6 Mining Frequent Patterns, Associations, and Correlations: Basic Concepts and Methods

Imagine that you are a sales manager at *AllElectronics*, and you are talking to a customer who recently bought a PC and a digital camera from the store. What should you recommend to her next? Information about which products are frequently purchased by your customers following their purchases of a PC and a digital camera in sequence would be very helpful in making your recommendation. Frequent patterns and association rules are the knowledge that you want to mine in such a scenario.

Frequent patterns are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a *(frequent) sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a *(frequent) structured pattern*. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

In this chapter, we introduce the basic concepts of frequent patterns, associations, and correlations (Section 6.1) and study how they can be mined efficiently (Section 6.2). We also discuss how to judge whether the patterns found are interesting (Section 6.3). In Chapter 7, we extend our discussion to advanced methods of frequent pattern mining, which mine more complex forms of frequent patterns and consider user preferences or constraints to speed up the mining process.

6.1 Basic Concepts

Frequent pattern mining searches for recurring relationships in a given data set. This section introduces the basic concepts of frequent pattern mining for the discovery of interesting associations and correlations between itemsets in transactional and relational databases. We begin in Section 6.1.1 by presenting an example of market basket analysis, the earliest form of frequent pattern mining for association rules. The basic concepts of mining frequent patterns and associations are given in Section 6.1.2.

6.1.1 Market Basket Analysis: A Motivating Example

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes such as catalog design, cross-marketing, and customer shopping behavior analysis.

A typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets" (Figure 6.1). The discovery of these associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip

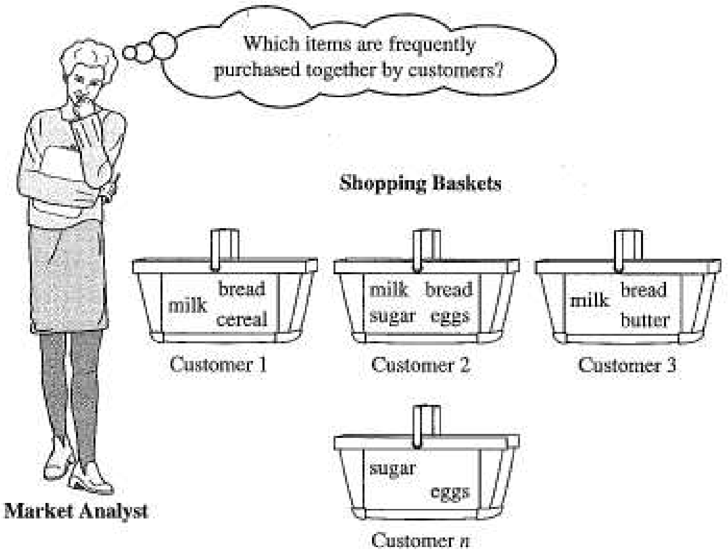


Figure 6.1 Market basket analysis.

to the supermarket? This information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

Let's look at an example of how market basket analysis can be useful.

Example 6.1 **Market basket analysis.** Suppose, as manager of an *AllElectronics* branch, you would like to learn more about the buying habits of your customers. Specifically, you wonder, *"Which groups or sets of items are customers likely to purchase on a given trip to the store?"* To answer your question, market basket analysis may be performed on the retail data of customer transactions at your store. You can then use the results to plan marketing or advertising strategies, or in the design of a new catalog. For instance, market basket analysis may help you design different store layouts. In one strategy, items that are frequently purchased together can be placed in proximity to further encourage the combined sale of such items. If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of both items.

In an alternative strategy, placing hardware and software at opposite ends of the store may entice customers who purchase such items to pick up other items along the way. For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading toward the software display to purchase antivirus software, and may decide to purchase a home security system as well. Market basket analysis can also help retailers plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers *as well as* computers. ∎

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently *associated* or purchased together. These patterns can be represented in the form of **association rules**. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in the following association rule:

$$computer \Rightarrow antivirus\_software\ [support = 2\%, confidence = 60\%]. \qquad (6.1)$$

Rule **support** and **confidence** are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for Rule (6.1) means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence threshold.** These thresholds can be a set by users or domain experts. Additional analysis can be performed to discover interesting statistical correlations between associated items.

