CHAPTER

Pattern mining: basic concepts and methods 4

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 smartphone, then a smart TV, and then a smart home device, 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. 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 4.1) and study how they can be mined efficiently (Section 4.2). We also discuss how to judge whether the patterns found are interesting (Section 4.3). In the subsequent chapter, we extend our discussion to advanced frequent pattern mining, including mining more complex forms of frequent patterns, and their applications.

4.1 Basic concepts

Frequent pattern mining uncovers 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 4.1.1 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 discussed in Section 4.1.2.

4.1.1 Market basket analysis: a motivating example

A set of items is referred to as an **itemset**.[1] 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 interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts

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

Data Mining. https://doi.org/10.1016/B978-0-12-811760-6.00014-X

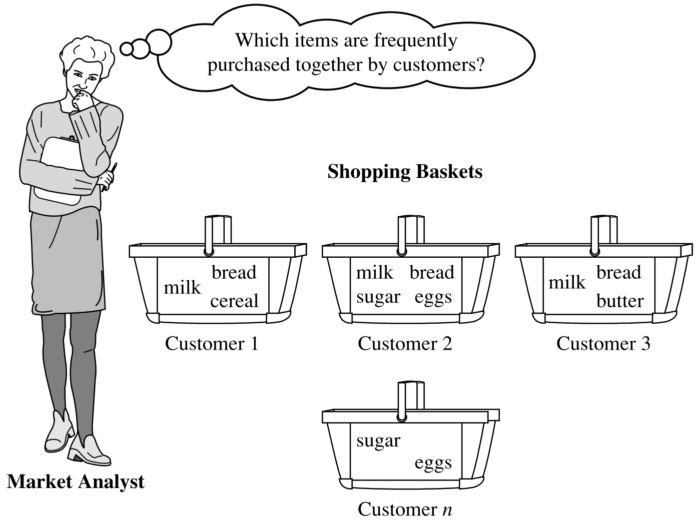

FIGURE 4.1

Market basket analysis.

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" (Fig. 4.1). 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 to the supermarket? This information can lead to increased sales, revenue, and customer acquisition by helping retailers do selective marketing and planned shelf space.

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

Example 4.1. Market basket analysis. Suppose, as manager of a retail company, 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 these results to choose marketing strategies and help create 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 reducing the prices 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 to extract 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:

$$computer \Rightarrow antivirus\_software \ \ [support = 2\%, confidence = 60\%]. \tag{4.1}$$

Rule **support** and **confidence** are two measures of rule interestingness. They reflect the usefulness and certainty of discovered rules, respectively. A support of 2% for Rule (4.1) 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 a **minimum support threshold** and a **minimum confidence threshold**. These thresholds can be set by users or domain experts. Additional analysis can be performed to discover interesting statistical correlations between associated items.

4.1.2 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, $P(A \cup B)$.[2] 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,

$$support\,(A \Rightarrow B) = P(A \cup B) \tag{4.2}$$

$$confidence\,(A \Rightarrow B) = P(B|A). \tag{4.3}$$

Rules that satisfy both a minimum support threshold (*min_sup*) and a minimum confidence threshold (*min_conf*) are called **strong**. By convention, support and confidence values are represented as percentages.

[2] 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.

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. Occurrence frequency is also referred as the **frequency**, **support count**, or **count** of the itemset. Note that the itemset support defined in Eq. (4.2) 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. (4.3), we have

$$confidence\,(A \Rightarrow B) = P(B|A) = \frac{support\,(A \cup B)}{support\,(A)} = \frac{support\_count\,(A \cup B)}{support\_count\,(A)}. \tag{4.4}$$

Eq. (4.4) 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 that can be applied for the discovery of correlation relationships between associated items will be discussed in Section 4.3. The overall performance of mining association rules is determined by the first step since the second step is much less costly than the first.

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 subitemsets. 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: $\{a_1\}$, $\{a_2\}$, ..., $\{a_{100}\}$; $\binom{100}{2}$ frequent 2-itemsets: $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_1, a_4\}$, ..., $\{a_2, a_3\}$, $\{a_2, a_4\}$, ..., $\{a_{99}, a_{100}\}$; and so on. The total number of frequent itemsets that it contains is thus

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \tag{4.5}$$

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*.

[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.

An itemset X is **closed** in a data set D if there exists no proper superitemset 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 superitemset 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. 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 4.2.

Example 4.2. 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 subitemset 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 subitemset 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. □

4.2 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 4.1.1. We begin by presenting **Apriori**, the basic algorithm for finding frequent itemsets in Section 4.2.1. In Section 4.2.2, we look at how to generate strong association rules from frequent itemsets. Section 4.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability. Section 4.2.4 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 4.2.5 presents methods for mining frequent itemsets that take advantage of the vertical data format.

[5] Y is a proper superitemset of X if X is a proper subitemset 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.

4.2.1 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.

"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 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 4.3. Apriori. Let's look at a concrete example, based on the transaction database, D, of Table 4.1. There are nine transactions in this database, that is, $|D| = 9$. We use Fig. 4.2 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.[6] 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.
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 Fig. 4.2.
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 Fig. 4.3. 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 Fig. 4.2.
7. The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support (Fig. 4.2).

Table 4.1 A transactional data set.

