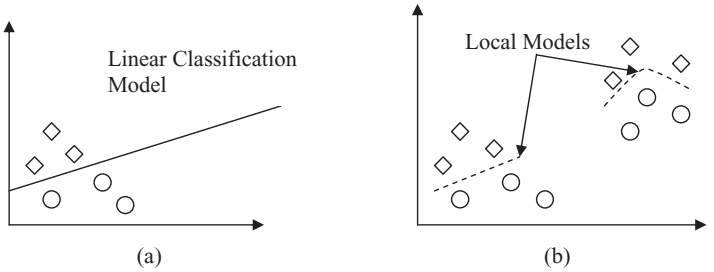

Figure 10.1. Parametric versus nonparametric methods. (a) Parametric methods build global models; (b) nonparametric methods result in local modelling.

cases are global models valid for the entire n-dimensional space of samples. The advantage of a parametric method is that it reduces the problem of modeling with only a small number of parameters. Its main disadvantage is that initial assumptions do not hold in many real-world problems, which causes a large error. In *nonparametric estimation*, all we assume is that similar inputs have similar outputs. Methods do not assume any a priori density or parametric form. There is no single global model. Local models are estimated as they occur, affected only by nearby training samples (Fig. 10.1).

The discovery of association rules is one of the major techniques of data mining, and it is perhaps the most common form of local-pattern discovery in unsupervised learning systems. It is a form of data mining that most closely resembles the process that most people think about when they try to understand the data-mining process, namely, "mining" for gold through a vast database. The gold in this case would be a rule that is interesting, that tells you something about your database that you did not already know and probably were not able to explicitly articulate. These methodologies retrieve all possible interesting patterns in the database. This is a strength in the sense that it leaves "no stone unturned." But it can be viewed also as a weakness because the user can easily become overwhelmed with a large amount of new information, and an analysis of their usability is difficult and time-consuming.

Besides the standard methodologies such as the Apriori technique for association-rule mining, we will explain some extensions such as *FP tree* and Classification Based on Multiple Association Rules (CMAR) algorithms. All these methodologies show how important and applicable the problem of market-basket analysis is and the corresponding methodologies for discovery of association rules in data.

10.1 MARKET-BASKET ANALYSIS

A market basket is a collection of items purchased by a customer in a single transaction, which is a well-defined business activity. For example, a customer's visits to a grocery store or an online purchase from a virtual store on the Web are typical customer transactions. Retailers accumulate huge collections of transactions by recording business activities over time. One common analysis run against a transactions database is to find

sets of items, or *itemsets*, that appear together in many transactions. A business can use knowledge of these patterns to improve the placement of these items in the store or the layout of mail-order catalog pages and Web pages. An itemset containing i items is called an *i-itemset*. The percentage of transactions that contain an itemset is called the itemset's *support*. For an itemset to be interesting, its support must be higher than a user-specified minimum. Such itemsets are said to be frequent.

Why is finding frequent itemsets a nontrivial problem? First, the number of customer transactions can be very large and usually will not fit in a central memory of a computer. Second, the potential number of frequent itemsets is exponential to the number of different items, although the actual number of frequent itemsets can be much smaller. Therefore, we want algorithms that are scalable (their complexity should increase linearly, not exponentially, with the number of transactions) and that examine as few infrequent itemsets as possible. Before we explain some of the more efficient algorithms, let us try to describe the problem more formally and develop its mathematical model.

From a database of sales transactions, we want to discover the important associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let DB be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. Note that the quantities of the items bought in a transaction are not considered, meaning that each item is a binary variable indicating whether an item was bought or not. Each transaction is associated with an identifier called a transaction identifier or TID. An example of the model for such a transaction database is given in Table 10.1.

Let X be a set of items. A transaction T is said to contain X if and only if $X \subseteq T$. An association rule implies the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set DB with *confidence c* if *c*% of the transactions in DB that contain X also contain Y. The rule $X \Rightarrow Y$ has *support s* in the transaction set DB if *s*% of the transactions in DB contain $X \cup Y$. Confidence denotes the strength of implication and support indicates the frequency of the patterns occurring in the rule. It is often desirable to pay attention to only those rules that may have a reasonably large support. Such rules with high confidence and strong support are referred to as *strong rules*. The task of mining association rules is essentially to discover strong association rules in large databases. The problem of mining association rules may be decomposed into two phases:

TABLE 10.1. A Model of A Simple Transaction Database

Database DB:	
TID	Items
001	A C D
002	B C E
003	A B C E
004	B E

1. Discover the large itemsets, that is, the sets of items that have transaction support *s* above a predetermined minimum threshold.
2. Use the large itemsets to generate the association rules for the database that have confidence *c* above a predetermined minimum threshold.

The overall performance of mining association rules is determined primarily by the first step. After the large itemsets are identified, the corresponding association rules can be derived in a straightforward manner. Efficient counting of large itemsets is thus the focus of most mining algorithms, and many efficient solutions have been designed to address previous criteria. The *Apriori* algorithm provided one early solution to the problem, and it will be explained in greater detail in this chapter. Other subsequent algorithms built upon the *Apriori* algorithm represent refinements of a basic solution and they are explained in a wide spectrum of articles including texts recommended in Section 10.10.

10.2 ALGORITHM APRIORI

The algorithm *Apriori* computes the frequent itemsets in the database through several iterations. Iteration *i* computes all frequent *i*-itemsets (itemsets with *i* elements). Each iteration has two steps: *candidate generation* and *candidate counting and selection*.

