Chapter 5

Association Pattern Mining: Advanced Concepts

"Each child is an adventure into a better life—an opportunity to change the old pattern and make it new."—Hubert H. Humphrey

5.1 Introduction

Association pattern mining algorithms often discover a large number of patterns, and it is difficult to use this large output for application-specific tasks. One reason for this is that a vast majority of the discovered associations may be uninteresting or redundant for a specific application. This chapter discusses a number of advanced methods that are designed to make association pattern mining more application-sensitive:

1. *Summarization:* The output of association pattern mining is typically very large. For an end-user, a smaller set of discovered itemsets is much easier to understand and assimilate. This chapter will introduce a number of summarization methods such as finding maximal itemsets, closed itemsets, or nonredundant rules.

2. *Querying:* When a large number of itemsets are available, the users may wish to query them for smaller summaries. This chapter will discuss a number of specialized summarization methods that are query friendly. The idea is to use a two-phase approach in which the data is preprocessed to create a summary. This summary is then queried.

3. *Constraint incorporation:* In many real scenarios, one may wish to incorporate application-specific constraints into the itemset generation process. Although a constraint-based algorithm may not always provide online responses, it does allow for the use of much lower support-levels for mining, than a two-phase "preprocess-once query-many" approach.

These topics are all related to the extraction of interesting summary information from itemsets in different ways. For example, compressed representations of itemsets are very useful

Table 5.1: Example of a snapshot of a market basket data set (Replicated from Table 4.1 of Chap. 4)

tid	Set of items
1	$\{Bread, Butter, Milk\}$
2	$\{Eggs, Milk, Yogurt\}$
3	$\{Bread, Cheese, Eggs, Milk\}$
4	$\{Eggs, Milk, Yogurt\}$
5	$\{Cheese, Milk, Yogurt\}$

for querying. A query-friendly compression scheme is very different from a summarization scheme that is designed to assure nonredundancy. Similarly, there are fewer constrained itemsets than unconstrained itemsets. However, the shrinkage of the discovered itemsets is because of the constraints rather than a compression or summarization scheme. This chapter will also discuss a number of useful applications of association pattern mining.

This chapter is organized as follows. The problem of pattern summarization is addressed in Sect. 5.2. A discussion of querying methods for pattern mining is provided in Sect. 5.3. Section 5.4 discusses numerous applications of frequent pattern mining. The conclusions are discussed in Sect. 5.5.

5.2 Pattern Summarization

Frequent itemset mining algorithms often discover a large number of patterns. The size of the output creates challenges for users to assimilate the results and make meaningful inferences. An important observation is that the vast majority of the generated patterns are often redundant. This is because of the downward closure property, which ensures that all subsets of a frequent itemset are also frequent. There are different kinds of compact representations in frequent pattern mining that retain different levels of knowledge about the true set of frequent patterns and their support values. The most well-known representations are those of *maximal* frequent itemsets, *closed* frequent itemsets, and other approximate representations. These representations vary in the degree of information loss in the summarized representation. Closed representations are fully lossless with respect to the support and membership of itemsets. Maximal representations are lossy with respect to the support but lossless with respect to membership of itemsets. Approximate condensed representations are lossy with respect to both but often provide the best practical alternative in application-driven scenarios.

5.2.1 Maximal Patterns

The concept of maximal itemsets was discussed briefly in the previous chapter. For convenience, the definition of maximal itemsets is restated here:

Definition 5.2.1 (Maximal Frequent Itemset) *A frequent itemset is maximal at a given minimum support level minsup if it is frequent and no superset of it is frequent.*

For example, consider the example of Table 5.1, which is replicated from the example of Table 4.1 in the previous chapter. It is evident that the itemset $\{Eggs, Milk, Yogurt\}$ is frequent at a minimum support level of 2 and is also maximal. The support of proper subsets of a maximal itemset is always equal to, or strictly larger than the latter because of the support-monotonicity property. For example, the support of $\{Eggs, Milk\}$, which is a proper subset of the itemset $\{Eggs, Milk, Yogurt\}$, is 3. Therefore, one strategy for summarization is to mine only the maximal itemsets. The remaining itemsets are derived as subsets of the maximal itemsets.

Although all the itemsets can be derived from the maximal itemsets with the subsetting approach, their support values cannot be derived. Therefore, maximal itemsets are lossy because they do not retain information about the support values. To provide a lossless representation in terms of the support values, the notion of *closed itemset mining* is used. This concept will be discussed in the next section.

A trivial way to find all the maximal itemsets would be to use any frequent itemset mining algorithm to find *all* itemsets. Then, only the maximal ones can be retained in a postprocessing phase by examining itemsets in decreasing order of length, and removing proper subsets. This process is continued until all itemsets have either been examined or removed. The itemsets that have not been removed at termination are the maximal ones. However, this approach is an inefficient solution. When the itemsets are very long, the number of maximal frequent itemsets may be orders of magnitude smaller than the number of frequent itemsets. In such cases, it may make sense to design algorithms that can directly prune parts of the search space of patterns during frequent itemset discovery. Most of the tree-enumeration methods can be modified with the concept of *lookaheads* to prune the search space of patterns. This notion is discussed in the previous chapter in the context of the *DepthProject* algorithm.

Although the notion of lookaheads is described in the Chap. 4, it is repeated here for completeness. Let P be a frequent pattern in the enumeration tree of itemsets, and $F(P)$ represent the set of candidate extensions of P in the enumeration tree. Then, if $P \cup F(P)$ is a subset of a frequent pattern that has already been found, then it implies that the entire enumeration tree rooted at P is frequent and can, therefore, be removed from further consideration. In the event that the subtree is not pruned, the candidate extensions of P need to be counted. During counting, the support of $P \cup F(P)$ is counted at the same time that the supports of single-item candidate extensions of P are counted. If $P \cup F(P)$ is frequent then the subtree rooted at P can be pruned as well. The former kind of *subset-based* pruning approach is particularly effective with depth-first methods. This is because maximal patterns are found much earlier with a depth-first strategy than with a breadth-first strategy. For a maximal pattern of length k, the depth-first approach discovers it after exploring only $(k-1)$ of its prefixes, rather than the 2^k possibilities. This maximal pattern then becomes available for subset-based pruning. The remaining subtrees containing subsets of $P \cup F(P)$ are then pruned. The superior lookahead-pruning of depth-first methods was first noted in the context of the *DepthProject* algorithm.

