• Apriori algorithm is used to find association between two objects.
min Support = 50%.
Threshold confidence = 70%.

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

So we have to find the association between items for eg how 1 & 3 are associated with each other.

Step 1: To create Item set (Distinct)

Item set	Support
1	2/4 = 50%
2	3/4 = 75%
3	3/4 = 75%
4	1/4 = 25% ×
5	3/4 = 75%

To find support, check how many times the particular item occur in transaction like 1 is occured 2 times & hence Support = 2/4

Step

As by observation, we can see, support of 4 is 25% i.e less than minimum support hence it is removed from itemset.
(item set → 1, 2, 3, 5)

Step 2: Make pairs

Item set	Support
{1,2}	1/4 = 25% ×
{1,3}	2/4 = 50%
{1,5}	1/4 = 25% ×
{2,3}	2/4 = 50%
{2,5}	3/4 = 75%
{3,5}	2/4 = 50%

Remove {1,2} & {1,5} as Support is less than 50%

Step 3 : Possible triplets out of {1,2,3,5} & consider only triplets whose support is minimum 25% i.e {1,2,3}, {1,3,5} {2,3,5}

Itemset	Support
{1,3,5}	1/4 = 25% ✗
{2,3,5}	2/4 = 50% ✓
{1,2,3}	1/4 = 25% ✗

only one itemset is selected {2,3,5} with support 2

To generate rules this itemset is selected.

(2,3,5)

Rules	Support	Confidence
(2^3) → 5	2	2/2 = 100% ✓
(3^5) → 2	2	2/2 = 100% ✓
(2^5) → 3	2	2/3 = 66%
2 → (3^5)	2	2/3 = 66%
5 → (2^3)	2	2/3 = 66%
3 → (2^5)	2	2/3 = 66%

To calculate confidence

Confidence = S(A∪B) / S(A)

eg. ① $\underline{(2^3)} \rightarrow \underline{5}$ (A → B) = (S(2^3) ∪ 5) / S(2^3) (triplet support is 2)

= 2/2 = 100%.

② $\underline{(3^5)} \rightarrow \underline{2}$ = (S(3^5) ∪ 2) / S(3^5)

= 2/2 = 100%

③ $\underline{(2^5)} \rightarrow \underline{3}$ = (S(2^5) ∪ 3) / S(2^5)

= 2/3 = 66%.

④ $\underline{2} \rightarrow \underline{(3^5)}$ = S(2 ∪ (3^5)) / S(2) = 2/3 = 66%.

⑤ 5 → (2^3) = S(5 ∪ (2^3)) / S(5) = 2/3 = 66%.

⑥ $\underline{3} \rightarrow \underline{(2^5)}$ = S(3 ∪ (2^5)) / S(3)

= 2/3 = 66%.

Now compare all confidence which are less than threshold ie 70%. so only 1st & 2nd will be used i.e

(2^3) → 5

(3^5) → 2

Consider Transactional data for an AllElectronics branch

TID	List of Item ID's
T100	I_1, I_2, I_5
T200	I_2, I_4
T300	I_2, I_3
T400	I_1, I_2, I_4
T500	I_1, I_3
T600	I_2, I_3
T700	I_1, I_3
T800	I_1, I_2, I_3, I_5
T900	I_1, I_2

There are nine transactions in this database, that is $|D| = 9$. Apply the Apriori algorithm for finding frequent itemsets in D. Consider min_support = 2 & confidence = 50%.

Step 1: To create itemset distinct.

Itemset	Support
I_1	6 = 6/9
I_2	7 = 7/9
I_3	6 = 6/9
I_4	2 = 2/9
I_5	2 = 2/9

Step 2: As minimum support = 2 hence all itemsets I_1, I_2, I_3, I_4, I_5 will be considered.

$\{I_1, I_2, I_3, I_4, I_5\}$

Step 3: Make pairs.

Itemset	Support
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1 ×
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2
$\{I_3, I_4\}$	0 ×
$\{I_3, I_5\}$	1 ×
$\{I_4, I_5\}$	0 ×

nsider Transactions

Remove all pairs that has support less than 2.