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

[6] $L_1 \bowtie L_1$ 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 →

C_1

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

Compare candidate support count with minimum support count →

L_1

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

Generate C_2 candidates from L_1 →

C_2

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

Scan D for count of each candidate →

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

Compare candidate support count with minimum support count →

L_2

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

Generate C_3 candidates from L_2 →

C_3

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

Scan D for count of each candidate →

C_3

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

Compare candidate support count with minimum support count →

L_3

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

FIGURE 4.2

Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 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. □

Fig. 4.4 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). 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). A procedure can then be called to generate association rules from the frequent itemsets. Such a procedure is described in Section 4.2.2.

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 4.3

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

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`.

4.2.2 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. (4.4) 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.

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:

```
L_1 = find_frequent_1-itemsets(D);
for (k = 2; L_{k-1} ≠ φ; k++) {
    C_k = apriori_gen(L_{k-1});
    for each transaction t ∈ D { // scan D for counts
        C_t = subset(C_k, t); // get the subsets of t that are candidates
        for each candidate c ∈ C_t
            c.count++;
    }
    L_k = {c ∈ C_k | c.count ≥ min_sup}
}
return L = ∪_k L_k;

procedure apriori_gen(L_{k-1}:frequent (k − 1)-itemsets)
for each itemset l_1 ∈ L_{k-1}
    for each itemset l_2 ∈ L_{k-1}
        if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2])
            ∧... ∧ (l_1[k − 2] = l_2[k − 2]) ∧ (l_1[k − 1] < l_2[k − 1]) then {
            c = l_1 ⋈ l_2; // join step: generate candidates
            if has_infrequent_subset(c, L_{k-1}) then
                delete c; // prune step: remove unfruitful candidate
            else add c to C_k;
        }
return C_k;

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

FIGURE 4.4

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

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 4.4. Generating association rules. Let's try an example based on the transactional data shown before in Table 4.1. 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:

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

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. □

4.2.3 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 (e.g., let $t = \{i_1, i_2, i_4\}$) in the database to generate the frequent 1-itemsets, L_1, we can generate all the 2-itemsets for each transaction (e.g., three 2-itemsets $\{i_1, i_2\}$, $\{i_1, i_4\}$, and $\{i_2, i_4\}$ for transaction t), hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts as shown in Fig. 4.5. A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of 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 database scans for j-itemsets, where $j > k$, will not need to consider such a transaction.

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

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 4.5

Hash table, H_2, for candidate 2-itemsets. This hash table was generated by scanning Table 4.1's transactions while determining L_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.

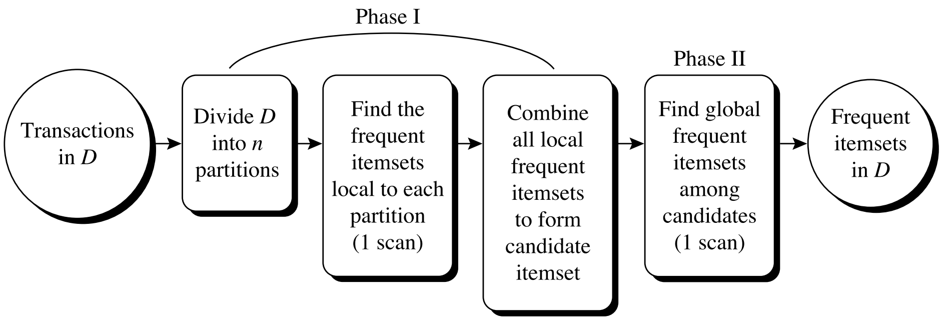

FIGURE 4.6

Mining by partitioning the data.

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 (Fig. 4.6). It consists of two phases. In phase I, the algorithm divides the transactions of D into n nonoverlapping partitions. If the minimum relative 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 the *local frequent itemsets* (i.e., the itemsets frequent within the partition) are found.
A local frequent itemset may or may not be frequent with respect to the entire database, D. However, *any itemset that is potentially frequent with respect to* D *must occur as a frequent itemset in at least one of the partitions*.[7] 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 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 against efficiency. The S sample size 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 reduce this possibility, we use a lower support threshold than the 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 the global frequent itemsets are included in L_S. If L_S actually contains all the frequent itemsets in D, then only one scan of D is required. Otherwise, a second pass can be done 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.

[7] The proof of this property is left as an exercise (see Exercise 4.3d).

Dynamic itemset counting (adding candidate itemsets at different points during a scan). A dynamic itemset counting technique is 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 after each complete database scan. The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Other variations are discussed in the next chapter or left as exercises.

4.2.4 A pattern-growth approach for mining frequent itemsets

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

- *It may still need to generate a huge number of candidate sets.* For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.
- *It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching.* It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

"Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?" An interesting method in this attempt is called **frequent pattern growth,** or simply **FP-growth**, which adopts a *divide-and-conquer* strategy as follows. First, it compresses the database representing frequent items into a **frequent pattern tree,** or **FP-tree**, which retains the itemset association information. It then divides the compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one itemset found so far, or "pattern fragment," and mines each database separately. For each "pattern fragment," only its associated data sets need to be examined. Therefore this approach may substantially reduce the size of the data sets to be searched, along with the "growth" of patterns being examined. You will see how it works in Example 4.5.

Example 4.5. FP-growth (finding frequent itemsets without candidate generation). We reexamine the mining of transaction database, D, of Table 4.1 in Example 4.3 using the frequent pattern growth approach.

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by L. Thus, we have $L = \{\{I2: 7\}, \{I1: 6\}, \{I3: 6\}, \{I4: 2\}, \{I5: 2\}\}$.

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with "null." Scan database D a second time. The items in each transaction are processed in L order (i.e., sorted according to descending support count), and a branch is created for each transaction. For example, the scan of the first transaction, "T100: I1, I2, I5," which contains three items (I2, I1, I5 in L order), leads to the construction of the first branch of the tree with three nodes, $\langle I2: 1\rangle$, $\langle I1: 1\rangle$, and $\langle I5: 1\rangle$, where I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, T200,

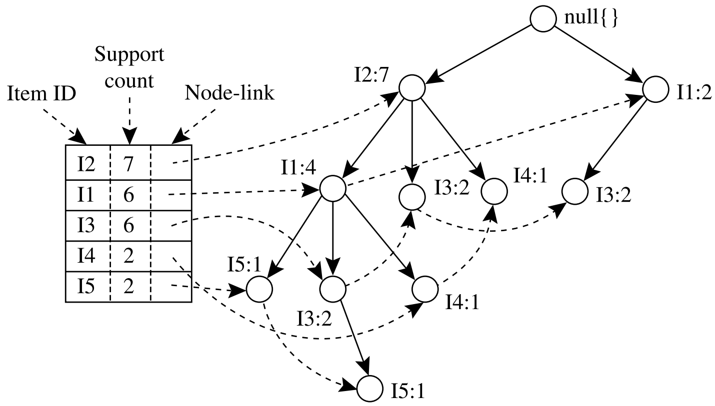

FIGURE 4.7

An FP-tree registers compressed frequent pattern information.

contains the items I2 and I4 in L order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common **prefix,** I2, with the existing path for T100. Therefore, we instead increment the count of the I2 node by 1, and create a new node, ⟨I4: 1⟩, which is linked as a child to ⟨I2: 2⟩. In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links**. The tree obtained after scanning all the transactions is shown in Fig. 4.7 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed into that of mining the FP-tree.

The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a "subdatabase," which consists of the set of *prefix paths* in the FP-tree cooccurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining of the FP-tree is summarized in Table 4.2 and detailed as follows.

- We first consider I5, which is the last item in L, rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two FP-tree branches of Fig. 4.7. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are ⟨I2, I1, I5: 1⟩ and ⟨I2, I1, I3, I5: 1⟩. Therefore, considering I5 as a suffix, its corresponding two prefix paths are ⟨I2, I1: 1⟩ and ⟨I2, I1, I3: 1⟩, which form its conditional pattern base. Using this conditional pattern base as a transaction database, we build an I5-conditional FP-tree, which contains only a single path, ⟨I2: 2, I1: 2⟩; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}.

Table 4.2 Mining the FP-tree by creating conditional (sub-)pattern bases.

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Algorithm: **FP_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

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

Output: The complete set of frequent patterns.
Method:

1. The FP-tree is constructed in the following steps:
   - **a.** Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   - **b.** Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following.
     Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that *N.item-name* = *p.item-name*, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

procedure FP_growth(*Tree*, α)
  if $Tree$ contains a single path P **then**
    for each combination (denoted as β) of the nodes in the path P
      generate pattern $\beta \cup \alpha$ with $support\_count = minimum\ support\ count\ of\ nodes\ in\ \beta$;
  else for each a_i in the header of $Tree$ {
    generate pattern $\beta = a_i \cup \alpha$ with $support\_count = a_i.support\_count$;
    construct β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$;
    if $Tree_\beta \neq \emptyset$ **then**
      call FP_growth($Tree_\beta$, β); }

FIGURE 4.8

FP-growth algorithm for discovering frequent itemsets without candidate generation.

- For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, ⟨I2: 2⟩, and derives one frequent pattern, {I2, I4: 2}.
- Similar to the preceding analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}. Its conditional FP-tree has two branches, ⟨I2: 4, I1: 2⟩ and ⟨I1: 2⟩, as shown in Fig. 4.9, which generates the set of patterns {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}.
- Finally, I1's conditional pattern base is {{I2: 4}}, with an FP-tree that contains only one node, ⟨I2: 4⟩, which generates one frequent pattern, {I2, I1: 4}.

This mining process is summarized in Fig. 4.8. □

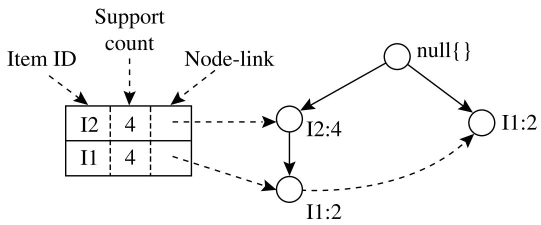

FIGURE 4.9

The conditional FP-tree associated with the conditional node I3.

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first partition the database into a set of projected databases and then construct an FP-tree and mine it in each projected database. This process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

4.2.5 Mining frequent itemsets using the vertical data format

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (i.e., $\{TID : itemset\}$), where *TID* is a transaction ID and *itemset* is the set of items bought in transaction *TID*. This is known as the **horizontal data format**. Alternatively, data can be presented in *item-TID_set* format (i.e., $\{item : TID\_set\}$), where *item* is an item name, and *TID_set* is the set of transaction identifiers containing the item. This is known as the **vertical data format**.

In this subsection, we look at how frequent itemsets can also be mined efficiently using vertical data format, which is the essence of the **Eclat** (Equivalence Class Transformation) algorithm.

