**Unit 7**

## Association Rule Mining

Association rules are if/then statements that help uncover relationships between seemingly unrelated data in a relational database or other information repository. An example of an association rule would be "*If a customer buys a dozen eggs, he is 80% likely to also purchase milk."* An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

Association rule mining is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness. For example, the rule found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing. In addition to the above example from market basket analysis association rules are employed today in many application areas including web usage mining, intrusion detection, bioinformatics etc.

The problem of association rule mining is defined as: Let  be a set of binary attributes called *items*. Let   be a set of transactions called the *database*. Each *transaction* in has a unique transaction ID and contains a subset of the items in A *rule* is defined as an implication of the form:



Where  and 

Every rule is composed by two different set of items, also known as X and Y *itemsets* and, where X is called *antecedent* or left-hand-side (LHS) and Y is *consequent* or right-hand-side (RHS).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Transaction ID | Milk | Bread | Butter | Beer | Diaper |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 0 | 0 |

To illustrate the concepts, we use a small example from the supermarket domain. The set of items in above table shows a small database containing the items, where, in each entry, the value 1 means the presence of the item in the corresponding transaction, and the value 0 represent the absence of an item in a that transaction. An example rule for the supermarket could be meaning that if butter and bread are bought, customers also buy milk.

**Support and Confidence**

In order to select interesting rules from the set of all possible rules, constraints on various measures of significance and interest are used. The best-known constraints are minimum thresholds on *support and confidence*.

*Support* of association rule is the percentage of transactions in *dataset* that contain both items. In formula



For example, in above data-set, the association rule has a support of *2/5* since both items occurs in *40%* of all transactions (2 out of 5 transactions).

*Confidence* of association rule with respect to set of transactions *T* in *dataset D is* the percentage of transactions in D containing *A* that also contain *B*. In formula



For example, in above data-set, the association rule has a confidence of *2/3,* since *66.66%* of all transactions containing *bread* also contains *milk*.

Rules that satisfy both a minimum support threshold and a minimum confidence threshold 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.

**Why Association Mining**

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, and catalog design and store layout.

Programmers use association rules to build programs capable of machine learning

**Apriori Algorithm**

It is a classic algorithm used in data mining for learning association rules. It is very simple. Learning association rules basically means finding the items that are purchased together more frequently than others. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent item set 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 *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, presented below, is used to reduce the search space. *Apriori Property states that any subset of frequent item set must be frequent*.

***Example***



Consider a database, D, consisting of 9 transactions. Suppose min. support count required is 2 (i.e. min-sup = 2/9 = 22 %). Let minimum confidence required is 70%. We have to first find out the frequent item set using Apriori algorithm. Then, Association rules will be generated using min. support & min. confidence.

***Solution***

**Step 1**: Generating 1-itemset Frequent Pattern

The set of frequent 1-itemsets, L1, consists of the candidate 1-itemsets satisfying minimum support. In the first iteration of the algorithm, each item is a member of the set of candidate.



In the first iteration of the algorithm, each item is a member of the set of candidate.

**Step 2**: Generating 2-itemset Frequent Pattern

To discover the set of frequent 2-itemsets, L2, the algorithm uses L1 *Join* L1to generate a candidate set of 2-itemsets, C2. Next, the transactions in D are scanned and the support count for each candidate item set in C2 is accumulated (as shown in the middle table). The set of frequent 2-itemsets, L2, is then determined, consisting of those candidate 2-itemsets in C2 having minimum support.



**Step 3**: Generating 3-itemset Frequent Pattern

The generation of the set of candidate 3-itemsets, C3, involves use of the Apriori Property. In order to find C3, we compute L2*Join*L2. C3= L2 *Join*L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}. Now, Join step is complete and Prune step will be used to reduce the size of C3. Prune step helps to avoid heavy computation due to large Ck.