The pruning approach provides a smaller set of patterns that includes *all* maximal patterns but may also include some nonmaximal patterns despite the pruning. Therefore, the approach discussed above may be applied to remove these nonmaximal patterns. Refer to the bibliographic notes for pointers to various maximal frequent pattern mining algorithms.

5.2.2 Closed Patterns

A simple definition of a closed pattern, or closed itemset, is as follows:

Definition 5.2.2 (Closed Itemsets) *An itemset X is closed, if none of its supersets have exactly the same support count as X.*

Closed frequent pattern mining algorithms require itemsets to be closed in addition to being frequent. So why are closed itemsets important? Consider a closed itemset X, and the set $\mathcal{S}(X)$ of itemsets which are subsets of X, and which have the same support as X. The only itemset from $\mathcal{S}(X)$ that will be returned by a closed frequent itemset mining algorithm, will be X. The itemsets contained in $\mathcal{S}(X)$ may be referred to as the *equi-support* subsets of X. An important observation is as follows:

Observation 5.2.1 *Let X be a closed itemset, and $\mathcal{S}(X)$ be its equi-support subsets. For any itemset $Y \in \mathcal{S}(X)$, the set of transactions $\mathcal{T}(Y)$ containing Y is exactly the same. Furthermore, there is no itemset Z outside $\mathcal{S}(X)$ such that the set of transactions in $\mathcal{T}(Z)$ is the same as $\mathcal{T}(X)$.*

This observation follows from the downward closed property of frequent itemsets. For any proper subset Y of X, the set of transactions $\mathcal{T}(Y)$ is always a superset of $\mathcal{T}(X)$. However, if the support values of X and Y are the same, then $\mathcal{T}(X)$ and $\mathcal{T}(Y)$ are the same, as well. Furthermore, if any itemset $Z \notin \mathcal{S}(X)$ yields $\mathcal{T}(Z) = \mathcal{T}(X)$, then the support of $Z \cup X$ must be the same as that of X. Because Z is not a subset of X, $Z \cup X$ must be a proper superset of X. This would lead to a contradiction with the assumption that X is closed.

It is important to understand that the itemset X encodes information about *all* the nonredundant counting information needed with respect to any itemset in $\mathcal{S}(X)$. *Every itemset in $\mathcal{S}(X)$ describes the same set of transactions, and therefore, it suffices to keep the single representative itemset.* The maximal itemset X from $\mathcal{S}(X)$ is retained. It should be pointed out that Definition 5.2.2 is a simplification of a more formal definition that is based on the use of a set-closure operator. The formal definition with the use of a set-closure operator is directly based on Observation 5.2.1 (which was derived here from the simplified definition). The informal approach used by this chapter is designed for better understanding. The frequent closed itemset mining problem is defined below.

Definition 5.2.3 (Closed Frequent Itemsets) *An itemset X is a closed frequent itemset at minimum support minsup, if it is both closed and frequent.*

The set of closed itemsets can be discovered in two ways:

1. The set of frequent itemsets at any given minimum support level may be determined, and the closed frequent itemsets can be derived from this set.

2. Algorithms can be designed to directly find the closed frequent patterns during the process of frequent pattern discovery.

While the second class of algorithms is beyond the scope of this book, a brief description of the first approach for finding all the closed itemsets will be provided here. The reader is referred to the bibliographic notes for algorithms of the second type.

A simple approach for finding frequent closed itemsets is to first partition all the frequent itemsets into equi-support groups. The maximal itemsets from each equi-support group may be reported. Consider a set of frequent patterns $\mathcal{F}$, from which the closed frequent patterns need to be determined. The frequent patterns in $\mathcal{F}$ are processed in increasing order of support and either ruled in or ruled out, depending on whether or not they are closed. Note that an increasing support ordering also ensures that closed patterns are encountered earlier than their redundant subsets. Initially, all patterns are unmarked. When an unmarked pattern $X \in \mathcal{F}$ is processed (based on the increasing support order selection), it is added to the frequent closed set $\mathcal{CF}$. The proper subsets of X with the same support cannot be closed. Therefore, all the proper subsets of X with the same support are marked. To achieve this goal, the subset of the itemset lattice representing $\mathcal{F}$ can be traversed in depth-first or breadth-first order starting at X, and exploring subsets of X. Itemsets that are subsets of X are marked when they have the same support as X. The traversal process backtracks when an itemset is reached with strictly larger support, or the itemset has already been marked by the current or a previous traversal. After the traversal is complete, the next unmarked node is selected for further exploration and added to $\mathcal{CF}$. The entire process of marking nodes is repeated, starting from the pattern newly added to $\mathcal{CF}$. At the end of the process, the itemsets in $\mathcal{CF}$ represent the frequent closed patterns.

5.2.3 Approximate Frequent Patterns

Approximate frequent pattern mining schemes are almost always lossy schemes because they do not retain all the information about the itemsets. The approximation of the patterns may be performed in one of the following two ways:

1. *Description in terms of transactions:* The closure property provides a lossless description of the itemsets in terms of their membership in transactions. A generalization of this idea is to allow "almost" closures, where the closure property is not exactly satisfied but is approximately specified. Thus, a "play" is allowed in the support values of the closure definition.

2. *Description in terms of itemsets themselves:* In this case, the frequent itemsets are clustered, and representatives can be drawn from each cluster to provide a concise summary. In this case, the "play" is allowed in terms of the distances between the representatives and remaining itemsets.

These two types of descriptions yield different insights. One is defined in terms of *transaction membership*, whereas the other is defined in terms of the *structure of the itemset*. Note that among the subsets of a 10-itemset X, a 9-itemset may have a much higher support, but a 1-itemset may have exactly the same support as X. In the first definition, the 10-itemset and 1-itemset are "almost" redundant with respect to each other in terms of transaction membership. In the second definition, the 10-itemset and 9-itemset are almost redundant with respect to each other in terms of itemset structure. The following sections will introduce methods for discovering each of these kinds of itemsets.