Example 4.6. Mining frequent itemsets using the vertical data format. Consider the horizontal data format of the transaction database, *D*, of Table 4.1 in Example 4.3. This can be transformed into the vertical data format shown in Table 4.3 by scanning the data set once.

Table 4.3 The vertical data format of the transaction data set *D* of Table 4.1.

itemset	TID_set
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Table 4.4 2-Itemsets in vertical data format.

itemset	TID_set
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Table 4.5 3-Itemsets in vertical data format.

itemset	TID_set
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

Mining can be performed on this data set by intersecting the TID_sets of every pair of frequent single items. The minimum support count is 2. Because every single item is frequent in Table 4.3, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table 4.4. Notice that because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}. By intersecting the TID_sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table 4.5, where there are only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}. □

Example 4.6 illustrates the process of mining frequent itemsets by exploring the vertical data format. First, we transform the horizontally formatted data into the vertical format by scanning the data set once. The support count of an itemset is simply the length of the TID_set of the itemset. Starting with $k = 1$, the frequent k-itemsets can be used to construct the candidate $(k + 1)$-itemsets based on the Apriori property. The computation is done by intersection of the TID_sets of the frequent k-itemsets to compute the TID_sets of the corresponding $(k + 1)$-itemsets. This process repeats, with k incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate $(k + 1)$-itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of $(k + 1)$-itemsets (for $k \geq 1$). This is because the TID_set of each k-itemset carries the complete information required for counting such support. However, the TID_sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.

To further reduce the cost of registering long TID_sets, as well as the subsequent costs of intersections, we can use a technique called *diffset*, which keeps track of only the differences of the TID_sets of a $(k+1)$-itemset and a corresponding k-itemset. For instance, in Example 4.6 we have {I1} = {T100, T400, T500, T700, T800, T900} and {I1, I2} = {T100, T400, T800, T900}. The *diffset* between the two is *diffset*({I1, I2}, {I1}) = {T500, T700}. Thus rather than recording the four TIDs that make up the intersection of {I1} and {I2}, we can instead use *diffset* to record just two TIDs, indicating the difference between {I1} and {I1, I2}. With such compressed bookkeeping, itemset frequency can still be calculated correctly. Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

4.2.6 Mining closed and max patterns

In Section 4.1.2 we saw how frequent itemset mining may generate a huge number of frequent itemsets, especially when the *min_sup* threshold is set low or when there exist long patterns in the data set. Example 4.2 showed that closed frequent itemsets[8] can substantially reduce the number of patterns generated in frequent itemset mining while preserving the complete information regarding the set of frequent itemsets. That is, from the set of closed frequent itemsets, we can easily derive the set of frequent itemsets and their support. Thus in practice, it is more desirable to mine the set of closed frequent itemsets rather than the set of all frequent itemsets in most cases.

"How can we mine closed frequent itemsets?" A naïve approach would be to first mine the complete set of frequent itemsets and then remove every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset. However, this is quite costly. As shown in Example 4.2, this method would have to first derive $2^{100}-1$ frequent itemsets to obtain a length-100 frequent itemset, all before it could begin to eliminate redundant itemsets. This is prohibitively expensive. In fact, there exist only a very small number of closed frequent itemsets in Example 4.2's data set.

A recommended methodology is to prune the search space as soon as we can identify the case of closed itemsets during mining. For example, an itemset merging method is introduced as follows.

Itemset merging. *If every transaction containing a frequent itemset X also contains an itemset Y but not any proper superset of Y, then $X \cup Y$ forms a frequent closed itemset and there is no need to search for any itemset containing X but no Y.*

For example, in Table 4.2 of Example 4.5, the projected conditional database for prefix itemset {I5:2} is {{I2, I1}, {I2, I1, I3}}, from which we can see that each of its transactions contains itemset {I2, I1} but no proper superset of {I2, I1}. Itemset {I2, I1} can be merged with {I5} to form the closed itemset {I5, I2, I1: 2}, and we do not need to mine for closed itemsets that contain I5 but not {I2, I1}.

Many search space pruning and closure checking methods have been developed for mining frequent closed itemsets. Moreover, because maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed for mining closed itemset can be extended to mining maximal frequent itemsets. Interested readers may like to dig deeper by studying related research papers.

---

[8] Remember that X is a *closed frequent* itemset in a data set S if there exists no proper superitemset Y such that Y has the same support count as X in S, and X satisfies minimum support.

4.3 Which patterns are interesting?—Pattern evaluation methods

Most association rule mining algorithms employ a support–confidence framework. Although minimum support and confidence thresholds *help* weed out or exclude the exploration of a good number of uninteresting rules, many of the rules generated are still not interesting to many users. This is especially true *when mining at low support thresholds or mining for long patterns*. This has been a major bottleneck for successful application of association rule mining.