Hence

Item set	Support
$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2

Step 4: Possible Triplets out of $\{I_1, I_2, I_3, I_4, I_5\}$

itemset	support	
$\{I_1, I_2, I_3\}$	1	✗
$\{I_1, I_3, I_4\}$	0	✗
$\{I_1, I_4, I_5\}$	0	✗
$\{I_2, I_3, I_4\}$	0	✗
$\{I_2, I_4, I_5\}$	0	✗
$\{I_3, I_4, I_5\}$	0	✗
$\{I_1, I_2, I_5\}$	2	✓

Hence only $\{I_1, I_2, I_5\}$ with support 2

To generate rules this itemset is selected

Rules	Support	confidence
$(I_1 \wedge I_2) \rightarrow I_5$	2	$2/4 = 50\%$.
$(I_2 \wedge I_5) \rightarrow I_1$	2	$2/2 = 100\%$.
$(I_1 \wedge I_5) \rightarrow I_2$	2	$2/2 = 100\%$.
$I_1 \rightarrow (I_2 \wedge I_5)$	2	$2/6 = 33\%$.
$I_2 \rightarrow (I_1 \wedge I_5)$	2	$2/7 = 29\%$.
$I_5 \rightarrow (I_1 \wedge I_2)$	2	$2/2 = 100\%$

To calculate confidence

$$\text{confidence} = \frac{S(A \cup B)}{S(A)}$$

1) $\underbrace{(I_1 \wedge I_3)}_{A} \rightarrow \underbrace{I_5}_{B} = \frac{S(I_1 \wedge I_3) \cup S(I_5)}{S(I_1 \wedge I_3)}$

$= \frac{2}{4} = 50\%.$

2) $\underbrace{(I_3 \wedge I_5)}_{A} \rightarrow \underbrace{I_1}_{B} = \frac{S(I_3 \wedge I_5) \cup S(I_1)}{S(I_3 \wedge I_5)}$

$= \frac{2}{2} = 100\%.$

3) $(I_1 \wedge I_5) \rightarrow I_2 = \frac{S(I_1, I_2, I_3)}{S(I_1 \wedge I_5)} = \frac{2}{2} = 100\%.$

4) $I_1 \rightarrow I_2 \wedge I_5 = \frac{S(I_1, I_2, I_3)}{S(I_1)} = \frac{2}{6} = 33\%.$

5) $I_2 \rightarrow I_1 \wedge I_5 = \frac{S(I_1, I_2, I_3)}{S(I_2)} = \frac{2}{7} = 29\%.$

6) $I_5 \rightarrow I_1 \wedge I_2 = \frac{S(I_1, I_2, I_3)}{S(I_5)} = \frac{2}{2} = 100\%.$

Now compare all confidence & select all rules that gives minimum 50% confidence hence rules are

1) $(I_2 \wedge I_3) \rightarrow I_5$
2) $(I_3 \wedge I_5) \rightarrow I_1$
3) $(I_1 \wedge I_5) \rightarrow I_2$
4) $I_5 \rightarrow (I_1 \wedge I_2)$

Apriori Algorithm

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule leaning that analyzes that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. In this tutorial, we will discuss the apriori algorithm with examples.

Introduction

We take an example to understand the concept better. You must have noticed that the Pizza shop seller makes a pizza, soft drink, and breadstick combo together. He also offers a discount to their customers who buy these combos. Do you ever think why does he do so? He thinks that customers who buy pizza also buy soft drinks and breadsticks. However, by making combos, he makes it easy for the customers. At the same time, he also increases his sales performance.

Similarly, you go to Big Bazar, and you will find biscuits, chips, and Chocolate bundled together. It shows that the shopkeeper makes it comfortable for the customers to buy these products in the same place.

The above two examples are the best examples of Association Rules in Data Mining. It helps us to learn the concept of apriori algorithms.

What is Apriori Algorithm?

Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.

Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

Components of Apriori algorithm

The given three components comprise the apriori algorithm.

1. Support
2. Confidence
3. Lift

Let's take an example to understand this concept.

We have already discussed above; you need a huge database containing a large no of transactions. Suppose you have 4000 customers transactions in a Big Bazar. You have to calculate the Support, Confidence, and Lift for two products, and you may say Biscuits and Chocolate. This is because customers frequently buy these two items together.

Out of 4000 transactions, 400 contain Biscuits, whereas 600 contain Chocolate, and these 600 transactions include a 200 that includes Biscuits and chocolates. Using this data, we will find out the support, confidence, and lift.

Support

Support refers to the default popularity of any product. You find the support as a quotient of the division of the number of transactions comprising that product by the total number of transactions. Hence, we get

Support (Biscuits) = (Transactions relating biscuits) / (Total transactions)

= 400/4000 = 10 percent.

