6.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules

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

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

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

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

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

From Eq. (6.3), we have

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.2 Frequent Itemset Mining Methods

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

6.2.1 Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules [AS94b]. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where k-itemsets are used to explore $(k+1)$-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and

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 6.6 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}. 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.

6.2.6 Mining Closed and Max Patterns

In Section 6.1.2 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 6.2 showed that closed frequent itemsets[9] 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 6.2, 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 6.2's data set.

A recommended methodology is to search for closed frequent itemsets directly during the mining process. This requires us to prune the search space as soon as we

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

can identify the case of closed itemsets during mining. Pruning strategies include the following:

Item 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 6.2 of Example 6.5, 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}.

Sub-itemset pruning: *If a frequent itemset* X *is a proper subset of an already found frequent closed itemset* Y *and* support_count(X)=support_count(Y)*, then* X *and all of* X*'s descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned.*

Similar to Example 6.2, suppose a transaction database has only two transactions: $\{\langle a_1, a_2, \ldots, a_{100}\rangle, \langle a_1, a_2, \ldots, a_{50}\rangle\}$, and the minimum support count is $min\_sup = 2$. The projection on the first item, a_1, derives the frequent itemset, $\{a_1, a_2, \ldots, a_{50} : 2\}$, based on the *itemset merging* optimization. Because $support(\{a_2\}) = support(\{a_1, a_2, \ldots, a_{50}\}) = 2$, and $\{a_2\}$ is a proper subset of $\{a_1, a_2, \ldots, a_{50}\}$, there is no need to examine a_2 and its projected database. Similar pruning can be done for $a_3, \ldots, a_{50}$ as well. Thus, the mining of closed frequent itemsets in this data set terminates after mining a_1's projected database.

Item skipping: *In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset* X *associated with a header table and a projected database. If a local frequent item* p *has the same support in several header tables at different levels, we can safely prune* p *from the header tables at higher levels.*

Consider, for example, the previous transaction database having only two transactions: $\{\langle a_1, a_2, \ldots, a_{100}\rangle, \langle a_1, a_2, \ldots, a_{50}\rangle\}$, where $min\_sup = 2$. Because a_2 in a_1's projected database has the same support as a_2 in the global header table, a_2 can be pruned from the global header table. Similar pruning can be done for $a_3, \ldots, a_{50}$. There is no need to mine anything more after mining a_1's projected database.

Besides pruning the search space in the closed itemset mining process, another important optimization is to perform efficient checking of each newly derived frequent itemset to see whether it is closed. This is because the mining process cannot ensure that every generated frequent itemset is closed.

When a new frequent itemset is derived, it is necessary to perform two kinds of closure checking: (1) *superset checking*, which checks if this new frequent itemset is a superset of some already found closed itemsets with the same support, and (2) *subset checking*, which checks whether the newly found itemset is a subset of an already found closed itemset with the same support.

If we adopt the *item merging* pruning method under a divide-and-conquer framework, then the superset checking is actually built-in and there is no need to explicitly

perform superset checking. This is because if a frequent itemset $X \cup Y$ is found later than itemset X, and carries the same support as X, it must be in X's projected database and must have been generated during itemset merging.

To assist in subset checking, a compressed **pattern-tree** can be constructed to maintain the set of closed itemsets mined so far. The pattern-tree is similar in structure to the FP-tree except that all the closed itemsets found are stored explicitly in the corresponding tree branches. For efficient subset checking, we can use the following property: *If the current itemset* S_c *can be subsumed by another already found closed itemset* S_a*, then (1)* S_c *and* S_a *have the same support, (2) the length of* S_c *is smaller than that of* S_a*, and (3) all of the items in* S_c *are contained in* S_a.

Based on this property, a **two-level hash index structure** can be built for fast accessing of the pattern-tree: The first level uses the identifier of the last item in S_c as a hash key (since this identifier must be within the branch of S_c), and the second level uses the support of S_c as a hash key (since S_c and S_a have the same support). This will substantially speed up the subset checking process.

This discussion illustrates methods for efficient mining of closed frequent itemsets. *"Can we extend these methods for efficient mining of maximal frequent itemsets?"* Because maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed here can be extended to mining maximal frequent itemsets. However, we leave this method as an exercise for interested readers.

6.3 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 the users. Unfortunately, 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 6.3.1). We then discuss how the support–confidence framework can be supplemented with additional interestingness measures based on *correlation analysis* (Section 6.3.2). Section 6.3.3 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.

6.3.1 Strong Rules Are Not Necessarily Interesting

Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being