In the first phase of the first iteration, the generated set of candidate itemsets contains all 1-itemsets (i.e., all items in the database). In the counting phase, the algorithm counts their support by searching again through the whole database. Finally, only 1-itemsets (items) with s above required threshold will be selected as frequent. Thus, after the first iteration, all frequent 1-itemsets will be known.

What are the itemsets generated in the second iteration? In other words, how does one generate 2-itemset candidates? Basically, all pairs of items are candidates. Based on knowledge about infrequent itemsets obtained from previous iterations, the *Apriori* algorithm reduces the set of candidate itemsets by pruning—*a priori*—those candidate itemsets that cannot be frequent. The pruning is based on the observation that if an itemset is frequent all its subsets could be frequent as well. Therefore, before entering the candidate-counting step, the algorithm discards every candidate itemset that has an infrequent subset.

Consider the database in Table 10.1. Assume that the minimum support s = 50%, so an itemset is frequent if it is contained in at least 50% of the transactions—in our example, in two out of every four transactions in the database. In each iteration, the *Apriori* algorithm constructs a candidate set of large itemsets, counts the number of occurrences of each candidate, and then determines large itemsets based on the predetermined minimum support s = 50%.

In the first step of the first iteration, all single items are candidates. *Apriori* simply scans all the transactions in a database DB and generates a list of candidates. In the next step, the algorithm counts the occurrences of each candidate and based on threshold s selects frequent itemsets. All these steps are given in Figure 10.2. Five 1-itemsets are generated in C_1 and, of these, only four are selected as large in L_1 because their support is greater than or equal to two, or $s \geq 50\%$.

1-itemsets $\mathbf{C_1}$
{A}
{C}
{D}
{B}
{E}

(a)

1-itemsets	Count	s[%]
{A}	2	50
{C}	3	75
{D}	1	25
{B}	3	75
{E}	3	75

(b1)

Large 1-itemsets $\mathbf{L_1}$	Count	s[%]
{A}	2	50
{C}	3	75
{B}	3	75
{E}	3	75

(b2)

Figure 10.2. First iteration of the *Apriori* algorithm for a database DB. (a) Generate phase; (b1) count phase; (b2) select phase.

2-itemsets $\mathbf{C_2}$
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

(a)

2-itemsets	Count	s[%]
{A, B}	1	25
{A, C}	2	50
{A, E}	1	25
{B, C}	2	50
{B, E}	3	75
{C, E}	2	50

(b1)

Large 2-itemsets $\mathbf{L_2}$	Count	s[%]
{A, C}	2	50
{B, C}	2	50
{B, E}	3	75
{C, E}	2	50

(b2)

Figure 10.3. Second iteration of the *Apriori* algorithm for a database DB. (a) Generate phase; (b1) count phase; (b2) select phase

To discover the set of large 2-itemsets, because any subset of a large itemset could also have minimum support, the *Apriori* algorithm uses $L_1 * L_1$ to generate the candidates. The operation * is defined in general as

$$L_k * L_k = \{X \cup Y \text{ where } X, Y \in L_k,\ |X \cap Y| = k-1\}$$

For k = 1 the operation represents a simple concatenation. Therefore, C_2 consists of 2-itemsets generated by the operation$|L_1| \cdot (|L_1|-1)/2$ as candidates in the second iteration. In our example, this number is $4 \cdot 3/2 = 6$. Scanning the database DB with this list, the algorithm counts the support for every candidate and in the end selects a large 2-itemsets L_2 for which $s \geq 50\%$. All these steps and the corresponding results of the second iteration are given in Figure 10.3.

The set of candidate itemset C_3 is generated from L_2 using the previously defined operation $L_2 * L_2$. Practically, from L_2, two large 2-itemsets with the same first item, such as {B, C} and {B, E}, are identified first. Then, *Apriori* tests whether the 2-itemset {C, E}, which consists of the second items in the sets {B, C} and {B, E}, constitutes a large 2-itemset or not. Because {C, E} is a large itemset by itself, we know that all the subsets of {B, C, E} are large, and then {B, C, E} becomes a candidate 3-itemset. There is no other candidate 3-itemset from L_2 in our database DB. *Apriori* then scans all the transactions and discovers the large 3-itemsets L_3, as shown in Figure 10.4.

3-itemsets $\mathbf{C_3}$
{B, C, E}

(a)

3-itemsets	Count	s[%]
{B, C, E}	2	50

(b1)

Large 3-itemsets $\mathbf{L_3}$	Count	s[%]
{B, C, E}	2	50

(b2)

Figure 10.4. Third iteration of the *Apriori* algorithm for a database DB. (a) Generate phase; (b1) count phase; (b2) select phase

In our example, because there is no candidate 4-itemset to be constituted from L_3, *Apriori* ends the iterative process.

Apriori counts not only the support of all frequent itemsets, but also the support of those infrequent candidate itemsets that could not be eliminated during the pruning phase. The set of all candidate itemsets that are infrequent but whose support is counted by *Apriori* is called the *negative border*. Thus, an itemset is in the negative border if it is infrequent, but all its subsets are frequent. In our example, analyzing Figures 10.2 and 10.3, we can see that the negative border consists of itemsets {D}, {A, B}, and {A, E}. The negative border is especially important for some improvements in the *Apriori* algorithm such as increased efficiency in the generation of large itemsets.