Confidence

Confidence refers to the possibility that the customers bought both biscuits and chocolates together. So, you need to divide the number of transactions that comprise both biscuits and chocolates by the total number of transactions to get the confidence.

Hence,

Confidence = (Transactions relating both biscuits and Chocolate) / (Total transactions involving Biscuits) = 200/400

= 50 percent.

It means that 50 percent of customers who bought biscuits bought chocolates also.

Lift

Consider the above example; lift refers to the increase in the ratio of the sale of chocolates when you sell biscuits. The mathematical equations of lift are given below.

Lift = (Confidence (Biscuits - chocolates)/ (Support (Biscuits)

= 50/10 = 5

It means that the probability of people buying both biscuits and chocolates together is five times more than that of purchasing the biscuits alone. If the lift value is below one, it requires that the people are unlikely to buy both the items together. Larger the value, the better is the combination.

How does the Apriori Algorithm work in Data Mining?

We will understand this algorithm with the help of an example

Consider a Big Bazar scenario where the product set is P = {Rice, Pulse, Oil, Milk, Apple}. The database comprises six transactions where 1 represents the presence of the product and 0 represents the absence of the product.

Transaction ID	Rice	Pulse	Oil Milk	Apple
t1	1	1	1	0	0
t2	0	1	1	1	0
t3	0	0	0	1	1
t4	1	1	0	1	0
t5	1	1	1	0	1
t6	1	1	1	1	1

The Apriori Algorithm makes the given assumptions

- All subsets of a frequent itemset must be frequent.
- The subsets of an infrequent item set must be infrequent.
- Fix a threshold support level. In our case, we have fixed it at 50 percent.

Step 1

Make a frequency table of all the products that appear in all the transactions. Now, short the frequency table to add only those products with a threshold support level of over 50 percent. We find the given frequency table.

Product	Frequency (Number of transactions)
Rice (R)	4
Pulse(P)	5
Oil(O)	4

Milk(M)	4

The above table indicated the products frequently bought by the customers.

Step 2

Create pairs of products such as RP, RO, RM, PO, PM, OM. You will get the given frequency table.

Itemset	Frequency (Number of transactions)
RP	4
RO	3
RM	2
PO	4
PM	3
OM	2

Step 3

Implementing the same threshold support of 50 percent and consider the products that are more than 50 percent. In our case, it is more than 3

Thus, we get RP, RO, PO, and PM

Step 4

Now, look for a set of three products that the customers buy together. We get the given combination.

1. RP and RO give RPO
2. PO and PM give POM

Step 5

Calculate the frequency of the two itemsets, and you will get the given frequency table.

Itemset	Frequency (Number of transactions)
RPO	4
POM	3

If you implement the threshold assumption, you can figure out that the customers' set of three products is RPO.

We have considered an easy example to discuss the apriori algorithm in data mining. In reality, you find thousands of such combinations.

How to improve the efficiency of the Apriori Algorithm?

There are various methods used for the efficiency of the Apriori algorithm

Hash-based itemset counting

In hash-based itemset counting, you need to exclude the k-itemset whose equivalent hashing bucket count is least than the threshold is an infrequent itemset.

Transaction Reduction

In transaction reduction, a transaction not involving any frequent X itemset becomes not valuable in subsequent scans.

Apriori Algorithm in data mining

We have already discussed an example of the apriori algorithm related to the frequent itemset generation. Apriori algorithm has many applications in data mining.

The primary requirements to find the association rules in data mining are given below.

Use Brute Force

Analyze all the rules and find the support and confidence levels for the individual rule. Afterward, eliminate the values which are less than the threshold support and confidence levels.

The two-step approaches

The two-step approach is a better option to find the associations rules than the Brute Force method.

Step 1

In this article, we have already discussed how to create the frequency table and calculate itemsets having a greater support value than that of the threshold support.

Step 2

To create association rules, you need to use a binary partition of the frequent itemsets. You need to choose the ones having the highest confidence levels.

In the above example, you can see that the RPO combination was the frequent itemset. Now, we find out all the rules using RPO.

RP-O, RO-P, PO-R, O-RP, P-RO, R-PO

You can see that there are six different combinations. Therefore, if you have n elements, there will be 2^n - 2 candidate association rules.