5.2.3.1 Approximation in Terms of Transactions

The closure property describes itemsets in terms of transactions, and the equivalence of different itemsets with this criterion. The notion of "approximate closures" is a generalization of this criterion. There are multiple ways to define "approximate closure," and a simpler definition is introduced here for ease in understanding.

In the earlier case of exact closures, one chooses the maximal supersets at a particular support value. In approximate closures, one does not necessarily choose the maximal supersets at a particular support value but allows a "play" δ, within a range of supports. Therefore, all frequent itemsets $\mathcal{F}$ can be segmented into a disjoint set of k "almost equi-support" groups $\mathcal{F}_1 \ldots \mathcal{F}_k$, such that for any pair of itemsets X, Y within any group $\mathcal{F}_i$, the value of $|sup(X) - sup(Y)|$ is at most δ. From each group, $\mathcal{F}_i$, only the maximal frequent representatives are reported. Clearly, when δ is chosen to be 0, this is exactly the set of closed itemsets. If desired, the exact error value obtained by removing individual items from approximately closed itemsets is also stored. There is, of course, still some uncertainty in support values because the support values of itemsets obtained by removing *two* items cannot be *exactly* inferred from this additional data.

Note that the "almost equi-support" groups may be constructed in many different ways when $\delta > 0$. This is because the ranges of the "almost equi-support" groups need not exactly be δ but can be less than δ. Of course, a greedy way of choosing the ranges is to always pick the itemset with the lowest support and add δ to it to pick the upper end of the range. This process is repeated to construct all the ranges. Then, the frequent closed itemsets can be extracted on the basis of these ranges.

The algorithm for finding frequent "almost closed" itemsets is very similar to that of finding frequent closed itemsets. As in the previous case, one can partition the frequent itemsets into almost equi-support groups, and determine the maximal ones among them. A traversal algorithm in terms of the graph lattice is as follows.

The first step is to decide the different ranges of support for the "almost equi-support" groups. The itemsets in $\mathcal{F}$ are processed groupwise in increasing order of support ranges for the "almost equi-support" groups. Within a group, unmarked itemsets are processed in increasing order of support. When these nodes are examined they are added to the almost closed set $\mathcal{AC}$. When a pattern $X \in \mathcal{F}$ is examined, all its proper subsets within the same group are marked, unless they have already been marked. To achieve this goal, the subset of the itemset lattice representing $\mathcal{F}$ can be traversed in the same way as discussed in the previous case of (exactly) closed sets. This process is repeated with the next unmarked node. At the end of the process, the set $\mathcal{AC}$ contains the frequent "almost closed" patterns. A variety of other ways of defining "almost closed" itemsets are available in the literature. The bibliographic notes contain pointers to these methods.

5.2.3.2 Approximation in Terms of Itemsets

The approximation in terms of itemsets can also be defined in many different ways and is closely related to clustering. Conceptually, the goal is to create clusters from the set of frequent itemsets $calF$, and pick representatives $\mathcal{J} = J_1 \dots J_k$ from the clusters. Because clusters are always defined with respect to a distance function $Dist(X, Y)$ between itemsets X and Y, the notion of δ-approximate sets is also based on a distance function.

Definition 5.2.4 (δ-Approximate Sets) *The set of representatives $\mathcal{J} = \{J_1 \dots J_k\}$ is δ-approximate, if for each frequent pattern $X \in \mathcal{F}$, and each $J_i \in \mathcal{J}$, the following is true:*

$$Dist(X, J_i) \leq \delta \tag{5.1}$$

Any distance function for set-valued data, such as the Jaccard coefficient, may be used. Note that the cardinality of the set k defines the level of compression. Therefore, the goal is to determine the smallest value of k for a particular level of compression δ. This objective is closely related to the partition-based formulation of clustering, in which the value of k is fixed, and the average distance of the individual objects to their representatives are optimized. Conceptually, this process also creates a clustering on the frequent itemsets. The frequent itemsets can be either strictly partitioned to their closest representative, or they can be allowed to belong to multiple sets for which their distance to the closest representative is at most δ.

So, how can the optimal size of the representative set be determined? It turns out that a simple greedy solution is very effective in most scenarios. Let $\mathcal{C}(\mathcal{J}) \subseteq \mathcal{F}$ denote the set of frequent itemsets *covered* by the representatives in $\mathcal{J}$. An itemset in $\mathcal{F}$ is said to be covered by a representative in $\mathcal{J}$, if it lies within a distance of at most δ from at least one representative of $\mathcal{J}$. Clearly, it is desirable to determine $\mathcal{J}$ so that $\mathcal{C}(\mathcal{J}) = \mathcal{F}$ and the size of the set $\mathcal{J}$ is as small as possible.

The idea of the greedy algorithm is to start with $\mathcal{J} = \{\}$ and add the first element from $\mathcal{F}$ to $\mathcal{J}$ that covers the maximum number of itemsets in $\mathcal{F}$. The covered itemsets are then removed from $\mathcal{F}$. This process is repeated iteratively by greedily adding more elements to $\mathcal{J}$ to maximize coverage in the residual set $\mathcal{F}$. The process terminates when the set $\mathcal{F}$ is empty. It can be shown that the function $f(\mathcal{J}) = |\mathcal{C}(\mathcal{J})|$ satisfies the *submodularity* property with respect to the argument $\mathcal{J}$. In such cases, greedy algorithms are generally effective in practice. In fact, in a minor variation of this problem in which $|\mathcal{C}(J)|$ is directly optimized for fixed size of J, a theoretical bound can also be established on the quality achieved by the greedy algorithm. The reader is referred to the bibliographic notes for pointers on submodularity.

5.3 Pattern Querying

Although the compression approach provides a concise summary of the frequent itemsets, there may be scenarios in which users may wish to query the patterns with specific properties. The query responses provide the *relevant* sets of patterns in an application. This relevant set is usually much smaller than the full set of patterns. Some examples are as follows:

1. Report all the frequent patterns containing X that have a minimum support of *minsup*.
2. Report all the association rules containing X that have a minimum support of *minsup* and a minimum confidence of *minconf*.