6.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules

Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be an itemset. Let D, the task-relevant data, be a set of database transactions where each transaction T is a nonempty itemset such that $T \subseteq \mathcal{I}$. Each transaction is associated with an identifier, called a *TID*. Let A be a set of items. A transaction T is said to contain A if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set D with **support** s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., the *union* of sets A and B say, or, both A and B). This is taken to be the probability, $P(A \cup B)$.[1] The rule $A \Rightarrow B$ has **confidence** c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, $P(B|A)$. That is,

$$support(A \Rightarrow B) = P(A \cup B) \tag{6.2}$$

$$confidence(A \Rightarrow B) = P(B|A). \tag{6.3}$$

Rules that satisfy both a minimum support threshold (*min_sup*) and a minimum confidence threshold (*min_conf*) are called **strong**. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an **itemset**.[2] An itemset that contains k items is a **k-itemset**. The set {*computer, antivirus_software*} is a 2-itemset. The **occurrence frequency of an itemset** is the number of transactions that contain the itemset. This is also known, simply, as the **frequency**, **support count**, or **count** of the itemset. Note that the itemset support defined in Eq. (6.2) is sometimes referred to as *relative support*, whereas the occurrence frequency is called the **absolute support**. If the relative support of an itemset I satisfies a prespecified **minimum support threshold** (i.e., the absolute support of I satisfies the corresponding **minimum support count threshold**), then I is a **frequent** itemset.[3] The set of frequent k-itemsets is commonly denoted by L_k.[4]

From Eq. (6.3), we have

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}. \tag{6.4}$$

[1]Notice that the notation $P(A \cup B)$ indicates the probability that a transaction contains the *union* of sets A and B (i.e., it contains every item in A and B). This should not be confused with $P(A \text{ or } B)$, which indicates the probability that a transaction contains either A or B.

[2]In the data mining research literature, "itemset" is more commonly used than "item set."

[3]In early work, itemsets satisfying minimum support were referred to as **large**. This term, however, is somewhat confusing as it has connotations of the number of items in an itemset rather than the frequency of occurrence of the set. Hence, we use the more recent term **frequent**.

[4]Although the term **frequent** is preferred over **large**, for historic reasons frequent k-itemsets are still denoted as L_k.

Equation (6.4) shows that the confidence of rule $A \Rightarrow B$ can be easily derived from the support counts of A and $A \cup B$. That is, once the support counts of A, B, and $A \cup B$ are found, it is straightforward to derive the corresponding association rules $A \Rightarrow B$ and $B \Rightarrow A$ and check whether they are strong. Thus, the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. **Find all frequent itemsets:** By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min_sup*.
2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence.

Additional interestingness measures can be applied for the discovery of correlation relationships between associated items, as will be discussed in Section 6.3. Because the second step is much less costly than the first, the overall performance of mining association rules is determined by the first step.

A major challenge in mining frequent itemsets from a large data set is the fact that such mining often generates a huge number of itemsets satisfying the minimum support (*min_sup*) threshold, especially when *min_sup* is set low. This is because if an itemset is frequent, each of its subsets is frequent as well. A long itemset will contain a combinatorial number of shorter, frequent sub-itemsets. For example, a frequent itemset of length 100, such as $\{a_1, a_2, \ldots, a_{100}\}$, contains $\binom{100}{1} = 100$ frequent 1-itemsets: $\{a_1\}, \{a_2\}, \ldots, \{a_{100}\}$; $\binom{100}{2}$ frequent 2-itemsets: $\{a_1, a_2\}, \{a_1, a_3\}, \ldots, \{a_{99}, a_{100}\}$; and so on. The total number of frequent itemsets that it contains is thus

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \tag{6.5}$$

This is too huge a number of itemsets for any computer to compute or store. To overcome this difficulty, we introduce the concepts of *closed frequent itemset* and *maximal frequent itemset*.