Advantages of Apriori Algorithm

- It is used to calculate large itemsets.
- Simple to understand and apply.

Disadvantages of Apriori Algorithms

- Apriori algorithm is an expensive method to find support since the calculation has to pass through the whole database.
- Sometimes, you need a huge number of candidate rules, so it becomes computationally more expensive.

5.2.3 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:

Hash-based technique (hashing itemsets into corresponding buckets): A hash-based technique can be used to reduce the size of the candidate k-itemsets, C_k, for $k > 1$. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L_1, from the candidate 1-itemsets in C_1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts (Figure 5.5). A 2-itemset whose corresponding bucket count in the hash table is below the support

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

H_2

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

Figure 5.5 Hash table, H_2, for candidate 2-itemsets: This hash table was generated by scanning the transactions of Table 5.1 while determining L_1 from C_1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of the candidate k-itemsets examined (especially when $k = 2$).

Transaction reduction (reducing the number of transactions scanned in future iterations): A transaction that does not contain any frequent k-itemsets cannot contain any frequent $(k+1)$-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for j-itemsets, where $j > k$, will not require it.

Partitioning (partitioning the data to find candidate itemsets): A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure 5.6). It consists of two phases. In Phase I, the algorithm subdivides the transactions of D into n nonoverlapping partitions. If the minimum support threshold for transactions in D is *min_sup*, then the minimum support count for a partition is *min_sup* × *the number of transactions in that partition*. For each partition, all frequent itemsets within the partition are found. These are referred to as local frequent itemsets. The procedure employs a special data structure that, for each itemset, records the TIDs of the transactions containing the items in the itemset. This allows it to find all of the local frequent k-itemsets, for $k = 1, 2, \ldots$, in just one scan of the database.

A local frequent itemset may or may not be frequent with respect to the entire database, D. *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.* Therefore, all local frequent itemsets are candidate itemsets with respect to D. The collection of frequent itemsets from all partitions forms the **global candidate itemsets** with respect to D. In Phase II, a second scan of D is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Sampling (mining on a subset of the given data): The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, we trade off some degree of accuracy

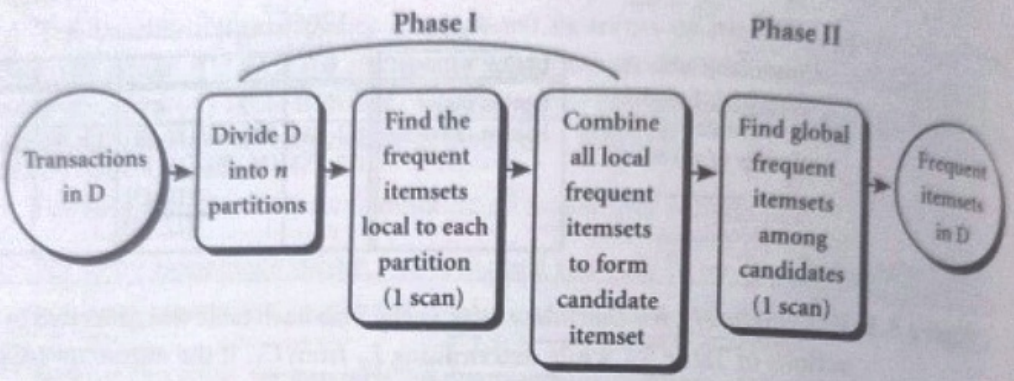

Figure 5.6 Mining by partitioning the data.

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

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

Other variations involving the mining of multilevel and multidimensional associations rules are discussed in the rest of this chapter. The mining of associations related to spatial data and multimedia data are discussed in Chapter 10.

5.2.4 Mining Frequent Itemsets without Candidate Generation

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to [illegible] performance gain. However, it can suffer from [illegible]

databases. We begin in Section 5.1.1 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 5.1.2. Section 5.1.3 presents a road map to the different kinds of frequent patterns, association rules, and correlation rules that can be mined.