In this section, we first look at how even strong association rules can be uninteresting and misleading (Section 4.3.1). We then discuss how the support–confidence framework can be supplemented with additional interestingness measures based on *correlation analysis* (Section 4.3.2). Section 4.3.3 presents additional pattern evaluation measures. It then provides an overall comparison of all the measures discussed here. By the end, you will learn which pattern evaluation measures are most effective for the discovery of only interesting rules.

4.3.1 Strong rules are not necessarily interesting

The interestingness of a rule can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics "behind" the data, can be used as one step toward the goal of weeding out uninteresting rules that would otherwise be presented to the user.

"How can we tell which strong association rules are really interesting?" Let's examine the following example.

Example 4.7. A misleading "strong" association rule. Suppose we are interested in analyzing transactions with respect to the purchase of computer games and videos. Let *game* refer to the transactions containing computer games, and *video* refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, whereas 7500 included videos, and 4000 included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered:

$$\begin{aligned} &buys\,(X, \text{“}computer\ games\text{”}) \Rightarrow buys\,(X, \text{“}videos\text{”}) \\ &[support = 40\%, confidence = 66\%]. \end{aligned} \tag{4.6}$$

Rule (4.6) is a strong association rule and would therefore be reported, since its support value of $\frac{4000}{10{,}000} = 40\%$ and confidence value of $\frac{4000}{6000} = 66\%$ satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (4.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (4.6). □

Example 4.7 also illustrates that the confidence of a rule $A \Rightarrow B$ can be deceiving. It does not measure the *real strength* (or lack of strength) of the *correlation* and *implication* between A and B. Hence, alternatives to the support–confidence framework can be useful in mining interesting data relationships.

4.3.2 From association analysis to correlation analysis

As we have seen so far, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be augmented to the support–confidence framework for association rules. This leads to *correlation rules* of the form

$$A \Rightarrow B \; [support, \; confidence, \; correlation]. \tag{4.7}$$

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B. There are many different correlation measures for us to choose. In this subsection, we study several correlation measures to determine which would be good for mining large data sets.

Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is **independent** of the occurrence of itemset B if $P(A \cup B) = P(A)P(B)$; otherwise, itemsets A and B are **dependent** and **correlated**. This definition can easily be extended to more than two itemsets. The **lift** between the occurrence of A and B can be measured by computing

$$lift\,(A, B) = \frac{P(A \cup B)}{P(A)P(B)}. \tag{4.8}$$

If the resulting value of Eq. (4.8) is less than 1, then the occurrence of A is *negatively correlated with* the occurrence of B, meaning that the occurrence of one likely leads to the absence of the other one. If the resulting value is greater than 1, then A and B are *positively correlated*, meaning that the occurrence of one implies the occurrence of the other. If the resulting value is equal to 1, then A and B are *independent*, and there is no correlation between them.

Eq. (4.8) is equivalent to $P(B|A)/P(B)$, or $conf(A \Rightarrow B)/P(B)$, which is also referred to as the *lift* of the association (or correlation) rule $A \Rightarrow B$. In other words, it assesses the degree to which the occurrence of one "lifts" the occurrence of the other. For example, if A corresponds to the sale of computer games and B corresponds to the sale of videos, then given the current market conditions, the sale of games is said to increase or "lift" the likelihood of the sale of videos by a factor of the value returned by Eq. (4.8).

Let's go back to the computer game and video data of Example 4.7.

Example 4.8. Correlation analysis using lift. To help filter out misleading "strong" associations of the form $A \Rightarrow B$ from the data of Example 4.7, we need to study how the two itemsets, A and B, are correlated. Let $\overline{game}$ refer to the transactions of Example 4.7 that do not contain computer games, and $\overline{video}$ refer to those that do not contain videos. The transactions can be summarized in a *contingency table*, as shown in Table 4.6.

From the table, we can see that the probability of purchasing a computer game is $P(\{game\}) = 0.60$, the probability of purchasing a video is $P(\{video\}) = 0.75$, and the probability of purchasing both is $P(\{game, video\}) = 0.40$. By Eq. (4.8), the lift of Rule (4.6) is $P(\{game, video\})/(P(\{game\}) \times P(\{video\})) = 0.40/(0.60 \times 0.75) = 0.89$. Because this value is less than 1, there is a negative correlation between the occurrence of $\{game\}$ and $\{video\}$. The numerator is the likelihood of a customer purchasing both, whereas the denominator is what the likelihood would have been if the two purchases were completely independent. Such a negative correlation cannot be identified by a support–confidence framework. □

Table 4.6 2 × 2 contingency table summarizing the transactions with respect to game and video purchases.

	$game$	$\overline{game}$	Σ_{row}
$video$	4000	3500	7500
$\overline{video}$	2000	500	2500
Σ_{col}	6000	4000	10,000

Table 4.7 Table 4.6 contingency table, now with the expected values.

	$game$	$\overline{game}$	Σ_{row}
$video$	4000 (4500)	3500 (3000)	7500
$\overline{video}$	2000 (1500)	500 (1000)	2500
Σ_{col}	6000	4000	10,000

The second correlation measure that we study is the χ^2 measure, which was introduced in Chapter 3 (Eq. (3.1)). To compute the χ^2 value, we take the squared difference between the observed and expected value for a slot (A and B pair) in the contingency table, divided by the expected value. This amount is summed for all slots of the contingency table. Let's perform a χ^2 analysis of Example 4.8.