10.3 FROM FREQUENT ITEMSETS TO ASSOCIATION RULES

The second phase in discovering association rules based on all frequent i-itemsets, which have been found in the first phase using the *Apriori* or some other similar algorithm, is relatively simple and straightforward. For a rule that implies $\{x_1,x_2, x_3\} \rightarrow x_4$, it is necessary that both itemset $\{x_1, x_2, x_3, x_4\}$ and $\{x_1, x_2, x_3\}$ are frequent. Then, the confidence c of the rule is computed as the quotient of supports for the itemsets $c = s(x_1,x_2,x_3,x_4)/s(x_1,x_2,x_3)$. Strong association rules are rules with a confidence value c above a given threshold.

For our example of database DB in Table 10.1, if we want to check whether the association rule $\{B, C\} \rightarrow E$ is a strong rule, first we select the corresponding supports from tables L_2 and L_3:

$$s(B, C) = 2, s(B, C, E) = 2$$

and using these supports we compute the confidence of the rule:

$$c(\{B, C\} \rightarrow E) = s(B, C, E) / s(B, C) = 2 / 2 = 1 \text{ (or } 100\%)$$

Whatever the selected threshold for strong association rules is (e.g., $c_T = 0.8$ or 80%), this rule will pass because its confidence is maximal, that is, if a transaction contains items B and C, it will also contain item E. Other rules are also possible for our database DB, such as $A \rightarrow C$ because $c(A \rightarrow C) = s(A, C)/s(A) = 1$, and both itemsets {A} and {A, C} are frequent based on the *Apriori* algorithm. Therefore, in this phase, it is necessary only to systematically analyze all possible association rules that

could be generated from the frequent itemsets, and select as strong association rules those that have a confidence value above a given threshold.

Notice that not all the discovered strong association rules (i.e., passing the required support *s* and required confidence *c*) are interesting enough to be presented and used. For example, consider the following case of mining the survey results in a school of 5000 students. A retailer of breakfast cereal surveys the activities that the students engage in every morning. The data show that 60% of the students (i.e., 3000 students) play basketball; 75% of the students (i.e., 3750 students) eat cereal; and 40% of them (i.e., 2000 students) play basketball and also eat cereal. Suppose that a data-mining program for discovering association rules is run on the following settings: the minimal support is 2000 (s = 0.4) and the minimal confidence is 60% (c = 0.6). The following association rule will be produced: "(play basketball) $\rightarrow$ (eat cereal)," since this rule contains the minimal student support and the corresponding confidence c = 2000/3000 = 0.66 is larger than the threshold value. However, the above association rule is misleading since the overall percentage of students eating cereal is 75%, larger than 66%. That is, playing basketball and eating cereal are in fact negatively associated. Being involved in one itemset decreases the likelihood of being involved in the other. Without fully understanding this aspect, one could make wrong business or scientific decisions from the association rules derived.

To filter out such misleading associations, one may define that an association rule $A \rightarrow B$ is *interesting* if its confidence exceeds a certain measure. The simple argument we used in the example above suggests that the right heuristic to measure association should be

$$s(A, B) / s(A) - s(B) > d$$

or alternatively:

$$s(A, B) - s(A) \cdot s(B) > k$$

where d or k are suitable constants. The expressions above essentially represent tests of statistical independence. Clearly, the factor of statistical dependence among analyzed itemsets has to be taken into consideration to determine the usefulness of association rules. In our simple example with students this test fails for the discovered association rule

$$s(A, B) - s(A) \cdot s(B) = 0.4 - 0.6 \cdot 0.75 = -0.05 < 0$$

and, therefore, despite high values for parameters s and c, the rule is not interesting. In this case, it is even misleading.

10.4 IMPROVING THE EFFICIENCY OF THE APRIORI ALGORITHM

Since the amount of the processed data in mining frequent itemsets tends to be huge, it is important to devise efficient algorithms to mine such data. Our basic *Apriori*

algorithm scans the database several times, depending on the size of the largest frequent itemset. Since Apriori algorithm was first introduced and as experience has accumulated, there have been many attempts to devise more efficient algorithms of frequent itemset mining including approaches such as hash-based technique, partitioning, sampling, and using vertical data format. Several refinements have been proposed that focus on reducing the number of database scans, the number of candidate itemsets counted in each scan, or both.

Partition-based Apriori is an algorithm that requires only two scans of the transaction database. The database is divided into disjoint partitions, each small enough to fit into available memory. In a first scan, the algorithm reads each partition and computes locally frequent itemsets on each partition. In the second scan, the algorithm counts the support of all locally frequent itemsets toward the complete database. If an itemset is frequent with respect to the complete database, it must be frequent in at least one partition. That is the heuristics used in the algorithm. Therefore, the second scan through the database counts itemset's frequency only for a union of all locally frequent itemsets. This second scan directly determines all frequent itemsets in the database as a subset of a previously defined union.

In some applications, the transaction database has to be mined frequently to capture customer behavior. In such applications, the efficiency of data mining could be a more important factor than the complete accuracy of the results. In addition, in some applications the problem domain may be vaguely defined. Missing some marginal cases that have confidence and support levels at the borderline may have little effect on the quality of the solution to the original problem. Allowing imprecise results can in fact significantly improve the efficiency of the applied mining algorithm.

As the database size increases, *sampling* appears to be an attractive approach to data mining. A sampling-based algorithm typically requires two scans of the database. The algorithm first takes a sample from the database and generates a set of candidate itemsets that are highly likely to be frequent in the complete database. In a subsequent scan over the database, the algorithm counts these itemsets' exact support and the support of their negative border. If no itemset in the negative border is frequent, then the algorithm has discovered all frequent itemsets. Otherwise, some superset of an itemset in the negative border could be frequent, but its support has not yet been counted. The sampling algorithm generates and counts all such potentially frequent itemsets in subsequent database scans.