5.1.1 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 5.1). The discovery of such 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

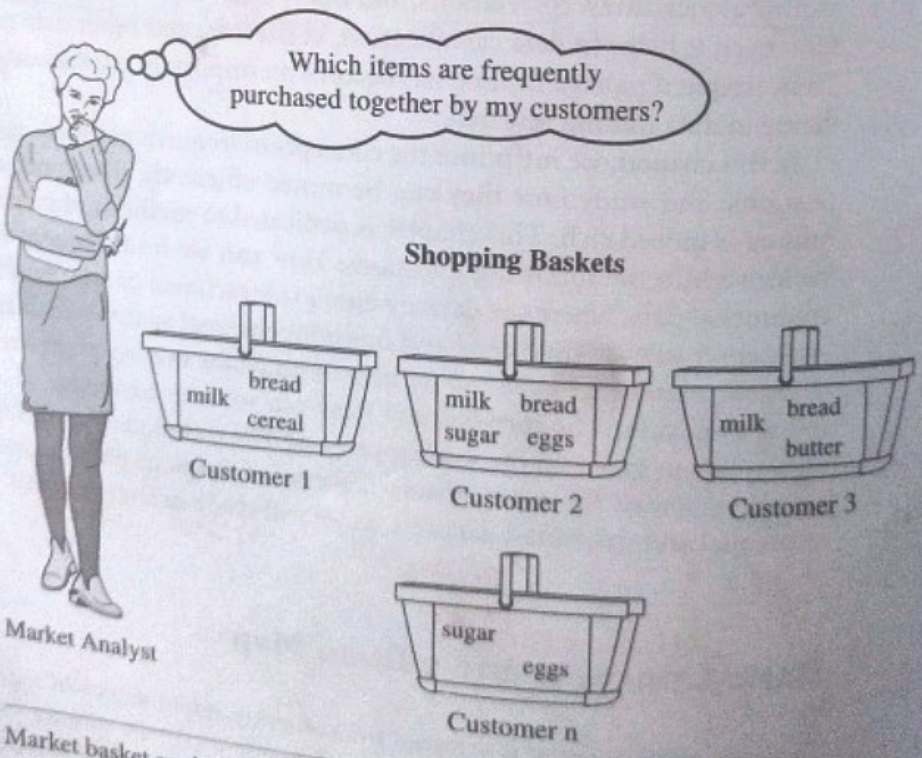

Figure 5.1 Market basket analysis.

milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such 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 5.1 **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 in order to further encourage the sale of such items together. 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 Association Rule (5.1) below:

$$computer \Rightarrow antivirus\_software \quad [support = 2\%, confidence = 60\%] \tag{5.1}$$

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 Association Rule (5.1) 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**. Such thresholds can be set by users or domain experts. Additional analysis can be performed to uncover interesting statistical correlations between associated items.

5.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, 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, or say, 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,

$$support(A \Rightarrow B) = P(A \cup B) \quad (5.2)$$

$$confidence(A \Rightarrow B) = P(B|A). \quad (5.3)$$

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 Equation (5.2) 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 Equation (5.3), we have

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)} \quad (5.4)$$

Equation (5.4) 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

[1] Notice that the notation $P(A \cup B)$ indicates the probability that a transaction contains the union of set A and set B (i.e., it contains every item in A and in B). This should not be confused with $P(A$ 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 to 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 historical reasons frequent k-itemsets are still denoted as L_k.

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 5.4. 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,

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \tag{5.5}$$

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

Let C be the set of closed frequent itemsets for a data set S satisfying a minimum support threshold, *min_sup*. Let $\mathcal{M}$ be the set of maximal frequent itemsets for S satisfying *min_sup*. Suppose that we have the support count of each itemset in C and $\mathcal{M}$. Notice that C and its count information can be used to derive the whole set of frequent itemsets. Thus we say that C contains complete information regarding its corresponding frequent itemsets. On the other hand, $\mathcal{M}$ registers only the support of the maximal 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.

It usually does not contain the complete support information regarding its corresponding frequent itemsets. We illustrate these concepts with the following example.

Example 5.2 **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, $C = \{\{a_1, a_2, \ldots, a_{100}\} : 1; \{a_1, a_2, \ldots, a_{50}\} : 2\}$. There is one maximal frequent itemset: $M = \{\{a_1, a_2, \ldots, a_{100}\} : 1\}$. (We cannot include $\{a_1, a_2, \ldots, a_{50}\}$ as a maximal frequent itemset because it has a frequent super-set, $\{a_1, a_2, \ldots, a_{100}\}$.) Compare this to the above, where we determined that there are $2^{100} - 1$ frequent itemsets, which is too huge a set to be enumerated!