Example 4.9. Correlation analysis using χ^2. To compute the correlation using χ^2 analysis for nominal data, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 4.7. From the table, we can compute the χ^2 value as follows:

$$\chi^2 = \Sigma \frac{(observed - expected)^2}{expected} = \frac{(4000-4500)^2}{4500} + \frac{(3500-3000)^2}{3000} + \frac{(2000-1500)^2}{1500} + \frac{(500-1000)^2}{1000} = 555.6.$$

Because the χ^2 value is greater than 1, and the observed value of the slot $(game, video) = 4000$, which is less than the expected value of 4500, *buying game* and *buying video* are *negatively correlated*. This is consistent with the conclusion derived from the analysis of the *lift* measure in Example 4.8. □

4.3.3 A comparison of pattern evaluation measures

The above discussion shows that instead of using the simple support–confidence framework to evaluate frequent patterns, other measures, such as *lift* and χ^2, often disclose more intrinsic pattern relationships. How effective are these measures? Should we also consider other alternatives?

Researchers have studied many pattern evaluation measures even before the start of in-depth research on scalable methods for mining frequent patterns. In the data mining community, several other pattern evaluation measures have attracted interest. In this subsection, we present four such measures: *all_confidence*, *max_confidence*, *Kulczynski*, and *cosine*. Each of these four measures has an interesting property: the value of each measure is only influenced by the supports of A, B, and $A \cup B$, or more exactly, by the conditional probabilities of $P(A|B)$ and $P(B|A)$, but not by the total number of transactions. Another common property is that each measure ranges from 0 to 1, and the higher the value, the closer the relationship between A and B.

Given two itemsets, A and B, the **all_confidence** measure of A and B is defined as

$$all\_conf(A,B) = \frac{sup(A \cup B)}{max\{sup(A), sup(B)\}} = min\{P(A|B), P(B|A)\}, \tag{4.9}$$

where $max\{sup(A), sup(B)\}$ is the maximum support of the itemsets A and B. Thus $all\_conf(A,B)$ is also the minimum confidence of the two association rules related to A and B, namely, "$A \Rightarrow B$" and "$B \Rightarrow A$."

Given two itemsets, A and B, the **max_confidence** measure of A and B is defined as

$$max\_conf(A,B) = max\{P(A \mid B), P(B \mid A)\}. \tag{4.10}$$

The *max_conf* measure is the maximum confidence of the two association rules, "$A \Rightarrow B$" and "$B \Rightarrow A$."

Given two itemsets, A and B, the **Kulczynski** measure of A and B (abbreviated as **Kulc**) is defined as

$$Kulc\,(A,B) = \frac{1}{2}(P(A|B) + P(B|A)). \tag{4.11}$$

It was proposed in 1927 by Polish mathematician S. Kulczynski. It can be viewed as an average of two confidence measures. That is, it is the average of two conditional probabilities: the probability of itemset B given itemset A, and the probability of itemset A given itemset B.

Finally, given two itemsets, A and B, the **cosine** measure of A and B is defined as

$$\begin{aligned} cosine\,(A,B) &= \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}} \\ &= \sqrt{P(A|B) \times P(B|A)}. \end{aligned} \tag{4.12}$$

The *cosine* measure can be viewed as a *harmonized lift* measure. The two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of A and B. This is an important difference, however, because by taking the square root, the cosine value is only influenced by the supports of A, B, and $A \cup B$, and not by the total number of transactions.

Now, together with *lift* and χ^2, we have introduced in total six pattern evaluation measures. You may wonder, *"Which is the best in assessing the discovered pattern relationships?"* To answer this question, we examine their performance on some typical data sets.

Example 4.10. Comparison of six pattern evaluation measures on typical data sets. The relationships between the purchases of two items, *milk* and *coffee*, can be examined by summarizing their purchase history in Table 4.8, a 2×2 contingency table, where an entry such as *mc* represents the number of transactions containing both milk and coffee.

Table 4.8 2 × 2 contingency table for two items.

	milk	$\overline{milk}$	Σ_{row}
coffee	mc	$\overline{m}c$	c
$\overline{coffee}$	$m\overline{c}$	$\overline{m}\overline{c}$	$\overline{c}$
Σ_{col}	m	$\overline{m}$	Σ

Table 4.9 Comparison of six pattern evaluation measures using contingency tables for a variety of data sets.

Data Set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{m}\overline{c}$	χ^2	*lift*	*all_conf.*	*max_conf.*	*Kulc.*	*cosine*
D_1	10,000	1000	1000	100,000	90,557	9.26	0.91	0.91	0.91	0.91
D_2	10,000	1000	1000	100	0	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100,000	670	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100,000	24,740	25.75	0.5	0.5	0.5	0.5
D_5	1000	100	10,000	100,000	8173	9.18	0.09	0.91	0.5	0.29
D_6	1000	10	100,000	100,000	965	1.97	0.01	0.99	0.5	0.10

Table 4.9 shows a set of transactional data sets with their corresponding contingency tables and the associated values for each of the six evaluation measures. Let's first examine the first four data sets, D_1 through D_4. From the table, we see that m and c are positively associated in D_1 and D_2, negatively associated in D_3, and neutral in D_4. For D_1 and D_2, m and c are positively associated because mc (10,000) is considerably greater than $\overline{m}c$ (1000) and $m\overline{c}$ (1000). Intuitively, for people who bought milk ($m = 10{,}000 + 1000 = 11{,}000$), it is very likely that they also bought coffee ($mc/m = 10/11 = 91\%$), and vice versa.

