

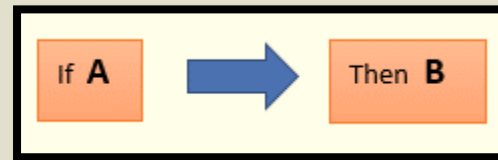


DATA MINING

Subject Incharge: Priya Sachdeva, Assistant Professor (CSE)

Association Rule Mining using Naïve Method

- Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently an itemset occurs in a transaction. A typical example is Market Based Analysis. Market Based Analysis is one of the key techniques used by large relations to show associations between items. It allows retailers to identify relationships between the items that people buy together frequently.
- Given a set of transactions, we can find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.
- Association rule learning works on the concept of If and Else Statement, such as if A, then B. Here the if element is called *Antecedent*, and then statement is called as *Consequent*. “If a customer buys bread, he’s 70% likely of buying milk.” In the above association rule, bread is the antecedent and milk is the consequent.



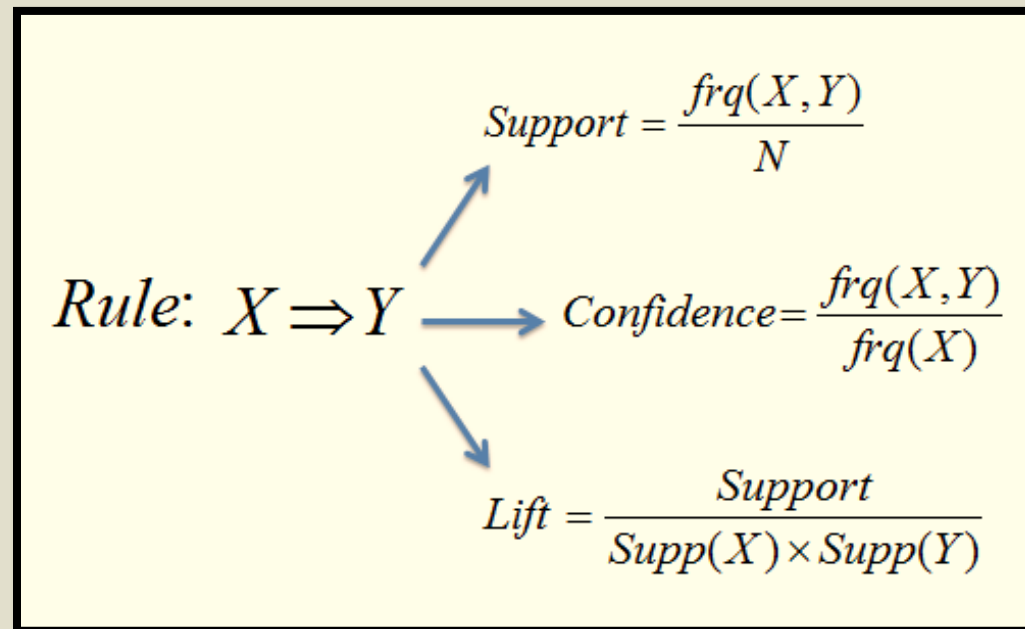
- These types of relationships where we can find out some association or relation between two items is known as single cardinality. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly.

To measure the associations between data items, the metrics are: Support, Confidence, Lift.

1. Support (S): It is a measure of how frequently the collection of items occur together as a percentage of all transactions. It is defined as the fraction of the transaction T that contains the itemset X.

2. Confidence (C): Confidence indicates how often the rule has been found to be true or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

3. Lift (I): It is the strength of any rule. It is the ratio of the observed support measure and expected support if X and Y are independent of each other.



Steps involved in Association Rule Mining

Association Rule Mining can be described as a two-step process.

Step 1: Find all frequent item sets. A set of items in a shopping basket can be referred to as an itemset. For example, [bread, butter, eggs] is an itemset from a supermarket database. A frequent itemset is one that occurs frequently in a database. The Support of an item is defined as the frequency of the item in the dataset.

$$\text{Support (Itemset)} = \frac{\text{Frequency of Itemset (Support Count)}}{\text{Total Number of Transactions}}$$

Step 2: Generate strong association rules from the frequent item sets. This uses a measure called Confidence to find strong associations.

$$\text{Confidence (A } \Rightarrow \text{ B)} = \frac{\text{Support Count(A, B)}}{\text{Support Count (A)}}$$