An itemset X is **closed** in a data set D if there exists no proper super-itemset Y[5] such that Y has the same support count as X in D. An itemset X is a **closed frequent itemset** in set D if X is both closed and frequent in D. An itemset X is a **maximal frequent itemset** (or **max-itemset**) in a data set D if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in D.

Let $\mathcal{C}$ be the set of closed frequent itemsets for a data set D satisfying a minimum support threshold, *min_sup*. Let $\mathcal{M}$ be the set of maximal frequent itemsets for D satisfying *min_sup*. Suppose that we have the support count of each itemset in $\mathcal{C}$ and $\mathcal{M}$. Notice that $\mathcal{C}$ and its count information can be used to derive the whole set of frequent itemsets.

[5] Y is a proper super-itemset of X if X is a proper sub-itemset of Y, that is, if $X \subset Y$. In other words, every item of X is contained in Y but there is at least one item of Y that is not in X.

Thus, we say that $\mathcal{C}$ contains complete information regarding its corresponding frequent itemsets. On the other hand, $\mathcal{M}$ registers only the support of the maximal itemsets. It usually does not contain the complete support information regarding its corresponding frequent itemsets. We illustrate these concepts with Example 6.2.

Example 6.2 **Closed and maximal frequent itemsets.** Suppose that a transaction database has only two transactions: $\{\langle a_1, a_2, \ldots, a_{100}\rangle; \langle a_1, a_2, \ldots, a_{50}\rangle\}$. Let the minimum support count threshold be $min\_sup = 1$. We find two closed frequent itemsets and their support counts, that is, $\mathcal{C} = \{\{a_1, a_2, \ldots, a_{100}\} : 1; \{a_1, a_2, \ldots, a_{50}\} : 2\}$. There is only one maximal frequent itemset: $\mathcal{M} = \{\{a_1, a_2, \ldots, a_{100}\} : 1\}$. Notice that we cannot include $\{a_1, a_2, \ldots, a_{50}\}$ as a maximal frequent itemset because it has a frequent superset, $\{a_1, a_2, \ldots, a_{100}\}$. Compare this to the preceding where we determined that there are $2^{100} - 1$ frequent itemsets, which are too many to be enumerated!

The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from $\mathcal{C}$, we can derive, say, (1) $\{a_2, a_{45} : 2\}$ since $\{a_2, a_{45}\}$ is a sub-itemset of the itemset $\{a_1, a_2, \ldots, a_{50} : 2\}$; and (2) $\{a_8, a_{55} : 1\}$ since $\{a_8, a_{55}\}$ is not a sub-itemset of the previous itemset but of the itemset $\{a_1, a_2, \ldots, a_{100} : 1\}$. However, from the maximal frequent itemset, we can only assert that both itemsets ($\{a_2, a_{45}\}$ and $\{a_8, a_{55}\}$) are frequent, but we cannot assert their actual support counts. ■

6.2 Frequent Itemset Mining Methods

In this section, you will learn methods for mining the simplest form of frequent patterns such as those discussed for market basket analysis in Section 6.1.1. We begin by presenting **Apriori**, the basic algorithm for finding frequent itemsets (Section 6.2.1). In Section 6.2.2, we look at how to generate strong association rules from frequent itemsets. Section 6.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability. Section 6.2.4 presents pattern-growth methods for mining frequent itemsets that confine the subsequent search space to only the data sets containing the current frequent itemsets. Section 6.2.5 presents methods for mining frequent itemsets that take advantage of the vertical data format.