One possibility is to exhaustively scan all the frequent itemsets and report the ones satisfying the user-specified constraints. This is, however, quite inefficient when the number of frequent patterns is large. There are two classes of methods that are frequently used for querying interesting subsets of patterns:

1. *Preprocess-once query-many paradigm:* The first approach is to mine all the itemsets at a low level of support and arrange them in the form of a hierarchical or lattice data structure. Because the first phase needs to be performed only once in offline fashion, sufficient computational resources may be available. Therefore, a low level of support is used to maximize the number of patterns preserved in the first phase. Many queries can be addressed in the second phase with the summary created in the first phase.
2. *Constraint-based pattern mining:* In this case, the user-specified constraints are pushed directly into the mining process. Although such an approach can be slower for each query, it allows the mining of patterns at much lower values of the support than is possible with the first approach. This is because the constraints can reduce the pattern sizes in the intermediate steps of the itemset discovery algorithm and can, therefore, enable the discovery of patterns at much lower values of the support than an (unconstrained) preprocessing phase.

In this section, both types of methods will be discussed.

5.3.1 Preprocess-once Query-many Paradigm

This particular paradigm is very effective for the case of simpler queries. In such cases, the key is to first determine all the frequent patterns at a very low value of the support. The resulting itemsets can then be arranged in the form of a data structure for querying. The simplest data structure is the itemset lattice, which can be considered a graph data structure for querying. However, itemsets can also be queried with the use of data structures

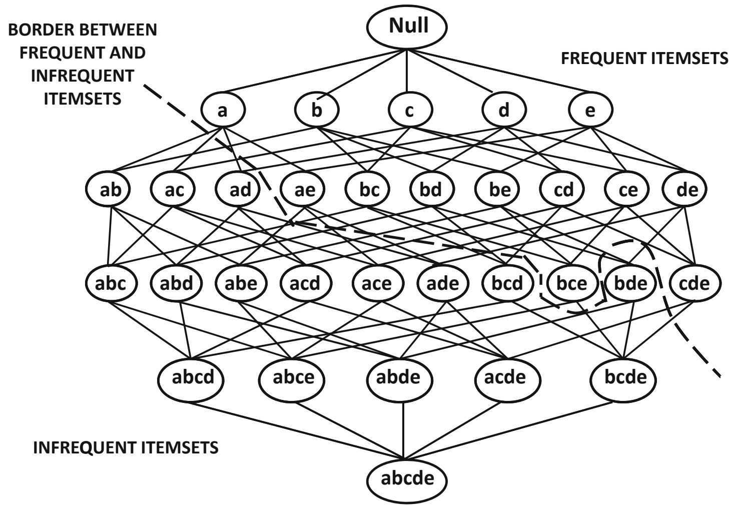

Figure 5.1: The itemset lattice (replicated from Fig. 4.1 of Chap. 4)

adapted from the information retrieval literature that use the bag-of-words representation. Both options will be explored in this chapter.

5.3.1.1 Leveraging the Itemset Lattice

As discussed in the previous chapter, the space of itemsets can be expressed as a lattice. For convenience, Fig. 4.1 of the previous chapter is replicated in Fig. 5.1. Itemsets above the dashed border are frequent, whereas itemsets below the border are infrequent.

In the preprocess-once query-many paradigm, the itemsets are mined at the lowest possible level of support s, so that a large frequent portion of the lattice (graph) of itemsets can be stored in main memory. This stage is a preprocessing phase; therefore, running time is not a primary consideration. The edges on the lattice are implemented as pointers for efficient traversal. In addition, a hash table maps the itemsets to the nodes in the graph. The lattice has a number of important properties, such as downward closure, which enable the discovery of nonredundant association rules and patterns.

This structure can effectively provide responses to many queries that are posed with support $minsup \geq s$. Some examples are as follows:

1. To determine all itemsets containing a set X at a particular level of $minsup$, one uses the hash table to map to the itemset X. Then, the lattice is traversed to determine the relevant supersets of X and report them. A similar approach can be used to determine all the frequent itemsets contained in X by using a traversal in the opposite direction.

2. It is possible to determine maximal itemsets directly during the traversal by identifying nodes that do not have edges to their immediate supersets at the user-specified minimum support level $minsup$.

3. It is possible to identify nodes within a specific hamming distance of X and a specified minimum support, by traversing the lattice structure both upward and downward from X for a prespecified number of steps.

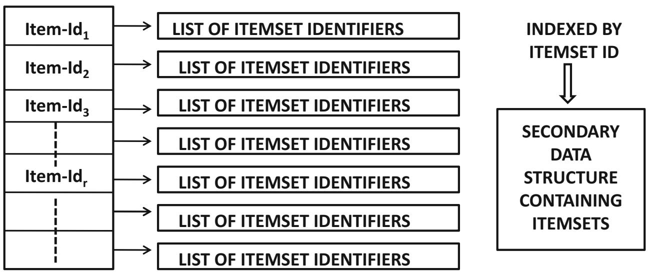
Item-Id$_1$
Item-Id$_2$
Item-Id$_3$
Item-Id$_r$
LIST OF ITEMSET IDENTIFIERS
INDEXED BY ITEMSET ID
SECONDARY DATA STRUCTURE CONTAINING ITEMSETS

Figure 5.2: Illustration of the inverted lists

It is also possible to determine nonredundant rules with the use of this approach. For example, for any itemset $Y' \subseteq Y$, the rule $X \Rightarrow Y$ has a confidence and support that is no greater than that of the rule $X \Rightarrow Y'$. Therefore, the rule $X \Rightarrow Y'$ is redundant with respect to the rule $X \Rightarrow Y$. This is referred to as *strict redundancy*. Furthermore, for any itemset I, the rule $I - Y' \Rightarrow Y'$ is redundant with respect to the rule $I - Y \Rightarrow Y$ only in terms of the confidence. This is referred to as *simple redundancy*. The lattice structure provides an efficient way to identify such nonredundant rules in terms of both simple redundancy and strict redundancy. The reader is referred to the bibliographic notes for specific search strategies on finding such rules.

5.3.1.2 Leveraging Data Structures for Querying

In some cases, it is desirable to use disk-resident representations for querying. In such cases, the memory-based lattice traversal process is likely to be inefficient. The two most commonly used data structures are the *inverted index* and the *signature table*. The major drawback in using these data structures is that they do not allow an ordered exploration of the set of frequent patterns, as in the case of the lattice structure.