The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from 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. ∎

5.1.3 Frequent Pattern Mining: A Road Map

Market basket analysis is just one form of frequent pattern mining. In fact, there are many kinds of frequent patterns, association rules, and correlation relationships. Frequent pattern mining can be classified in various ways, based on the following criteria:

1. **Based on the *completeness* of patterns to be mined:** As we discussed in the previous subsection, we can mine the **complete set of frequent itemsets**, the **closed frequent itemsets**, and the **maximal frequent itemsets**, given a minimum support threshold. We can also mine **constrained frequent itemsets** (i.e., those that satisfy a set of user-defined constraints), **approximate frequent itemsets** (i.e., those that derive only approximate support counts for the mined frequent itemsets), **near-match frequent itemsets** (i.e., those that tally the support count of the near or almost matching itemsets), **top-k frequent itemsets** (i.e., the k most frequent itemsets for a user-specified value, k), and so on.

   Different applications may have different requirements regarding the completeness of the patterns to be mined, which in turn can lead to different evaluation and optimization methods. In this chapter, our study of mining methods focuses on mining the *complete set of frequent itemsets*, *closed frequent itemsets*, and *constrained frequent itemsets*. We leave the mining of frequent itemsets under other completeness requirements as an exercise.

2. **Based on the *levels of abstraction* involved in the rule set:** Some methods for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rules mined includes the following rules where X is a variable representing a customer:

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}HP\_printer\text{”}) \tag{5.6}$$

$$buys(X, \text{“}laptop\_computer\text{”}) \Rightarrow buys(X, \text{“}HP\_printer\text{”}) \quad (5.7)$$

In Rules (5.6) and (5.7), the items bought are referenced at different levels of abstraction (e.g., “*computer*” is a higher-level abstraction of “*laptop computer*”). We refer to the rule set mined as consisting of **multilevel association rules**. If, instead, the rules within a given set do not reference items or attributes at different levels of abstraction, then the set contains **single-level association rules**.

- **Based on the *number of data dimensions* involved in the rule:** If the items or attributes in an association rule reference only one dimension, then it is a **single-dimensional association rule**. Note that Rule (5.1), for example, could be rewritten as Rule (5.8):

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}antivirus\_software\text{”}) \quad (5.8)$$

Rules (5.6), (5.7), and (5.8) are single-dimensional association rules because they each refer to only one dimension, *buys*.[6]

If a rule references two or more dimensions, such as the dimensions *age*, *income*, and *buys*, then it is a **multidimensional association rule**. The following rule is an example of a multidimensional rule:

$$age(X, \text{“}30\ldots39\text{”}) \wedge income(X, \text{“}42K\ldots48K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}). \quad (5.9)$$

- **Based on the *types of values* handled in the rule:** If a rule involves associations between the presence or absence of items, it is a **Boolean association rule**. For example, Rules (5.1), (5.6), and (5.7) are Boolean association rules obtained from market basket analysis.

If a rule describes associations between quantitative items or attributes, then it is a **quantitative association rule**. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (5.9) is also considered a quantitative association rule. Note that the quantitative attributes, *age* and *income*, have been discretized.

- **Based on the *kinds of rules* to be mined:** Frequent pattern analysis can generate various kinds of rules and other interesting relationships. Association rules are the most popular kind of rules generated from frequent patterns. Typically, such mining can generate a large number of rules, many of which are redundant or do not indicate a correlation relationship among itemsets. Thus, the discovered associations can be further analyzed to uncover statistical correlations, leading to **correlation rules**.

We can also mine **strong gradient relationships** among itemsets, where a gradient is the ratio of the measure of an item when compared with that of its parent (a generalized itemset), its child (a specialized itemset), or its sibling (a comparable itemset). One such example is: “*The average sales from Sony_Digital_Camera increase over 16% when sold together with Sony_Laptop_Computer*”: both Sony_Digital_Camera and Sony_Laptop_Computer are siblings, where the parent itemset is Sony.

[6] Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a *dimension*.