6.2.1 Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules [AS94b]. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where k-itemsets are used to explore $(k+1)$-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by L_1. Next, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the **Apriori property** is used to reduce the search space.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, *min_sup*, then I is not frequent, that is, $P(I) < min\_sup$. If an item A is added to the itemset I, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than I. Therefore, $I \cup A$ is not frequent either, that is, $P(I \cup A) < min\_sup$.

This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well.* It is called *antimonotonicity* because the property is monotonic in the context of failing a test.[6]

"How is the Apriori property used in the algorithm?" To understand this, let us look at how L_{k-1} is used to find L_k for $k \geq 2$. A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step**: To find L_k, a set of **candidate** k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the $(k-1)$-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \cdots < l_i[k-1]$. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first $(k-2)$ items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $\{l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]\}$.
2. **The prune step**: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A database scan to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property

[6]The Apriori property has many applications. For example, it can also be used to prune search during data cube computation (Chapter 5).

is used as follows. Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any $(k-1)$-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

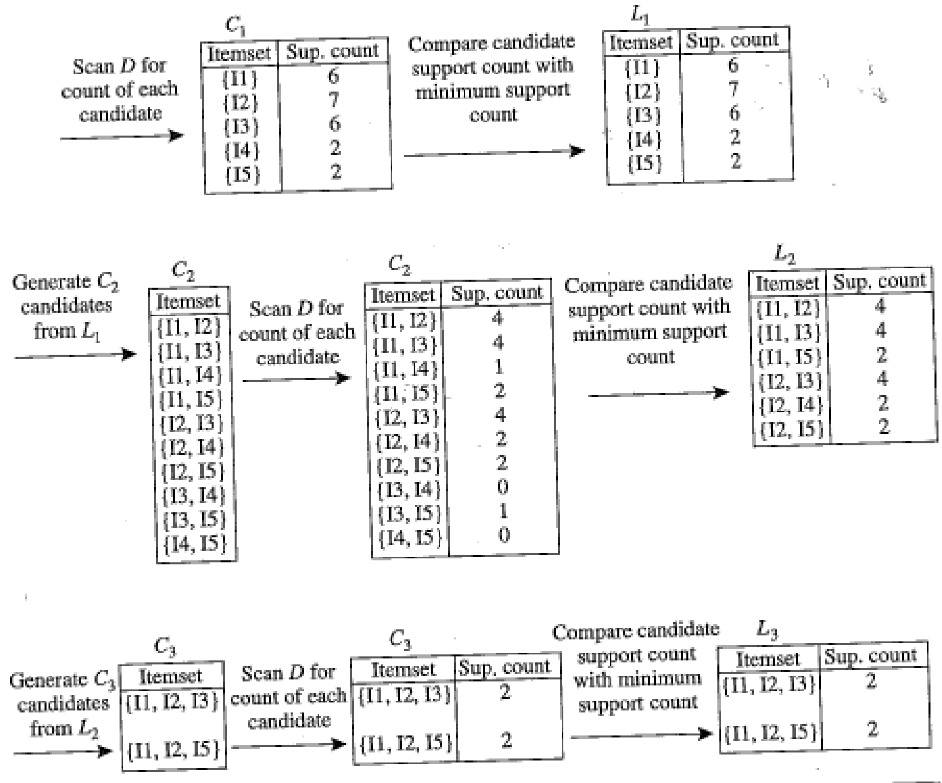
Example 6.3 **Apriori.** Let's look at a concrete example, based on the *AllElectronics* transaction database, D, of Table 6.1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 6.2 to illustrate the Apriori algorithm for finding frequent itemsets in D.

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. The algorithm simply scans all of the transactions to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is, $min\_sup = 2$. (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is $2/9 = 22\%$.) The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.
3. To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2.[7] C_2 consists of $\binom{|L_1|}{2}$ 2-itemsets. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.

Table 6.1 Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

[7] $L_1 \bowtie L_1$ is equivalent to $L_1 \times L_1$, since the definition of $L_k \bowtie L_k$ requires the two joining itemsets to share $k-1=0$ items.


Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan D for count of each candidate
Compare candidate support count with minimum support count

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 6.2 Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

4. Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in the middle table of the second row in Figure 6.2.

5. The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.

6. The generation of the set of the candidate 3-itemsets, C_3, is detailed in Figure 6.3. From the join step, we first get $C_3 = L_2 \bowtie L_2 =$ {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3. Note that when given a candidate k-itemset, we only need to check if its $(k-1)$-subsets are frequent since the Apriori algorithm uses a level-wise

(a) Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L_2. Therefore, keep {I1, I2, I3} in C_3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

(c) Therefore, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

Figure 6.3 Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

search strategy. The resulting pruned version of C_3 is shown in the first table of the bottom row of Figure 6.2.

7. The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support (Figure 6.2).

8. The algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 4-itemsets, C_4. Although the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, $C_4 = \phi$, and the algorithm terminates, having found all of the frequent itemsets. ■

Figure 6.4 shows pseudocode for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets, L_1. In steps 2 through 10, L_{k-1} is used to generate candidates C_k to find L_k for $k \geq 2$. The apriori_gen procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3). This procedure is described later. Once all of the candidates have been generated, the database is scanned (step 4). For each transaction, a subset function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6 and 7). Finally, all the candidates satisfying the minimum support (step 9) form the set of frequent itemsets, L (step 11).

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   $L_1$ = find_frequent_1-itemsets(D);
(2)   for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)       $C_k$ = apriori_gen($L_{k-1}$);
(4)       for each transaction $t \in D$ { // scan $D$ for counts
(5)           $C_t$ = subset($C_k$, $t$); // get the subsets of $t$ that are candidates
(6)           for each candidate $c \in C_t$
(7)               c.count++;
(8)       }
(9)       $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)  }
(11)  return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
(1)   for each itemset $l_1 \in L_{k-1}$
(2)       for each itemset $l_2 \in L_{k-1}$
(3)           if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2])$
                  $\wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)               $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)               if has_infrequent_subset($c$, $L_{k-1}$) then
(6)                   delete $c$; // prune step: remove unfruitful candidate
(7)               else add $c$ to $C_k$;
(8)           }
(9)   return $C_k$;

procedure has_infrequent_subset($c$: candidate $k$-itemset;
          $L_{k-1}$: frequent $(k-1)$-itemsets); // use prior knowledge
(1)   for each $(k-1)$-subset $s$ of $c$
(2)       if $s \notin L_{k-1}$ then
(3)           return TRUE;
(4)   return FALSE;
```

Figure 6.4 Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

A procedure can then be called to generate association rules from the frequent itemsets. Such a procedure is described in Section 6.2.2.

The apriori_gen procedure performs two kinds of actions, namely, **join** and **prune**, as described before. In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates (steps 1–4). The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure has_infrequent_subset.

6.2.2 Generating Association Rules from Frequent Itemsets

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum confidence). This can be done using Eq. (6.4) for confidence, which we show again here for completeness:

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

The conditional probability is expressed in terms of itemset support count, where $support\_count(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $support\_count(A)$ is the number of transactions containing the itemset A. Based on this equation, association rules can be generated as follows:

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

Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

Example 6.4 **Generating association rules.** Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X? The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:

$$\begin{array}{ll}
\{I1, I2\} \Rightarrow I5, & confidence = 2/4 = 50\% \\
\{I1, I5\} \Rightarrow I2, & confidence = 2/2 = 100\% \\
\{I2, I5\} \Rightarrow I1, & confidence = 2/2 = 100\% \\
I1 \Rightarrow \{I2, I5\}, & confidence = 2/6 = 33\% \\
I2 \Rightarrow \{I1, I5\}, & confidence = 2/7 = 29\% \\
I5 \Rightarrow \{I1, I2\}, & confidence = 2/2 = 100\%
\end{array}$$

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

6.2.3 Improving the Efficiency of Apriori

"How can we further improve the efficiency of Apriori-based mining?" Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized as follows: