Frequent Itemset Mining Methods

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate generation

The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties. 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 *Lk* 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.
A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step**: To find *Lk*, a set of **candidate** *k*-itemsets is generated by joining $L_{k\text{-}1}$ with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in $L_{k\text{-}1}$. The notation *li*[*j*] refers to the *j*th item in *li*. For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the (k-1)-itemset, this means that the items are sorted such that *li*[1] < *li*[2] < ...< *li*[*k* -1]. The join, $L_{k\text{-}1}$ $L_{k\text{-}1}$, is performed, where members of $L_{k\text{-}1}$ are joinable if their first (*k* -2) items are in common. That is, members *l*1 and *l*2 of $L_{k\text{-}1}$ are joined 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]). 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], : : : , *l*1[*k* -2], *l*1[*k* -1], *l*2[*k* -1]}.

2. **The prune step**: *Ck* is a superset of *Lk*, that is, its members may or may not be frequent, but all of the frequent *k*-itemsets are included in *Ck*. A database scan to determine the count of each candidate in *Ck* would result in the determination of *Lk* (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to *Lk*). *Ck*, however, can be huge, and so this could involve heavy computation. To reduce the size of *Ck*, 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 *Lk*-1, then the candidate cannot be frequent either and so can be removed from *Ck*.