Based on the Apriori property that all subsets of a frequent item set must also be frequent, we can determine that four latter candidates cannot possibly be frequent. For example, let’s take {I1, I2, I3}.The 2-item subsets of it are {I1, I2}, {I1, I3} & {I2, I3}. Since all 2-item subsets of {I1, I2, I3} are members of L2, We will keep {I1, I2, I3} in C3. Lets take another example of {I2, I3, I5} which shows how the pruning is performed. The 2-item subsets are {I2, I3}, {I2, I5} & {I3,I5}. BUT, {I3, I5} is not a member of L2and hence it is not frequent violating Apriori Property. Thus we will have to remove {I2, I3, I5} from C3. Therefore, C3= {{I1, I2, I3}, {I1, I2, I5}} after checking for all members of result of Join operation for Pruning. Now, the transactions in D are scanned in order to determine L3, consisting of those candidates 3-itemsets in C3 having minimum support.



**Step 4**: Generating 4-itemset Frequent Pattern

The algorithm uses L3 *Join*L3to generate a candidate set of 4-itemsets, C4. Although the join results in {{I1, I2, I3, I5}}, this item set is pruned since its subset {{I2, I3, I5}}is not frequent. Thus, C4= φ, and algorithm terminates, having found all of the frequent items. This completes our Apriori Algorithm.

These frequent itemsets will be used to generate strong association rules ( where strong association rules satisfy both minimum support & minimum confidence).

**Step 5:** Generating Association Rules from Frequent Itemsets

**Procedure:**

For each frequent item set ***“l”,*** generate all nonempty subsets of ***l.*** For every nonempty subset ***s*** of ***l***, output the rule **“****”**if support\_count(*l*) / support\_count(s) >= **min\_conf** where min\_conf is minimum confidence threshold.

**Back To Example**

We had L = {{I1}, {I2}, {I3}, {I4}, {I5}, {I1,I2}, {I1,I3}, {I1,I5}, {I2,I3}, {I2,I4}, {I2,I5}, {I1,I2,I3}, {I1,I2,I5}}.

Let’s take ***l*** = {I1, I2, I5}. It’s all nonempty subsets are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5}.

Let minimum confidence threshold is, say 70%. The resulting association rules are shown below, each listed with its confidence.

**R1: I1 ^ I2 =>I5**

Confidence = support\_count {I1, I2,

I5}/support\_count{I1,I2} = 2/4 = 50%, R1 is Rejected.

**R2: I1 ^ I5 =>I2**

Confidence = support\_count {I1,I2,I5}/ support\_count {I1,I5} = 2/2 = 100%, R2 is Selected.

**R3: I2 ^ I5 =>I1**

Confidence = support\_count {I1,I2,I5}/ support\_count {I2,I5} = 2/2 = 100%, R3 is Selected.

**R4: I1 =>I2 ^ I5**

Confidence = support\_count {I1,I2,I5}/ support\_count {I1} = 2/6 = 33%, R4 is Rejected.

**R5: I2 =>I1 ^ I5**

Confidence = support\_count {I1,I2,I5}/ support\_count {I2} = 2/7 = 29%, R5 is Rejected.

**R6: I5 =>I1 ^ I2**

Confidence = support\_count {I1,I2,I5}/ support\_count {I5} = 2/2 = 100%, R6 is Selected.

In this way, we have found three strong association rules.

**Improving Efficiency of Apriori Algorithm**

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**

A hash-based technique can be used to reduce the size of the candidate *k*-itemsets, *Ck*, for *k* > 1. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, *L*1, from the candidate 1-itemsets in *C*1, we can generate all of the 2-itemsets for each transaction, hash them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts. A 2-itemset whose corresponding bucket count in the hash table 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 the candidate *k*-itemsets examined.



**Transaction Reduction**

It reduces the number of transactions scanned in future iterations. A transaction that does not contain any frequent *k*-itemsets cannot contain any frequent (*k*+1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for *j*-itemsets, where *j* > *k*, will not require it.

**Partitioning**

The set of transactions may be divided into a number of disjoint subsets. Then each partition is searched for frequent itemsets. These frequent itemsets are called local frequent itemsets. *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions*. Therefore, all local frequent itemsets are candidate itemsets with respect to *D*. The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to *D*.

**Sampling**

A random sample (usually large enough to fit in the main memory) may be obtained from the overall set of transactions and the sample is searched for frequent itemsets. These frequent itemsets are called *sample frequent itemsets*. 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 lessen this possibility, we use a lower support threshold than minimum support to find the frequent itemsets local to *S*