Based on the *kinds of patterns* to be mined. Many kinds of frequent patterns can be mined from different kinds of data sets. For this chapter, our focus is on frequent itemset mining, that is, the mining of frequent itemsets (sets of items) from transactional or relational data sets. However, other kinds of frequent patterns can be found from other kinds of data sets. **Sequential pattern mining** searches for frequent *subsequences* in a *sequence data set*, where a sequence records an ordering of events. For example, with sequential pattern mining, we can study the order in which items are frequently purchased. For instance, customers may tend to first buy a PC, followed by a digital camera, and then a memory card. **Structured pattern mining** searches for frequent *substructures* in a *structured data set*. Notice that *structure* is a general concept that covers many different kinds of structural forms, such as graphs, lattices, trees, sequences, sets, single items, or combinations of such structures. Single items are the simplest form of structure. Each element of an itemset may contain a subsequence, a subtree, and so on, and such containment relationships can be defined recursively. Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

In the next section, we will study efficient methods for mining the basic (i.e., single-level, single-dimensional, Boolean) frequent itemsets from transactional databases, and show how to generate association rules from such itemsets. The extension of this scope of mining to multilevel, multidimensional, and quantitative rules is discussed in Section 5.3. The mining of strong correlation relationships is studied in Section 5.4. Constraint-based mining is studied in Section 5.5. We address the more advanced topic of mining sequence and structured patterns in later chapters. Nevertheless, most of the methods studied here can be easily extended for mining more complex kinds of patterns.

5.2 Efficient and Scalable Frequent Itemset Mining Methods

In this section, you will learn methods for mining the simplest form of frequent patterns—*single-dimensional, single-level, Boolean frequent itemsets*, such as those discussed for market basket analysis in Section 5.1.1. We begin by presenting Apriori, the basic algorithm for finding frequent itemsets (Section 5.2.1). In Section 5.2.2, we look at how to generate strong association rules from frequent itemsets. Section 5.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability. Section 5.2.4 presents methods for mining frequent itemsets that, unlike Apriori, do not involve the generation of "candidate" frequent itemsets. Section 5.2.5 presents methods for mining frequent itemsets that take advantage of vertical data format. Methods for mining closed frequent itemsets are discussed in Section 5.2.6.

5.2.1 The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on

the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see following. 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 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**, presented below, is used to reduce the search space. We will first describe this property, and then show an example illustrating its use.

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 **antimonotone** 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 *antimonotone* because the property is monotonic in the context of failing a test.[7]

"*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). By convention, 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 $l_i[1] < l_i[2] < \ldots < l_i[k-1]$. 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 $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. 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 scan of the database 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

[7] The Apriori property has many applications. It can also be used to prune search during data cube computation (Chapter 4).

Table 5.1 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

involve heavy computation. To reduce the size of C_k, the Apriori property 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 5.3 Apriori. Let's look at a concrete example, based on the *AllElectronics* transaction database, D, of Table 5.1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 5.2 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 in order 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.[8] 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.

[8] $L_1 \bowtie L_1$ 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.