Because it is costly to find frequent itemsets in large databases, *incremental updating* techniques should be developed to maintain the discovered frequent itemsets (and corresponding association rules) so as to avoid mining the whole updated database again. Updates on the database may not only invalidate some existing frequent itemsets but also turn some new itemsets into frequent ones. Therefore, the problem of maintaining previously discovered frequent itemsets in large and dynamic databases is nontrivial. The idea is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to substantially reduce the pool of candidates to be reexamined.

In many applications, interesting associations among data items often occur at a relatively high *concept level*. For example, one possible hierarchy of food components

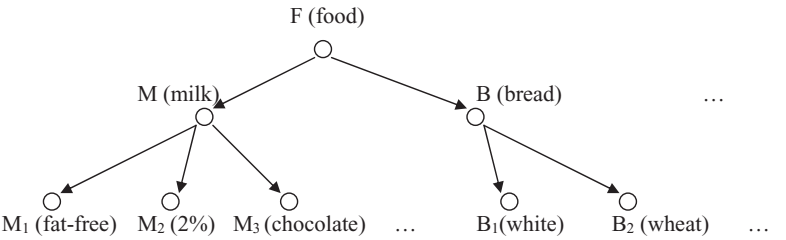

Figure 10.5. An example of concept hierarchy for mining multiple-level frequent itemsets.

is presented in Figure 10.5, where M (milk) and B (bread), as concepts in the hierarchy, may have several elementary sub-concepts. The lowest level elements in the hierarchy (M_1, M_2, . . . , B_1, B_2, . . .) are types of milk and bread defined with their bar-codes in the store. The purchase patterns in a transaction database may not show any substantial regularities at the elementary data level, such as at the bar-code level (M_1, M_2, M_3, B_1, B_2, . . .), but may show some interesting regularities at some high concept level(s), such as milk M and bread B.

Consider the class hierarchy in Figure 10.5. It could be difficult to find high support for purchase patterns at the primitive-concept level, such as chocolate milk and wheat bread. However, it would be easy to find in many databases that more than 80% of customers who purchase milk may also purchase bread. Therefore, it is important to mine frequent itemsets at a generalized abstraction level or at multiple-concept levels; these requirements are supported by the *Apriori* generalized-data structure.

One extension of the *Apriori* algorithm considers an *is-a* hierarchy on database items, where information about multiple abstraction levels already exists in the database organization. An *is-a* hierarchy defines which items are a specialization or generalization of other items. The extended problem is to compute frequent itemsets that include items from different hierarchy levels. The presence of a hierarchy modifies the notation of when an item is contained in a transaction. In addition to the items listed explicitly, the transaction contains their ancestors in the taxonomy. This allows the detection of relationships involving higher hierarchy levels, since an itemset's support can increase if an item is replaced by one of its ancestors.

10.5 FP GROWTH METHOD

Let us define one of the most important problems with scalability of the *Apriori* algorithm. To generate one FP of length 100, such as $\{a_1, a_2, \ldots, a_{100}\}$, the number of candidates that has to be generated will be at least

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

and it will require hundreds of database scans. The complexity of the computation increases exponentially. That is only one of the many factors that influence the development of several new algorithms for association-rule mining.

TABLE 10.2. The Transactional Database T

TID	Itemset
01	f, a, c, d, g, i, m, p
02	a, b, c, f, l, m, o
03	b, f, h, j, o
04	b, c, k, s, p
05	a, f, c, e, l, p, m, n

FP growth method is an efficient way of mining frequent itemsets in large databases. The algorithm mines frequent itemsets without the time-consuming candidate-generation process that is essential for *Apriori*. When the database is large, FP growth first performs a database projection of the frequent items; it then switches to mining the main memory by constructing a compact data structure called the FP tree. For an explanation of the algorithm, we will use the transactional database in Table 10.2 and the minimum support threshold of 3.

First, a scan of the database T derives a list L of frequent items occurring three or more than three times in the database. These are the items (with their supports):

$$L = \{(f,4), (c,4), (a,3), (b,3), (m,3), (p,3)\}.$$

The items are listed in descending order of frequency. This ordering is important since each path of the FP tree will follow this order.

Second, the root of the tree, labeled ROOT, is created. The database T is scanned a second time. The scan of the first transaction leads to the construction of the first branch of the FP tree: {(f,1), (c,1), (a,1), (m,1), (p,1)}. Only those items that are in the list of frequent items L are selected. The indices for nodes in the branch (all are 1) represent the cumulative number of samples at this node in the tree, and of course, after the first sample, all are 1. The order of the nodes is not as in the sample but as in the list of frequent items L. For the second transaction, because it shares items f, c, and a it shares the prefix {f, c, a} with the previous branch and extends to the new branch {(f, 2), (c, 2), (a, 2), (m, 1), (p, 1)}, increasing the indices for the common prefix by one. The new intermediate version of the FP tree, after two samples from the database, is given in Figure 10.6a. The remaining transactions can be inserted similarly, and the final FP tree is given in Figure 10.6b.

To facilitate tree traversal, an *item header table* is built, in which each item in list L connects nodes in the FP tree with its values through node links. All f nodes are connected in one list, all c nodes in the other, and so on. For simplicity of representation only the list for b nodes is given in Figure 10.6b. Using the compact-tree structure, the FP growth algorithm mines the complete set of frequent itemsets.