The data structures discussed in this section can be used for *either* transactions or itemsets. However, some of these data structures, such as signature tables, work particularly well for itemsets because they explicitly leverage correlations between itemsets for efficient indexing. Note that correlations are more significant in the case of itemsets than raw transactions. Both these data structures are described in some detail below.

Inverted Index: The inverted index is a data structure that is used for retrieving sparse set-valued data, such as the bag-of-words representation of text. Because frequent patterns are also sparse sets drawn over a much larger universe of items, they can be retrieved efficiently with an inverted index.

Each itemset is assigned a unique *itemset-id*. This can easily be generated with a hash function. This itemset-id is similar to the *tid* that is used to represent transactions. The itemsets themselves may be stored in a secondary data structure that is indexed by the itemset-id. This secondary data structure can be a hash table that is based on the same hash function used to create the itemset-id.

The inverted list contains a list for each item. Each item points to a list of itemset-ids. This list may be stored on disk. An example of an inverted list is illustrated in Fig. 5.2. The inverted representation is particularly useful for *inclusion* queries over small sets of items. Consider a query for all itemsets containing X, where X is a small set of items. The inverted lists for each item in X is stored on the disk. The intersection of these lists is determined.

This provides the relevant itemset-ids but not the itemsets. If desired, the relevant itemsets can be accessed from disk and reported. To achieve this goal, the secondary data structure on disk needs to be accessed with the use of the recovered itemset-ids. This is an additional overhead of the inverted data structure because it may require random access to disk. For large query responses, such an approach may not be practical.

While inverted lists are effective for inclusion queries over small sets of items, they are not quite as effective for similarity queries over longer itemsets. One issue with the inverted index is that it treats each item independently, and it does not leverage the significant correlations between the items in the itemset. Furthermore, the retrieval of the full itemsets is more challenging than that of only itemset-ids. For such cases, the *signature table* is the data structure of choice.

Signature Tables: Signature tables were originally designed for indexing market basket transactions. Because itemsets have the same set-wise data structure as transactions, they can be used in the context of signature tables. Signature tables are particularly useful for sparse binary data in which there are significant correlations among the different items. Because itemsets are inherently defined on the basis of correlations, and different itemsets have large overlaps among them, signature tables are particularly useful in such scenarios.

A *signature* is a set of items. The set of items U in the original data is partitioned into sets of K signatures $S_1 \ldots S_K$, such that $U = \cup_{i=1}^{K} S_i$. The value of K is referred to as the *signature cardinality*. An itemset X is said to *activate* a signature S_i at level r if and only if $|S_i \cap X| \geq r$. This level r is referred to as the *activation threshold*. In other words, the itemset needs to have a user-specified minimum number r of items in common with the signature to activate it.

The super-coordinate of an itemset exists in K-dimensional space, where K is the signature cardinality. Each dimension of the super-coordinate has a unique correspondence with a particular signature and vice versa. The value of this dimension is 0–1, which indicates whether or not the corresponding signature is activated by that itemset. Thus, if the items are partitioned into K signatures $\{S_1, \ldots S_K\}$, then there are 2^K possible super-coordinates. Each itemset maps on to a unique super-coordinate, as defined by the set of signatures activated by that itemset. If S_{i_1}, S_{i_2}, $\ldots$ S_{i_l} be the set of signatures which an itemset activates, then the super-coordinates of that itemset are defined by setting the $l \leq K$ dimensions $\{i_1, i_2, \ldots i_l\}$ in this super-coordinate to 1 and the remaining dimensions to 0. Thus, this approach creates a many-to-one mapping, in which multiple itemsets may map into the same super-coordinate. For highly correlated itemsets, only a small number of signatures will be activated by an itemset, provided that the partitioning of U into signatures is designed to ensure that each signature contains correlated items.

The signature table contains a set of 2^K entries. One entry in the signature table corresponds to each possible super-coordinate. This creates a strict partitioning of the itemsets on the basis of the mapping of itemsets to super-coordinates. This partitioning can be used for similarity search. The signature table can be stored in main memory because the number of distinct super-coordinates can be mapped to main memory when K is small. For example, when K is chosen to be 20, the number of super-coordinates is about a million. The actual itemsets that are indexed by each entry of the signature table are stored on disk. Each entry in the signature table points to a list of pages that contain the itemsets indexed by that super-coordinate. The signature table is illustrated in Fig. 5.3.

A signature can be understood as a small category of items from the universal set of items U. Thus, if the items in each signature are closely correlated, then an itemset is likely to activate a small number of signatures. These signatures provide an idea of the

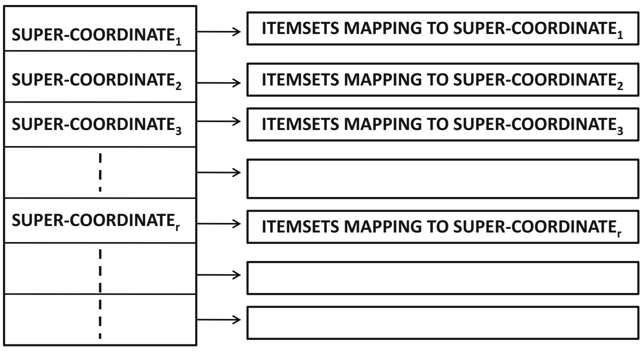
SUPER-COORDINATE$_1$
SUPER-COORDINATE$_2$
SUPER-COORDINATE$_3$
SUPER-COORDINATE$_r$
ITEMSETS MAPPING TO SUPER-COORDINATE$_1$
ITEMSETS MAPPING TO SUPER-COORDINATE$_2$
ITEMSETS MAPPING TO SUPER-COORDINATE$_3$
ITEMSETS MAPPING TO SUPER-COORDINATE$_r$

Figure 5.3: Illustration of the signature table

approximate pattern of buying behavior for that itemset. Thus, it is crucial to perform the clustering of the items into signatures so that two criteria are satisfied:

1. The items within a cluster S_i are correlated. This ensures a more *discriminative* mapping, which provides better indexing performance.

2. The aggregate support of the items within each cluster is similar. This is necessary to ensure that the signature table is balanced.