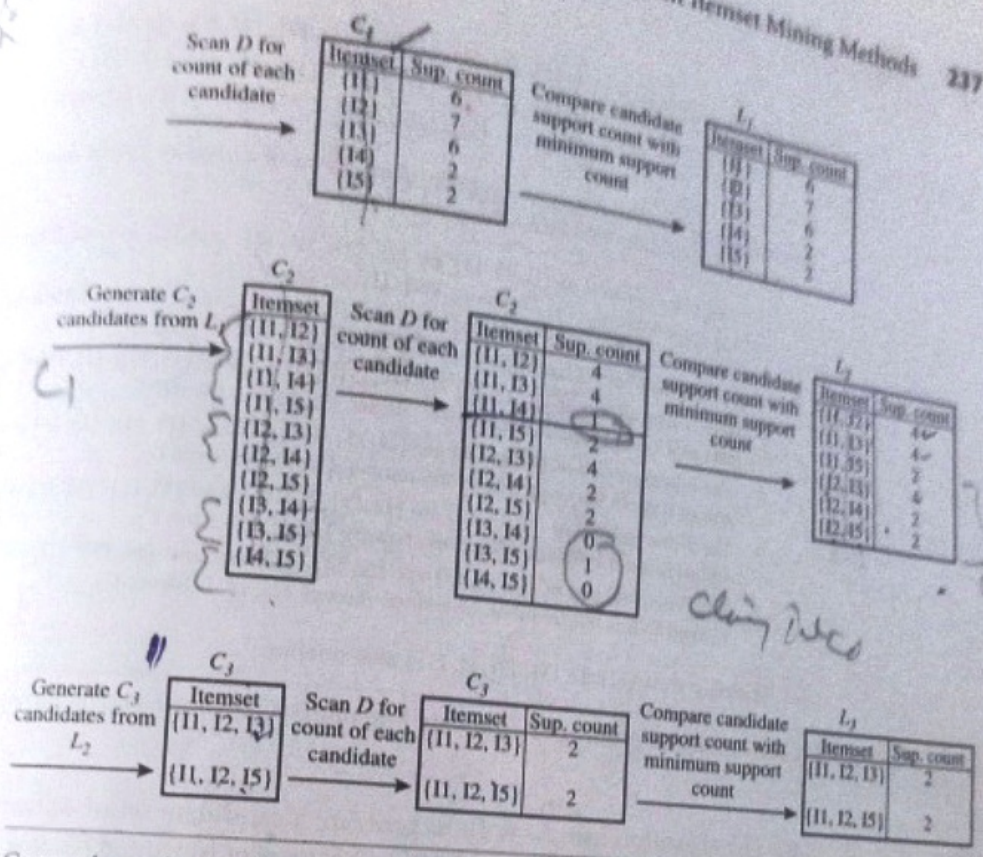

Figure 5.2 Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

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 5.2.
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 candidate 3-itemsets, C_3, is detailed in Figure 5.3. 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 search strategy. The resulting pruned version of C_3 is shown in the first table of the bottom row of Figure 5.2.
7. The transactions in D are scanned in order to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support (Figure 5.2).

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

8. The algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 4-itemsets, C_4. Although the join results in $\{\{I1, I2, I3, I5\}\}$, this itemset 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 5.4 shows pseudo-code for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets, L_1. In steps 2 to 10, L_{k-1} is used to generate candidates C_k in order 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 below. 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 of those candidates satisfying minimum support (step 9) form the set of frequent itemsets, L (step 11). A procedure can then be called to generate association rules from the frequent itemsets. Such a procedure is described in Section 5.2.2.

The apriori_gen procedure performs two kinds of actions, namely, join and prune, as described above. In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates (steps 1 to 4). The prune component (steps 5 to 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.

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)  L1 = find_frequent_1-itemsets(D);
(2)  for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)      Ck = apriori_gen(Lk−1);
(4)      for each transaction t ∈ D { // scan D for counts
(5)          Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)          for each candidate c ∈ Ct
(7)              c.count++;
(8)      }
(9)      Lk = {c ∈ Ck | c.count ≥ min_sup}
(10) }
(11) return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)  for each itemset l1 ∈ Lk−1
(2)      for each itemset l2 ∈ Lk−1
(3)          if (l1[1] = l2[1]) ∧ (l1[2] = l2[2]) ∧ ... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)              c = l1 ⋈ l2; // join step: generate candidates
(5)              if has_infrequent_subset(c, Lk−1) then
(6)                  delete c; // prune step: remove unfruitful candidate
(7)              else add c to Ck;
(8)          }
(9)  return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
         Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)  for each (k − 1)-subset s of c
(2)      if s ∉ Lk−1 then
(3)          return TRUE;
(4)  return FALSE;
```

Figure 5.4 The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

5.2.2 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 Equation (5.4) 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 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 5.4 Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown in Table 5.1. Suppose the data contain the frequent itemset $l = \{I1, I2, I5\}$. What are the association rules that can be generated from l? The nonempty subsets of l are $\{I1, I2\}$, $\{I1, I5\}$, $\{I2, I5\}$, $\{I1\}$, $\{I2\}$, and $\{I5\}$. The resulting association rules are as shown below, each listed with its confidence:

$I1 \wedge I2 \Rightarrow I5$, $\quad confidence = 2/4 = 50\%$
$I1 \wedge I5 \Rightarrow I2$, $\quad confidence = 2/2 = 100\%$
$I2 \wedge I5 \Rightarrow I1$, $\quad confidence = 2/2 = 100\%$
$I1 \Rightarrow I2 \wedge I5$, $\quad confidence = 2/6 = 33\%$
$I2 \Rightarrow I1 \wedge I5$, $\quad confidence = 2/7 = 29\%$
$I5 \Rightarrow I1 \wedge I2$, $\quad confidence = 2/2 = 100\%$

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

5.2.3 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:

Hash-based technique (hashing itemsets into corresponding buckets): A hash-based technique can be used to reduce the size of the candidate k-itemsets, C_k, for $k > 1$. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L_1, from the candidate 1-itemsets in C_1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a *hash table* structure, and increase the corresponding bucket counts (Figure 5.3). A 2-itemset whose corresponding bucket count in the hash table is below the support