Apriori Algorithm

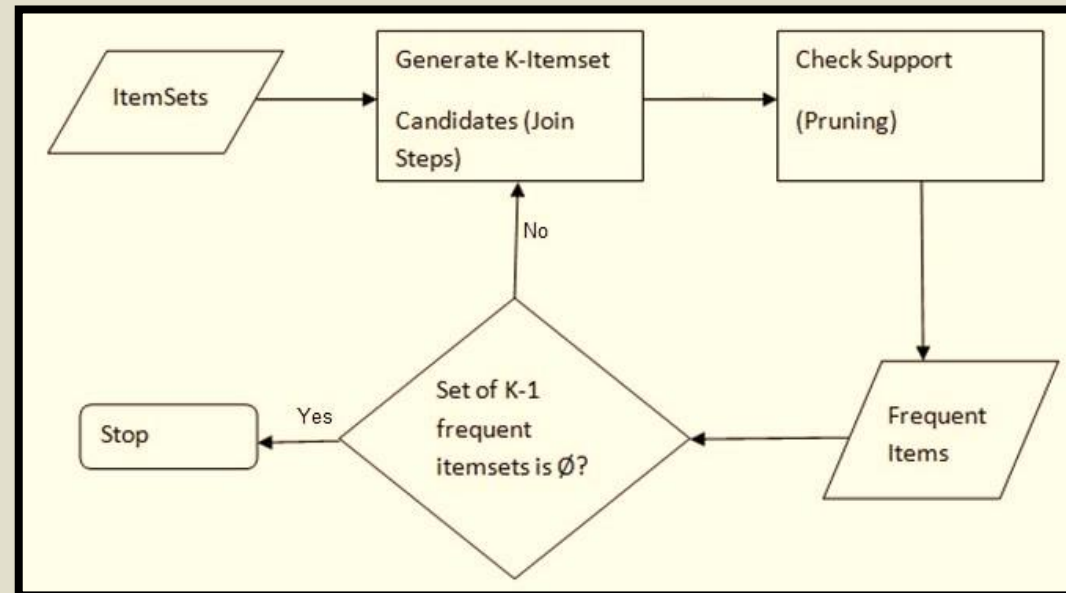
- Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store.
- It is very important for effective Market Basket Analysis and it helps the customers in purchasing their items with more ease which increases the sales of the markets.
- Its named Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k -frequent itemsets are used to find $k+1$ itemsets.

Apriori Algorithm

- To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called Apriori property which helps by reducing the search space.
- Apriori Property: The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that all subsets of a frequent itemset must be frequent. If an itemset is infrequent, all its supersets will be infrequent.

The steps followed in the Apriori Algorithm of data mining are:

1. **Generating the candidate set:** Calculate the support of item sets (of size $k=1$) in the transactional database (support is the frequency of occurrence of an itemset).
2. **Join Step:** Join the frequent itemsets to form sets of size $k + 1$ and repeat the above sets until no more itemsets can be formed.
3. **Prune Step:** This step scans the count of each item in the database. If the candidate item does not meet minimum support (threshold), then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.



Direct Hashing and Pruning (DHP)

- Apriori is an influential and well-known algorithm for mining association rules. However, the main drawback of Apriori algorithm is the large amount of candidate itemsets it generates. Several hash-based algorithms, such as DHP were proposed to deal with the problem.
- Hashing & Pruning improve the performance of traditional Apriori algorithm. It is a variation of the well-known Apriori algorithm. Hashing technique uses hash function to reduce the size of candidate item set.

Direct Hashing and Pruning (DHP)

- In the DHP algorithm, a hash table is used in order to reduce the size of the candidate $k+1$ itemsets generated at each step. It tries to reduce the number of passes made over a transactional database while keeping the number of itemsets counted in a pass relatively low. If the size of hash table is small which requires less memory to store.
- Some sets are hashed into same bucket this is called collision problem. More database scans are required to count the support of collided item.
- DHP is particularly powerful for finding the frequent itemsets in early stage.

- It finds L1-itemsets and makes a hash table for C2, and then determines L2 based on the hash table generated in previous pass. However, there is no guarantee that collisions can be avoided. If we use small number of buckets to determine the frequent itemsets, a heavy collision will be occurred. At this time, only few candidate itemsets will be filtered out and the performance might be worse than Apriori algorithm. Enlarging the number of buckets can solve the problem, but the requirement of large memory space will reduce the performance.

Working of DHP algorithm:

Step1: Scan the database to count the support of candidate (C1) item set and select the items. Count $\geq \text{min_sup}$ to add into large item set (L1).

Step 2: Now make possible set of candidate-2 item set in each transaction of database (D2). Hash function is applied on each candidate-2 item set to find the corresponding bucket number.