To construct the signature table, a graph is constructed so that each node of the graph corresponds to an item. For every pair of items that is frequent, an edge is added to the graph, and the weight of the edge is a function of the support of that pair of items. In addition, the weight of a node is the support of a particular item. It is desired to determine a clustering of this graph into K partitions so that the cumulative weights of edges across the partitions is as small as possible and the partitions are well balanced. Reducing the weights of edges across partitions ensures that correlated sets of items are grouped together. The partitions should be as well balanced as possible so that the itemsets mapping to each super-coordinate are as well balanced as possible. Thus, this approach transforms the items into a similarity graph, which can be clustered into partitions. A variety of clustering algorithms can be used to partition the graph into groups of items. Any of the graph clustering algorithms discussed in Chap. 19, such as *METIS*, may be used for this purpose. The bibliographic notes also contain pointers to some methods for signature table construction.

After the signatures have been determined, the partitions of the data may be defined by using the super-coordinates of the itemsets. Each itemset belongs to the partition that is defined by its super-coordinate. Unlike the case of inverted lists, the itemsets are *explicitly* stored within this list, rather than just their identifiers. This ensures that the secondary data structure does not need to be accessed to explicitly recover the itemsets. This is the reason that the signature table can be used to recover the itemsets themselves, rather than only the identifiers of the itemsets.

The signature table is capable of handling general similarity queries that cannot be efficiently addressed with inverted lists. Let x be the number of items in which an itemset matches with a target Q, and y be the number of items in which it differs with the target Q. The signature table is capable of handling similarity functions of the form $f(x, y)$ that satisfy the following two properties, for a fixed target record Q:

$$\frac{\Delta f(x,y)}{\Delta x} \geq 0 \tag{5.2}$$

$$\frac{\Delta f(x,y)}{\Delta y} \leq 0 \tag{5.3}$$

This is referred to as the *monotonicity* property. These intuitive conditions on the function ensure that it is an increasing function in the number of matches and decreasing in the hamming distance. While the match function and the hamming distance obviously satisfy these conditions, it can be shown that other functions for set-wise similarity, such as the cosine and the Jaccard coefficient, also satisfy them. For example, let P and Q be the sets of items in two itemsets, where Q is the target itemset. Then, the cosine function can be expressed as follows, in terms of x and y:

$$\begin{aligned} Cosine(P,Q) &= \frac{x}{\sqrt{|P|}\cdot\sqrt{|Q|}} \\ &= \frac{x}{\sqrt{(2\cdot x + y - |Q|)}\cdot\sqrt{|Q|}} \\ Jaccard(P,Q) &= \frac{x}{x+y} \end{aligned}$$

These functions are increasing in x and decreasing in y. These properties are important because they allow bounds to be computed on the similarity function in terms of bounds on the arguments. In other words, if γ is an upper bound on the value of x and θ is a lower bound on the value of y, then it can be shown that $f(\gamma,\theta)$ is an upper (optimistic) bound on the value of the function $f(x,y)$. This is useful for implementing a branch-and-bound method for similarity computation.

Let Q be the target itemset. Optimistic bounds on the match and hamming distance from Q to the itemsets within each super-coordinate are computed. These bounds can be shown to be a function of the target Q, the activation threshold, and the choice of signatures. The precise method for computing these bounds is described in the pointers found in the bibliographic notes. Let the optimistic bound on the match be x_i and that on distance be y_i for the ith super-coordinate. These are used to determine an optimistic bound on the similarity $f(x,y)$ between the target and any itemset indexed by the ith super-coordinate. Because of the monotonicity property, this optimistic bound for the ith super-coordinate is $B_i = f(x_i, y_i)$. The super-coordinates are sorted in decreasing (worsening) order of the optimistic bounds B_i. The similarity of Q to the itemsets that are pointed to by these super-coordinates is computed in this sorted order. The closest itemset found so far is dynamically maintained. The process terminates when the optimistic bound B_i to a super-coordinate is lower (worse) than the similarity value of the closest itemset found so far to the target. At this point, the closest itemset found so far is reported.

5.3.2 Pushing Constraints into Pattern Mining

The methods discussed so far in this chapter are designed for retrieval queries with item-specific constraints. In practice, however, the constraints may be much more general and cannot be easily addressed with any particular data structure. In such cases, the constraints may be need to be directly pushed into the mining process.

In all the previous methods, a preprocess-once query-many paradigm is used; therefore, the querying process is limited by the initial minimum support chosen during the preprocessing phase. Although such an approach has the advantage of providing online capabilities for query responses, it is sometimes not effective when the constraints result in removal of most of the itemsets. In such cases, a much lower level of minimum support may be required than could be reasonably selected during the initial preprocessing phase. The advantage of pushing the constraints into the mining process is that the constraints can be used to prune out many of the intermediate itemsets *during the execution* of the frequent pattern mining algorithms. This allows the use of much lower minimum support levels. The price for this flexibility is that the resulting algorithms can no longer be considered truly online algorithms when the data sets are very large.

Consider, for example, a scenario where the different items are tagged into different categories, such as snacks, dairy, baking products, and so on. It is desired to determine patterns, such that all items belong to the same category. Clearly, this is a constraint on the discovery of the underlying patterns. Although it is possible to first mine all the patterns, and then filter down to the relevant patterns, this is not an efficient solution. If the number of patterns mined during the preprocessing phase is no more than 10^6 and the level of selectivity of the constraint is more than 10^{-6}, then the final set returned may be empty, or too small.

Numerous methods have been developed in the literature to address such constraints directly in the mining process. These constraints are classified into different types, depending upon their impact on the mining algorithm. Some examples of well-known types of constraints, include *succinct*, *monotonic*, *antimonotonic*, and *convertible*. A detailed description of these methods is beyond the scope of this book. The bibliographic section contains pointers to many of these algorithms.

5.4 Putting Associations to Work: Applications

Association pattern mining has numerous applications in a wide variety of real scenarios. This section will discuss some of these applications briefly.

5.4.1 Relationship to Other Data Mining Problems

The association model is intimately related to other data mining problems such as classification, clustering, and outlier detection. Association patterns can be used to provide effective solutions to these data mining problems. This section will explore these relationships briefly. Many of the relevant algorithms are also discussed in the chapters on these different data mining problems.

5.4.1.1 Application to Classification

The association pattern mining problem is closely related to that of classification. *Rule-based classifiers* are closely related to association-rule mining. These types of classifiers are discussed in detail in Sect. 10.4 of Chap. 10, and a brief overview is provided here.