The results of the four newly introduced measures show that m and c are strongly positively associated in both data sets by producing a measure value of 0.91. However, *lift* and χ^2 generate dramatically different measure values for D_1 and D_2 due to their sensitivity to $\overline{mc}$. In fact, in many real-world scenarios, $\overline{mc}$ is usually huge and unstable. For example, in a market basket database, the total number of transactions could fluctuate on a daily basis and overwhelmingly exceed the number of transactions containing any particular itemset. Therefore a good interestingness measure should not be affected by transactions that do not contain the itemsets of interest; otherwise, it would generate unstable results, as illustrated in D_1 and D_2.

Similarly, in D_3, the four new measures correctly show that m and c are strongly negatively associated because the mc to c ratio equals the mc to m ratio, that is, $100/1100 = 9.1\%$. However, *lift* and χ^2 both contradict this in an incorrect way: their values for D_2 are between those for D_1 and D_3.

For data set D_4, both *lift* and χ^2 indicate a highly positive association between m and c, whereas the others indicate a "neutral" association because the ratio of mc to $\overline{m}c$ equals the ratio of mc to $m\overline{c}$, which is 1. This means that if a customer buys coffee (or milk), the probability that he or she will also purchase milk (or coffee) is exactly 50%. □

"Why are lift *and* χ^2 *so poor at distinguishing pattern association relationships in the previous transactional data sets?"* To answer this, we have to consider the *null-transactions*. A **null-transaction** is a transaction that does not contain any of the itemsets being examined. In our example, $\overline{mc}$ rep-

resents the number of null-transactions. *Lift* and χ^2 have difficulty distinguishing interesting pattern association relationships because they are both strongly influenced by $\overline{mc}$. Typically, the number of null-transactions can outweigh the number of individual purchases because, for example, many people may buy neither milk nor coffee. On the other hand, the other four measures are good indicators of interesting pattern associations because their definitions remove the influence of $\overline{mc}$ (i.e., they are not influenced by the number of null-transactions).

This discussion shows that it is highly desirable to have a measure that is independent of the number of null-transactions. A measure is **null-invariant** if its value is free from the influence of null-transactions. Null-invariance is an important property for measuring association patterns in large transaction databases. Among the six discussed measures in this subsection, only *lift* and χ^2 are not null-invariant measures.

"*Among the* all_confidence, max_confidence, Kulczynski, *and* cosine *measures, which is best at indicating interesting pattern relationships*?"

To answer this question, we introduce the **imbalance ratio** (**IR**), which assesses the imbalance of two itemsets, A and B, in rule implications. It is defined as

$$IR(A, B) = \frac{|sup(A) - sup(B)|}{sup(A) + sup(B) - sup(A \cup B)}, \tag{4.13}$$

where the numerator is the absolute value of the difference between the support of the itemsets A and B, and the denominator is the number of transactions containing A or B. If the two directional implications between A and B are the same, then $IR(A, B)$ will be zero. Otherwise, the larger the difference between the two, the larger the imbalance ratio. This ratio is independent of the number of null-transactions and independent of the total number of transactions.

Let's continue examining the remaining data sets in Example 4.10.

Example 4.11. Comparing null-invariant measures in pattern evaluation. Although the four measures introduced in this section are null-invariant, they may present dramatically different values on some subtly different data sets. Let's examine data sets D_5 and D_6, shown earlier in Table 4.9, where the two events m and c have unbalanced conditional probabilities. That is, the ratio of mc to c is greater than 0.9. This means that knowing that c occurs should strongly suggest that m occurs also. The ratio of mc to m is less than 0.1, indicating that m implies that c is quite unlikely to occur. The *all_confidence* and *cosine* measures view both cases as negatively associated and the *Kulc* measure views both as neutral. The *max_confidence* measure claims strong positive associations for these cases. The measures give very diverse results!

"*Which measure intuitively reflects the true relationship between the purchase of milk and coffee?*" Actually, in this case, it is difficult to argue whether the two data sets have positive or negative association. From one point of view, only $mc/(mc + m\overline{c}) = 1000/(1000 + 10{,}000) = 9.09\%$ of milk-related transactions contain coffee in D_5, and this percentage is $1000/(1000 + 100{,}000) = 0.99\%$ in D_6, both indicating a negative association. On the other hand, 90.9% of transactions in D_5 (i.e., $mc/(mc + \overline{m}c) = 1000/(1000 + 100)$) and 9% in D_6 (i.e., $1000/(1000 + 10)$) containing coffee contain milk as well, which indicates a positive association between milk and coffee, a very different conclusion.

In this case, it is fair to treat it as neutral, as *Kulc* does. In the meantime, it will be good to also indicate its skewness using the *imbalance ratio (IR)*. According to Eq. (4.13), for D_4 we have

$IR(m, c) = 0$, a perfectly balanced case; for D_5, $IR(m, c) = 0.89$, a rather imbalanced case; whereas for D_6, $IR(m, c) = 0.99$, a very skewed case. Therefore the two measures, *Kulc* and IR, work together, presenting a clear picture for all three data sets, D_4 through D_6. □

In summary, the use of only support and confidence measures to mine associations may generate a large number of rules, many of which can be uninteresting to users. Instead, we can augment the support–confidence framework with a pattern interestingness measure, which helps focus the mining toward rules with strong pattern relationships. The added measure substantially reduces the number of rules generated and leads to the discovery of more meaningful rules. Besides those introduced in this section, many other interestingness measures have been studied in the literature. Unfortunately, most of them do not have the null-invariance property. Because large data sets typically have many null-transactions, it is important to consider the null-invariance property when selecting appropriate interestingness measures for pattern evaluation. Among the four null-invariant measures studied here, namely *all_confidence*, *max_confidence*, *Kulc*, and *cosine*, we recommend using *Kulc* in conjunction with the imbalance ratio.