According to the list L of frequent items, the complete set of frequent itemsets can be divided into subsets (six for our example) without overlap: (1) frequent itemsets having item p (the end of list L); (2) the itemsets having item m but not p; (3) the frequent itemsets with b and without both m and p; . . . ; and (6) the large itemsets only

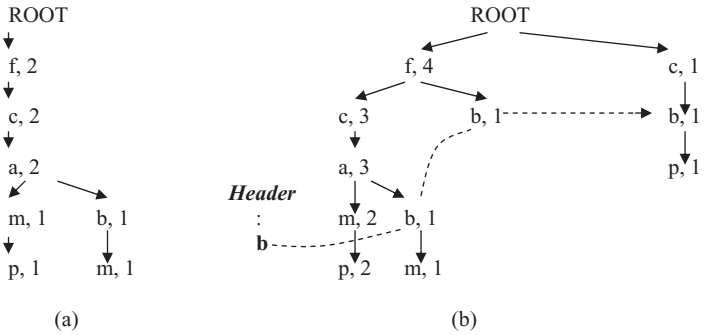

Figure 10.6. FP tree for the database T in Table 10.2. (a) FP tree after two samples; (b) final FT tree.

with f. This classification is valid for our example, but the same principles can be applied to other databases and other L lists.

Based on node-link connection, we collect all the transactions that p participates in by starting from the header table of p and following p's node links. In our example, two paths will be selected in the FP tree: {(f,4), (c,3), (a,3), (m,2), (p,2)} and {(c,1), (b,1), (p,1)}, where samples with a frequent item p are {(f,2), (c,2), (a,2), (m,2), (p,2)}, and {(c,1), (b,1), (p,1)}. The given threshold value (3) satisfies only the frequent itemsets *{(c,3), (p,3)},* or the simplified *{c, p}*. All other itemsets with p are below the threshold value.

The next subsets of frequent itemsets are those with m and without p. The FP tree recognizes the paths {(f,4), (c,3), (a,3), (m,2)}, and {(f,4), (c,3), (a,3), (b,1), (m,1)}, or the corresponding accumulated samples {(f,2), (c,2), (a,2), (m,2)}, and {(f,1), (c,1), (a,1), (b,1), (m,1)}. Analyzing the samples we discover the frequent itemset *{(f,3), (c,3), (a,3), (m,3)}* or, simplified, *{f, c, a, m}*.

Repeating the same process for subsets (3) to (6) in our example, additional frequent itemsets could be mined. These are itemsets *{f, c, a}* and *{f, c}*, but they are already subsets of the frequent itemset *{f, c, a, m}*. Therefore, the final solution in the FP growth method is the set of frequent itemsets, which is, in our example, *{{c, p}, {f, c, a, m}}*.

Experiments have shown that the FP growth algorithm is faster than the *Apriori* algorithm by about one order of magnitude. Several optimization techniques are added to the FP growth algorithm, and there exists its versions for mining sequences and patterns under constraints.

10.6 ASSOCIATIVE-CLASSIFICATION METHOD

CMAR is a classification method adopted from the FP growth method for generation of frequent itemsets. The main reason we included CMAR methodology in this chapter is its FP growth roots, but there is the possibility of comparing CMAR accuracy and efficiency with the C4.5 methodology.

Suppose data samples are given with n attributes ($A_1, A_2, \ldots, A_n$). Attributes can be categorical or continuous. For a continuous attribute, we assume that its values are discretized into intervals in the preprocessing phase. A training data set T is a set of samples such that for each sample there exists a class label associated with it. Let $C = \{c_1, c_2, \ldots, c_m\}$ be a finite set of *class labels*.

In general, a pattern $P = \{a_1, a_2, \ldots, a_k\}$ is a set of attribute values for different attributes ($1 \le k \le n$). A sample is said to match the pattern P if it has all the attribute values given in the pattern. For rule R: $P \rightarrow c$, the number of data samples matching pattern P and having class label c is called the *support* of rule R, denoted *sup(R)*. The ratio of the number of samples matching pattern P and having class label c versus the total number of samples matching pattern P is called the *confidence* of R, denoted as *conf(R)*. The association-classification method (CMAR) consists of two phases:

1. rule generation or training, and
2. classification or testing.

In the first, rule generation phase, CMAR computes the complete set of rules in the form R: $P \rightarrow c$, such that sup(R) and conf(R) pass the given thresholds. For a given support threshold and confidence threshold, the associative-classification method finds the complete set of class-association rules (CAR) passing the thresholds. In the testing phase, when a new (unclassified) sample comes, the classifier, represented by a set of association rules, selects the rule that matches the sample and has the highest confidence, and uses it to predict the classification of the new sample.

We will illustrate the basic steps of the algorithm through one simple example. Suppose that for a given training data set T, as shown in Table 10.3, the support threshold is 2 and the confidence threshold is 70%.

First, CMAR scans the training data set and finds the set of attribute values occurring beyond the threshold support (at least twice in our database). One simple approach is to sort each attribute and to find all frequent values. For our database T, this is a set $F = \{a_1, b_2, c_1, d_3\}$, and it is called a frequent itemset. All other attribute values fail the support threshold. Then, CMAR sorts attribute values in F, in support-descending order, that is, F-list = (a_1, b_2, c_1, d_3).