Consider the rule $X \Rightarrow Y$, where X is the antecedent and Y is the consequent. In associative classification, the consequent Y is a single item corresponding to the class variable, and the antecedent contains the feature variables. These rules are mined from the training data. Typically, the rules are not determined with the traditional support and confidence measures. Rather, the most *discriminative* rules with respect to the different classes need to be determined. For example, consider an itemset X and two classes c_1 and c_2. Intuitively, the itemset X is discriminative between the two classes, if the absolute difference in the confidence of the rules $X \Rightarrow c_1$ and $X \Rightarrow c_2$ is as large as possible. Therefore, the mining process should determine such discriminative rules.

Interestingly, it has been discovered, that even a relatively straightforward modification of the association framework to the classification problem is quite effective. An example of such a classifier is the CBA framework for Classification Based on Associations. More details on rule-based classifiers are discussed in Sect. 10.4 of Chap. 10.

5.4.1.2 Application to Clustering

Because association patterns determine highly correlated subsets of attributes, they can be applied to quantitative data after discretization to determine dense regions in the data. The *CLIQUE* algorithm, discussed in Sect. 7.4.1 of Chap. 7, uses discretization to transform quantitative data into binary attributes. Association patterns are discovered on the transformed data. The data points that overlap with these regions are reported as subspace clusters. This approach, of course, reports clusters that are highly overlapping with one another. Nevertheless, the resulting groups correspond to the dense regions in the data, which provide significant insights about the underlying clusters.

5.4.1.3 Applications to Outlier Detection

Association pattern mining has also been used to determine outliers in market basket data. The key idea here is that the outliers are defined as transactions that are not "covered" by most of the association patterns in the data. A transaction is said to be covered by an association pattern when the corresponding association pattern is contained in the transaction. This approach is particularly useful in scenarios where the data is high dimensional and traditional distance-based algorithms cannot be easily used. Because transaction data is inherently high dimensional, such an approach is particularly effective. This approach is discussed in detail in Sect. 9.2.3 of Chap. 9.

5.4.2 Market Basket Analysis

The prototypical problem for which the association rule mining problem was first proposed is that of market basket analysis. In this problem, it is desired to determine rules relating buying behavior of customers. The knowledge of such rules can be very useful for a retailer. For example, if an association rule reveals that the sale of beer implies a sale of diapers, then a merchant may use this information to optimize his or her shelf placement and promotion decisions. In particular, rules that are interesting or unexpected are the most informative for market basket analysis. Many of the traditional and alternative models for market basket analysis are focused on such decisions.

5.4.3 Demographic and Profile Analysis

A closely related problem is that of using demographic profiles to make recommendations. An example is the rule discussed in Sect. 4.6.3 of Chap. 4.

$$Age[85, 95] \Rightarrow Checkers$$

Other demographic attributes, such as the gender or the ZIP code, can be used to determine more refined rules. Such rules are referred to as *profile association rules*. Profile association rules are very useful for target marketing decisions because they can be used to identify relevant population segments for specific products. Profile association rules can be viewed in a similar way to classification rules, except that the antecedent of the rule typically identifies a profile segment, and the consequent identifies a population segment for target marketing.

5.4.4 Recommendations and Collaborative Filtering

Both the aforementioned applications are closely related to the generic problem of recommendation analysis and collaborative filtering. In collaborative filtering, the idea is to make recommendations to users on the basis of the buying behavior of other similar users. In this context, *localized pattern mining* is particularly useful. In localized pattern mining, the idea is to cluster the data into segments, and then determine the patterns in these segments. The patterns from each segment are typically more resistant to noise from the global data distribution and provide a clearer idea of the patterns within like-minded customers. For example, in a movie recommendation system, a particular pattern for movie titles, such as $\{\text{Gladiator}, \text{Nero}, \text{Julius Caesar}\}$, may not have sufficient support on a global basis. However, within like-minded customers, who are interested in historical movies, such a pattern may have sufficient support. This approach is used in applications such as collaborative filtering. The problem of localized pattern mining is much more challenging because of the need to simultaneously determine the clustered segments and the association rules. The bibliographic section contains pointers to such localized pattern mining methods. Collaborative filtering is discussed in detail in Sect. 18.5 of Chap. 18.

5.4.5 Web Log Analysis

Web log analysis is a common scenario for pattern mining methods. For example, the set of pages accessed during a session is no different than a market-basket data set containing transactions. When a set of Web pages is accessed together frequently, this provides useful insights about correlations in user behavior with respect to Web pages. Such insights can be leveraged by site-administrators to improve the structure of the Web site. For example, if a pair of Web pages are frequently accessed together in a session but are not currently linked together, it may be helpful to add a link between them. The most sophisticated forms of Web log analysis typically work with the temporal aspects of logs, beyond the set-wise framework of frequent itemset mining. These methods will be discussed in detail in Chaps. 15 and 18.

5.4.6 Bioinformatics

Many new technologies in bioinformatics, such as microarray and mass spectrometry technologies, allow the collection of different kinds of very high-dimensional data sets. A classical example of this kind of data is gene-expression data, which can be expressed as an $n \times d$ matrix, where the number of columns d is very large compared with typical market basket applications. It is not uncommon for a microarray application to contain a hundred thousand columns. The discovery of frequent patterns in such data has numerous applications in the discovery of key biological properties that are encoded by these data sets. For such cases, long pattern mining methods, such as maximal and closed pattern mining are very useful. In fact, a number of methods, discussed in the bibliographic notes, have specifically been designed for such data sets.

5.4.7 Other Applications for Complex Data Types

Frequent pattern mining algorithms have been generalized to more complex data types such as temporal data, spatial data, and graph data. This book contains different chapters for these complex data types. A brief discussion of these more complex applications is provided here:

1. *Temporal Web log analytics:* The use of temporal information from Web logs greatly enriches the analysis process. For example, certain patterns of accesses may occur frequently in the logs and these can be used to build event prediction models in cases where future events may be predicted from the current pattern of events.

2. *Spatial co-location patterns:* Spatial co-location patterns provide useful insights about the spatial correlations among different individuals. Frequent pattern mining algorithms have been generalized to such scenarios. Refer to Chap. 16.

