- 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.

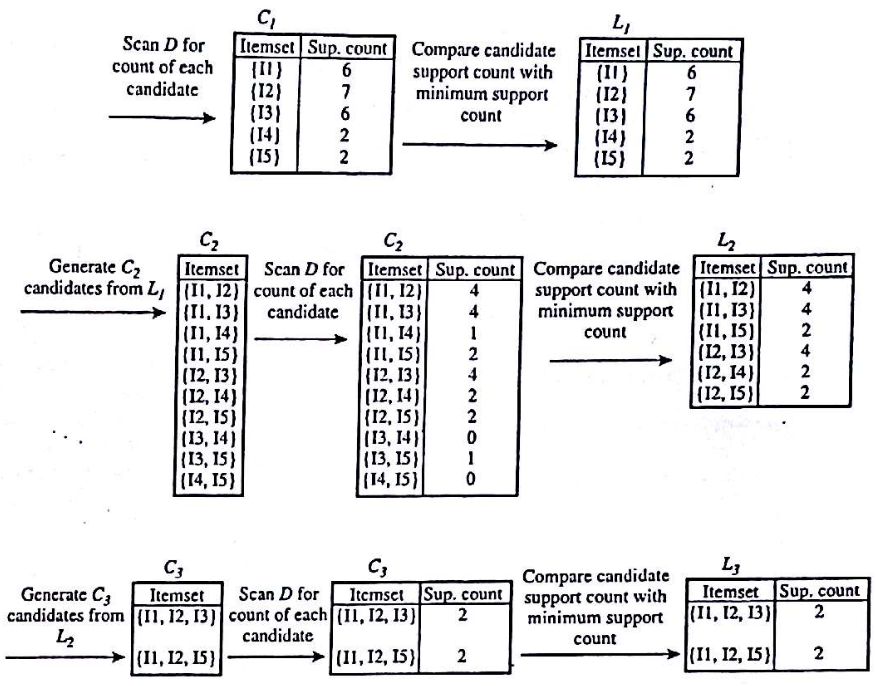
TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

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.


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 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)   $L_1$ = 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 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:

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

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 5.5). A 2-itemset whose corresponding bucket count in the hash table is below the support

Create hash table H_2 using hash function $h(x, y) = ((\textit{order of } x) \times 10 + (\textit{order of } y)) \textit{ mod } 7$ $\longrightarrow$

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