Now, CMAR scans the training data set again to construct an FP tree. The FP tree is a prefix tree with respect to the F-list. For each sample in a training data set, attributes

TABLE 10.3. Training Database T for the CMAR Algorithm

ID	A	B	C	D	Class
01	a_1	b_1	c_1	d_1	A
02	a_1	b_2	c_1	d_2	B
03	a_2	b_3	c_2	d_3	A
04	a_1	b_2	c_3	d_3	C
05	a_1	b_2	c_1	d_3	C

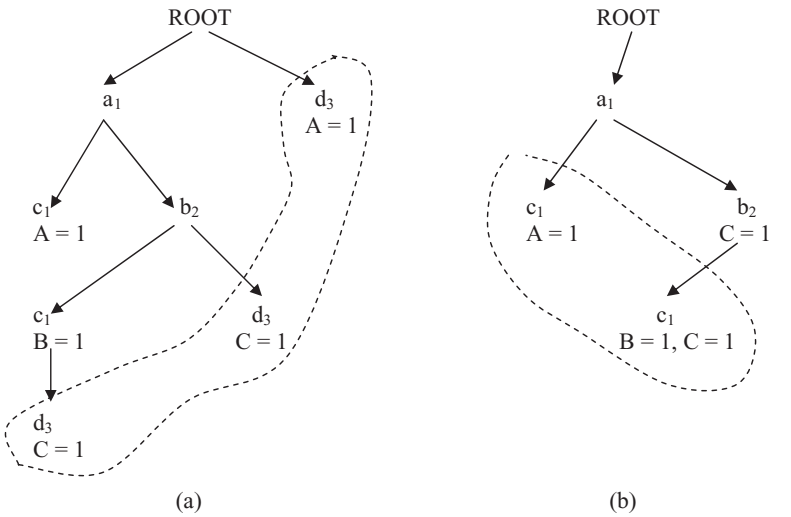

Figure 10.7. FP tree for the database in Table 10.3. (a) nonmerged FT tree; (b) FT tree after merging d_3 nodes.

values appearing in the F-list are extracted and sorted according to the order in the F-list. For example, for the first sample in database T, (a_1, c_1) are extracted and inserted in the tree as the left-most branch in the tree. The class label of the sample and the corresponding counter are attached to the last node in the path.

Samples in the training data set share prefixes. For example, the second sample carries attribute values (a_1, b_2, c_1) in the F-list and shares a common prefix a_1 with the first sample. An additional branch from the node a_1 will be inserted in the tree with new nodes b_2 and c_1. A new class label B with the count equal to 1 is also inserted at the end of the new path. The final FP tree for the database T is given in Figure 10.7a.

After analyzing all the samples and constructing an FP tree, the set of CAR can be generated by dividing all rules into subsets without overlap. In our example it will be four subsets: (1) the rules having d_3 value; (2) the rules having c_1 but no d_3; (3) the rules having b_2 but neither d_3 nor c_1; and (4) the rules having only a_1. CMAR finds these subsets one by one.

To find the subset of rules having d_3, CMAR traverses nodes having the attribute value d_3 and looks "upward" the FP tree to collect d_3-projected samples. In our example, there are three samples represented in the FP tree, and they are (a_1, b_2, c_1, d_3):C, (a_1, b_2, d_3):C, and (d_3):A. The problem of finding all FPs in the training set can be reduced to mining FPs in the d_3-projected database. In our example, in the d_3-projected database, since the pattern (a_1, b_2, d_3) occurs twice its support is equal to the required threshold value 2. Also, the rule based on this FP, (a_1, b_2, d_3) $\rightarrow$ C has a confidence of 100% (above the threshold value), and that is the only rule generated in the given projection of the database.

After a search for rules having d_3 value, all the nodes of d_3 and their corresponding class labels are merged into their parent nodes at the FP tree. The FP tree is shrunk as

shown in Figure 10.7b. The remaining set of rules can be mined similarly by repeating the previous procedures for a c_1-projected database, then for the b_2-projected database, and finally for the a_1-projected database. In this analysis, (a_1, c_1) is an FP with support 3, but all rules are with confidence less than the threshold value. The same conclusions can be drawn for pattern (a_1, b_2) and for (a_1). Therefore, the only association rule generated through the training process with the database T is (a_1, b_2, d_3)$\rightarrow$C with support equal to 2 and 100% confidence.

When a set of rules is selected for classification, CMAR is ready to classify new samples. For the new sample, CMAR collects the subset of rules matching the sample from the total set of rules. Trivially, if all the rules have the same class, CMAR simply assigns that label to the new sample. If the rules are not consistent in the class label, CMAR divides the rules into groups according to the class label and yields the label of the "strongest" group. To compare the strength of groups, it is necessary to measure the "combined effect" of each group. Intuitively, if the rules in a group are highly positively correlated and have good support, the group should have a strong effect. CMAR uses the strongest rule in the group as its representative, that is, the rule with highest χ^2 test value (adopted for this algorithm for a simplified computation). Preliminary experiments have shown that CMAR outperforms the C4.5 algorithm in terms of average accuracy, efficiency, and scalability.

10.7 MULTIDIMENSIONAL ASSOCIATION–RULES MINING

A multidimensional transactional database DB has the schema

$$(\text{ID}, A_1, A_2, \ldots, A_n, \text{items})$$

where ID is a unique identification of each transaction, A_i are structured attributes in the database, and items are sets of items connected with the given transaction. The information in each tuple t = (id, a_1, a_2, . . . , a_n, items-t) can be partitioned into two: dimensional part (a_1, a_2, . . . , a_n) and itemset part (items-t). It is commonsense to divide the mining process into two steps: first mine patterns about dimensional information and then find frequent itemsets from the projected sub-database, or vice versa. Without any preferences in the methodology we will illustrate the first approach using the multidimensional database DB in Table 10.4.