4.4 Summary

- The discovery of frequent patterns, associations, and correlation relationships among huge amounts of data is useful in selective marketing, decision analysis, and business management. A popular area of application is **market basket analysis**, which studies customers' buying habits by searching for itemsets that are frequently purchased together (or in sequence).
- **Association rule mining** consists of first finding **frequent itemsets** (sets of items, such as A and B, satisfying a *minimum support threshold*, or percentage of the task-relevant tuples), from which **strong** association rules in the form of $A \Rightarrow B$ are generated. These rules also satisfy a *minimum confidence threshold* (a prespecified probability of satisfying B under the condition that A is satisfied). Associations can be further analyzed to uncover **correlation rules**, which convey statistical correlations between itemsets A and B.
- Many efficient and scalable algorithms have been developed for **frequent itemset mining**, from which association and correlation rules can be derived. These algorithms can be classified into three categories: (1) *Apriori-like algorithms*, (2) *frequent pattern growth–based algorithms* such as FP-growth, and (3) *algorithms that use the vertical data format*.
- The **Apriori algorithm** is a seminal algorithm for mining frequent itemsets for Boolean association rules. It explores the level-wise mining Apriori property that *all nonempty subsets of a frequent itemset must also be frequent.* At the kth iteration (for $k \geq 2$), it forms frequent k-itemset candidates based on the frequent $(k-1)$-itemsets, and scans the database once to find the *complete* set of frequent k-itemsets, L_k.

  Variations involving hashing and transaction reduction can be used to make the procedure more efficient. Other variations include partitioning the data (mining on each partition and then combining the results) and sampling the data (mining on a data subset). These variations can reduce the number of data scans required to as little as two or even one.
- **Frequent pattern growth** is a method of mining frequent itemsets without candidate generation. It constructs a highly compact data structure (an *FP-tree*) to compress the original transaction database. Rather than employing the generate-and-test strategy of Apriori-like methods, it focuses

on frequent pattern (fragment) growth, which avoids costly candidate generation, resulting in greater efficiency.

- **Mining frequent itemsets using the vertical data format (Eclat)** is a method that transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical data format of *item-TID_set*. It mines the transformed data set by *TID_set* intersections based on the Apriori property and additional optimization techniques such as *diffset*.
- Not all strong association rules are interesting. Therefore, the support–confidence framework should be augmented with a pattern evaluation measure, which promotes the mining of *interesting* rules. A measure is **null-invariant** if its value is free from the influence of **null-transactions** (i.e., the *transactions that do not contain any of the itemsets being examined*). Among many pattern evaluation measures, we examined ***lift***, χ^2, ***all_confidence, max_confidence, Kulczynski***, and ***cosine***, and showed that only the latter four are null-invariant. We suggest using the Kulczynski measure, together with the imbalance ratio, to present pattern relationships among itemsets.

4.5 Exercises

4.1. Suppose you have the set $\mathcal{C}$ of all frequent closed itemsets on a data set D, as well as the support count for each frequent closed itemset. Describe an algorithm to determine whether a given itemset X is frequent or not, and the support of X if it is frequent.

4.2. An itemset X is called a *generator* on a data set D if there does not exist a proper subitemset $Y \subset X$ such that $support(X) = support(Y)$. A generator X is a *frequent generator* if $support(X)$ passes the minimum support threshold. Let $\mathcal{G}$ be the set of all frequent generators on a data set D.

a. Can you determine whether an itemset A is frequent and the support of A, if it is frequent, using only $\mathcal{G}$ and the support counts of all frequent generators? If yes, present your algorithm. Otherwise, what other information is needed? Can you give an algorithm assuming the information needed is available?

b. What is the relationship between closed itemsets and generators?

4.3. The Apriori algorithm makes use of *prior knowledge* of subset support properties.

a. Prove that all nonempty subsets of a frequent itemset must also be frequent.

b. Prove that the support of any nonempty subset s' of itemset s must be at least as great as the support of s.

c. Given frequent itemset l and subset s of l, prove that the confidence of the rule "$s' \Rightarrow (l - s')$" cannot be more than the confidence of "$s \Rightarrow (l - s)$," where s' is a subset of s.

d. A *partitioning* variation of Apriori subdivides the transactions of a database D into n nonoverlapping partitions. Prove that any itemset that is frequent in D must be frequent in at least one partition of D.

4.4. Let c be a candidate itemset in C_k generated by the Apriori algorithm. How many length-$(k-1)$ subsets do we need to check in the prune step? Per your previous answer, can you give an improved version of procedure `has_infrequent_subset` in Fig. 4.4?

4.5. Section 4.2.2 describes a method for *generating association rules* from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed there. (*Hint:* consider incorporating the properties of Exercises 4.3(b), (c) into your design.)