3. *Chemical and biological graph applications:* In many real scenarios, such as chemical and biological compounds, the determination of structural patterns provides insights about the properties of these molecules. Such patterns are also used to create classification models. These methods are discussed in Chap. 17.

4. *Software bug analysis:* The structure of computer programs can often be represented as call graphs. The analysis of the frequent patterns in the call graphs and key deviations from these patterns provides insights about the bugs in the underlying software.

Many of the aforementioned applications will be discussed in later chapters of this book.

5.5 Summary

In order to use frequent patterns effectively in data-driven applications, it is crucial to create concise summaries of the underlying patterns. This is because the number of returned patterns may be very large and difficult to interpret. Numerous methods have been designed to create a compressed summary of the frequent patterns. Maximal patterns provide a concise summary but are lossy in terms of the support of the underlying patterns. They can often be determined effectively by incorporating different kinds of pruning strategies in frequent pattern mining algorithms.

Closed patterns provide a lossless description of the underlying frequent itemsets. On the other hand, the compression obtained from closed patterns is not quite as significant as that obtained from the use of maximal patterns. The concept of "almost closed" itemsets allows good compression, but there is some degree of information loss in the process. A different way of compressing itemsets is to cluster itemsets so that all itemsets can be expressed within a prespecified distance of particular representatives.

Query processing of itemsets is important in the context of many applications. For example, the itemset lattice can be used to resolve simple queries on itemsets. In some cases, the lattice may not fit in main memory. For these cases, it may be desirable to use disk resident data structures such as the inverted index or the signature table. In cases where the constraints are arbitrary or have a high level of selectivity, it may be desirable to push the constraints directly into the mining process.

Frequent pattern mining has many applications, including its use as a subroutine for other data mining problems. Other applications include market basket analysis, profile analysis, recommendations, Web log analysis, spatial data, and chemical data. Many of these applications are discussed in later chapters of this book.

5.6 Bibliographic Notes

The first algorithm for maximal pattern mining was proposed in [82]. Subsequently, the *DepthProject* [4] and *GenMax* [233] algorithms were also designed for maximal pattern mining. *DepthProject* showed that the depth-first method has several advantages for determining maximal patterns. Vertical bitmaps were used in *MAFIA* [123] to compress the sizes of the underlying *tid* lists. The problem of closed pattern mining was first proposed in [417] in which an *Apriori*-based algorithm, known as *A-Close*, was presented. Subsequently, numerous algorithms such as *CLOSET* [420], *CLOSET+* [504], and *CHARM* [539] were proposed for closed frequent pattern mining. The last of these algorithms uses the vertical data format to mine long patterns in a more efficient way. For the case of very high-dimensional data sets, closed pattern mining algorithms were proposed in the form of *CARPENTER* and *COBBLER*, respectively [413, 415]. Another method, known as pattern-fusion [553], fuses the different pattern segments together to create a long pattern.

The work in [125] shows how to use deduction rules to construct a minimal representation for all frequent itemsets. An excellent survey on condensed representations of frequent itemsets may be found in [126]. Numerous methods have subsequently been proposed to approximate closures in the form of δ-freesets [107]. Information-theoretic methods for itemset compression have been discussed in [470].

The use of clustering-based methods for compression focuses on the itemsets rather than the transactions. The work in [515] clusters the patterns on the basis of their similarity and frequency to create a condensed representation of the patterns. The submodularity property used in the greedy algorithm for finding the best set of covering itemsets is discussed in [403].

The algorithm for using the itemset lattice for interactive rule exploration is discussed in [37]. The concepts of simple redundancy and strict redundancy are also discussed in this work. This method was also generalized to the case of profile association rules [38]. The inverted index, presented in this chapter, may be found in [441]. A discussion of a market basket-specific implementation, together with the signature table, may be found in [41]. A compact disk structure for storing and querying frequent itemsets has been studied in [359].

A variety of constraint-based methods have been developed for pattern mining. Succinct constraints are the easiest to address because they can be pushed directly into data selection. *Monotonic constraints* need to be checked only once to restrict pattern growth [406, 332], whereas *antimonotonic constraints* need to be pushed deep into the pattern mining process. Another form of pattern mining, known as *convertible* constraints [422], can be addressed by sorting items in ascending or descending order for restraining pattern growth.

The *CLIQUE* algorithm [58] shows how association pattern mining methods may be used for clustering algorithms. The *CBA* algorithm for rule-based classification is discussed in [358]. A survey on rule-based classification methods may be found in [115]. The frequent pattern mining problem has also been used for outlier detection in very long transactions [263]. Frequent pattern mining has also been used in the field of bioinformatics [413, 415]. The determination of localized associations [27] is very useful for the problem of recommendations and collaborative filtering. Methods for mining long frequent patterns in the context of bioinformatics applications may be found in [413, 415, 553]. Association rules can also be used to discover spatial co-location patterns [388]. A detailed discussion of frequent pattern mining methods for graph applications, such as software bug analysis, and chemical and biological data, is provided in Aggarwal and Wang [26].

5.7 Exercises

1. Consider the transaction database in the table below:

tid	**items**
1	a, b, c, d
2	b, c, e, f
3	a, d, e, f
4	a, e, f
5	b, d, f

Determine all maximal patterns in this transaction database at support levels of 2, 3, and 4.

2. Write a program to determine the set of maximal patterns, from a set of frequent patterns.

3. For the transaction database of Exercise 1, determine all the closed patterns at support levels of 2, 3, and 4.

4. Write a computer program to determine the set of closed frequent patterns, from a set of frequent patterns.

5. Consider the transaction database in the table below:

tid	**items**
1	a, c, d, e
2	a, d, e, f
3	b, c, d, e, f
4	b, d, e, f
5	b, e, f
6	c, d, e
7	c, e, f
8	d, e, f

Determine all frequent maximal and closed patterns at support levels of 3, 4, and 5.

6. Write a computer program to implement the greedy algorithm for finding a representative itemset from a group of itemsets.

7. Write a computer program to implement an inverted index on a set of market baskets. Implement a query to retrieve all itemsets containing a particular set of items.

8. Write a computer program to implement a signature table on a set of market baskets. Implement a query to retrieve the closest market basket to a target basket on the basis of the cosine similarity.