TABLE 10.4. Multidimensional-Transactional Database DB

ID	A_1	A_2	A_3	Items
01	a	1	m	x, y, z
02	b	2	n	z, w
03	a	2	m	x, z, w
04	c	3	p	x, w

One can first find the frequent multidimensional-value combinations and then find the corresponding frequent itemsets of a database. Suppose that the threshold value for our database DB in Table 10.4 is set to 2. Then, the combination of attribute values that occurs two or more than two times is frequent, and it is called a multidimensional pattern or MD pattern. For mining MD patterns, a modified Bottom Up Computation (BUC) algorithm can be used (it is an efficient "iceberg cube" computing algorithm). The basic steps of the BUC algorithm are as follows:

1. First, sort all tuples in the database in alphabetical order of values in the first dimension (A_1), because the values for A_1 are categorical. The only MD pattern found for this dimension is (a, *, *) because only the value *a* occurs two times; the other values b and c occur only once and they are not part of the MD patterns. Value * for the other two dimensions shows that they are not relevant in this first step, and they could have any combination of allowed values.

   Select tuples in a database with found M pattern (or patterns). In our database, these are the samples with ID values 01 and 03. Sort the reduced database again with respect to the second dimension (A_2), where the values are 1 and 2. Since no pattern occurs twice, there are no MD patterns for exact A_1 and A_2 values. Therefore, one can ignore the second dimension A_2 (this dimension does not reduce the database further). All selected tuples are used in the next phase.

   Selected tuples in the database are sorted in alphabetical order of values for the third dimension (in our example A_3 with categorical values). A subgroup (a, *, m) is contained in two tuples and it is an MD pattern. Since there are no more dimensions in our example, the search continues with the second step.
2. Repeat the processes in step 1; only start not with the first but with the second dimension (first dimension is not analyzed at all in this iteration). In the following iterations, reduce the search process further for one additional dimension at the beginning. Continue with other dimensions.

In our example in the second iteration, starting with attribute A_2, MD pattern (*, 2, *) will be found**.** Including dimension A_3, there are no additional MD patterns. The third and last iteration in our example starts with the A_3 dimension and the corresponding pattern is (*, *, m).

In summary, the modified BUC algorithm defines a set of MD patterns with the corresponding projections of a database. The processing tree for our example of database DB is shown in Figure 10.8. Similar trees will be generated for a larger number of dimensions.

When all MD patterns are found, the next step in the analysis of multidimensional-transactional database is the mining of frequent itemsets in the MD-projected database for each MD pattern. An alternative approach is based on finding frequent itemsets first and then the corresponding MD patterns.

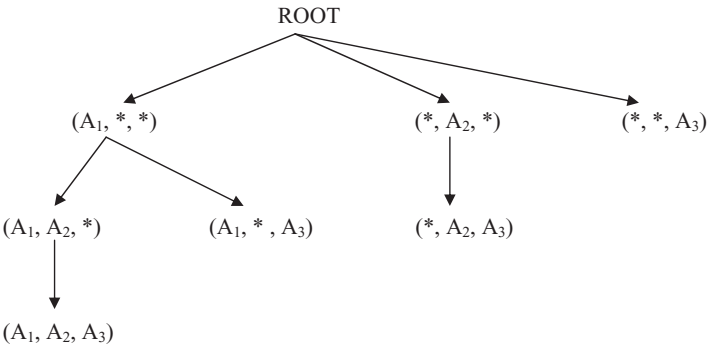

Figure 10.8. A processing tree using the BUC algorithm for the database in Table 10.4.

10.8 REVIEW QUESTIONS AND PROBLEMS

1. What is the essential difference between association rules and decision rules (described in Chapter 6)?
2. What are the typical industries in which market-basket analysis plays an important role in the strategic decision-making processes?
3. What are the common values for support and confidence parameters in the *Apriori* algorithm? Explain using the retail industry as an example.
4. Why is the process of discovering association rules relatively simple compared with generating large itemsets in transactional databases?
5. Given a simple transactional database X:

X:	TID	Items
	T01	A, B, C, D
	T02	A, C, D, F
	T03	C, D, E, G, A
	T04	A, D, F, B
	T05	B, C, G
	T06	D, F, G
	T07	A, B, G
	T08	C, D, F, G

Using the threshold values support = 25% and confidence = 60%,

(a) find all large itemsets in database X;
(b) find strong association rules for database X;
(c) analyze misleading associations for the rule set obtained in (b).

6. Given a transactional database Y:

Y:	TID	Items
	T01	A1, B1, C2
	T02	A2, C1, D1
	T03	B2, C2, E2
	T04	B1, C1, E1
	T05	A3, C3, E2
	T06	C1, D2, E2

Using the threshold values for support s = 30% and confidence c = 60%,

(a) find all large itemsets in database Y;
(b) if itemsets are organized in a hierarchy so that A = {A1, A2, A3}, B = {B1, B2}, C = {C1, C2, C3}, D = {D1, D2}, and E = {E1, E2}, find large itemsets that are defined on the conceptual level including a hierarchy of items;
(c) find strong association rules for large itemsets in (b).

7. Implement the *Apriori* algorithm and discover large itemsets in transactional database.

8. Search the Web to find the basic characteristics of publicly available or commercial software tools for association-rule discovery. Document the results of your search.

9. Given a simple transactional database, find FP tree for this database if

(a) support threshold is 5;
(b) support threshold is 3.

TID	Items
1	a b c d
2	a c d f
3	c d e g a
4	a d f b
5	b c g
6	d f g
7	a b g
8	c d f g

10. Given a simple transaction database:

TID	Items
1	X Z V
2	X Y U
3	Y Z V
4	Z V W

Using two iterations of the *Apriori* algorithm find large 2-itemsets if required support is $s \geq 50\%$. Show all steps of the algorithm.

11. Given a frequent itemset A,B,C,D, and E, how many possible association rules exist?

12. What are the frequent itemsets with a minimum support of 3 for the given set of transactions?

TID	Items
101	A,B,C,D,E
102	A,C,D
103	D,E
104	B,C,E
105	A,B,D,E
106	A,B
107	B,D,E
108	A,B,D
109	A,D
110	D,E

13. The *conviction* is a measure for an analysis of a quality of association rules. The formula for conviction CV in terms of probabilities is given as:

$$CV(A \rightarrow B) = (P[A] - P[B']) / P(A, B')$$

or in terms of support and confidence of an association rule:

$$CV(A \rightarrow B) = (1\text{-sup}[B]) / (1\text{-conf}[A \rightarrow B])$$

What are basic characteristics of the *conviction* measure? Explain the meaning of some characteristic values.

14. Consider the dataset given in the table below.

Customer ID	Transaction Id	Items
418	234145	{X, Z}
345	543789	{U, V, W, X, Y, Z}
323	965157	{U, W, Y}
418	489651	{V, X, Z}
567	748965	{U, Y}
567	325687	{W, X, Y}
323	147895	{X, Y, Z}
635	617851	{U, Z}
345	824697	{V, Y}
635	102458	{V, W, X}

(a) Compute the support for item sets {Y}, {X, Z} and {X, Y, Z} by treating each transaction ID as a market basket.
(b) Use the results from part (a) to compute the confidence for rules $XZ \rightarrow Y$ and $Y \rightarrow XZ$.
(c) Repeat part (a) by treating each customer ID as a market basket. Each item should be treated as a binary variable (1 if an item appears in at least one transaction bought by the customer, and 0 otherwise).
(d) Use the results from part (c) to compute the confidence for rules $XZ \rightarrow Y$ and $Y \rightarrow XZ$.
(e) Find FP-tree for this database if support threshold is 5.

15. A collection of market-basket data has 100,000 frequent items, and 1,000,000 infrequent items. Each pair of frequent items appears 100 times; each pair consisting of one frequent and one infrequent item appears 10 times, and each pair of infrequent items appears once. Answer each of the following questions. Your answers only have to be correct to within 1%, and for convenience, you may optionally use scientific notation, for example, 3.14×10^8 instead of 314,000,000.

(a) What is the total number of pair occurrences? That is, what is the sum of the counts of all pairs?
(b) We did not state the support threshold, but the given information lets us put bounds on the support threshold s. What are the tightest upper and lower bounds on s?

10.9 REFERENCES FOR FURTHER STUDY

Adamo, J., *Data Mining for Association Rules and Sequential Patterns*, Springer, Berlin, 2001.

This book presents a collection of algorithms for data mining on the lattice structure of the feature space. Given the computational complexity and time requirements of mining association rules and sequential patterns, the design of efficient algorithms is critical. Most algorithms provided in the book are designed for both sequential and parallel execution, and they support sophisticated data mining of large-scale transactional databases.

Cheng J., Y. Ke, W. Ng, A Survey on Algorithms for Mining Frequent Itemsets Over Data Streams, *Knowledge and Information Systems*, Vol. 16, No. 1, 2008, pp.1–27.

The increasing prominence of data streams arising in a wide range of advanced applications such as fraud detection and trend learning has led to the study of online mining of frequent itemsets. Unlike mining static databases, mining data streams poses many new challenges. In addition to the one-scan nature, the unbounded memory requirement and the high data arrival rate of data streams, the combinatorial explosion of itemsets exacerbates the mining task. The high complexity of the frequent itemset mining problem hinders the application of the stream-mining techniques. We recognize that a critical review of existing techniques is needed in order to design and develop efficient mining algorithms and data structures that are able to match the processing rate of the mining with the high arrival rate of data streams. Within a unifying set of notations and terminologies, we describe in this paper the efforts and main techniques for mining data streams and present a comprehensive survey of a number of the state-of-the-art algorithms on mining frequent itemsets over data streams.

Goethals B., Frequent Set Mining, in *Data Mining and Knowledge Discovery Handbook*, Maimon L., Rokach L., ed., Springer, New York, 2005, pp. 377–397.

Frequent sets lie at the basis of many data-mining algorithms. As a result, hundreds of algorithms have been proposed in order to solve the frequent set mining problem. In this chapter,

we attempt to survey the most successful algorithms and techniques that try to solve this problem efficiently. During the first 10 years after the proposal of the frequent set mining problem, several hundreds of scientific papers were written on the topic and it seems that this trend is keeping its pace.

Han, J., M. Kamber, *Data Mining: Concepts and Techniques*, 2nd edition, Morgan Kaufmann, San Francisco, 2006.

vThis book gives a sound understanding of data-mining principles. The primary orientation of the book is for database practitioners and professionals with emphasis on OLAP and data warehousing. In-depth analysis of association rules and clustering algorithms is the additional strength of the book. All algorithms are presented in easily understood pseudo-code and they are suitable for use in real-world, large-scale data-mining projects including advanced applications such as Web mining and text mining.