Step 3: Scan database (D2) and hash each item set of transactions into corresponding hash bucket. Some item sets are hashed into same bucket this is called collision problem.

Step4: Select only that candidate-2 item set whose corresponding bucket count $\geq \text{min_sup}$.





- a. If there is no collision, then add the selected item sets into L2.
- b. Else one more scan of database is required to count the support of collided item sets.

Step 5: Now make possible set of candidate-3 item set (D3) and repeat the same procedure.

Dynamic Itemset Counting

- The Dynamic Itemset Counting (DIC) algorithm is a variation of Apriori, which tries to reduce the number of passes made over a transactional database while keeping the number of itemsets counted in a pass relatively low. Thus, it is basically used for increasing the efficiency of Apriori Algorithm.
- Itemsets are dynamically added and deleted as transactions are read.
- Relies on the fact that for an itemset to be frequent, all of its subsets must also be frequent, so we only examine those itemsets whose subsets are all frequent.
- Algorithm stops after every M transactions to add more itemsets.
- Train analogy: There are stations every M transactions. The passengers are itemsets. Itemsets can get on at any stop as long as they get off at the same stop in the next pass around the database. Only itemsets on the train are counted when they occur in transactions. At the very beginning we can start counting 1-itemsets, at the first station we can start counting some of the 2-itemsets. At the second station we can start counting 3-itemsets as well as any more 2-itemsets that can be counted and so on.

Itemsets are marked in four different ways as they are counted:

- **Solid box:**  confirmed frequent itemset - an itemset we have finished counting and exceeds the support threshold *minsupp*
- **Solid circle:**  confirmed infrequent itemset - we have finished counting and it is below *minsupp*
- **Dashed box:**  suspected frequent itemset - an itemset we are still counting that exceeds *minsupp*
- **Dashed circle:**  suspected infrequent itemset - an itemset we are still counting that is below *minsupp*

DIC Algorithm

1. Mark the empty itemset with a solid square. Mark all the 1-itemsets with dashed circles. Leave all other itemsets unmarked.
2. While any dashed itemsets remain:
 1. Read M transactions. For each transaction, increment the respective counters for the itemsets that appear in the transaction and are marked with dashes.
 2. If a dashed circle's count exceeds *minsupp* (threshold), turn it into a dashed square. If any immediate superset of it has all of its subsets as solid or dashed squares, add a new counter for it and make it a dashed circle.
 3. Once a dashed itemset has been counted through all the transactions, make it solid and stop counting it.

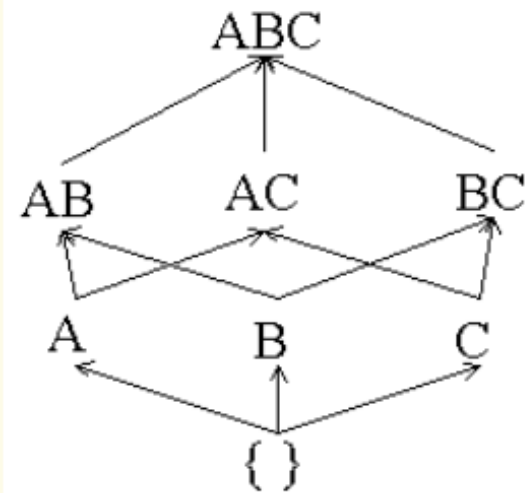
Itemset lattices: An itemset lattice contains all of the possible itemsets for a transaction database. Each itemset in the lattice points to all of its supersets.

- Example: minsupp = 25% and $M = 2$.

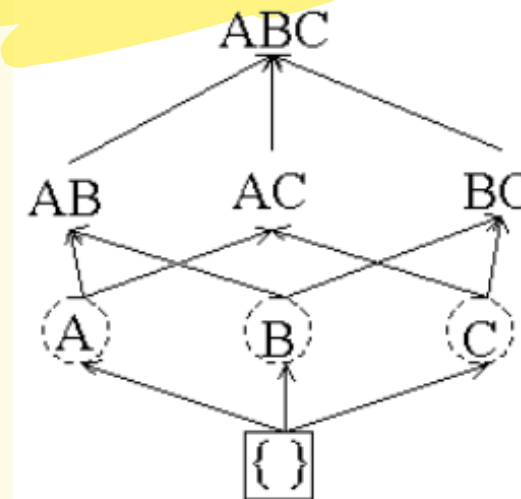
<i>TID</i>	<i>A</i>	<i>B</i>	<i>C</i>
T1	1	1	0
T2	1	0	0
T3	0	1	1
T4	0	0	0

Transaction Database

Itemset lattice for the above transaction database:



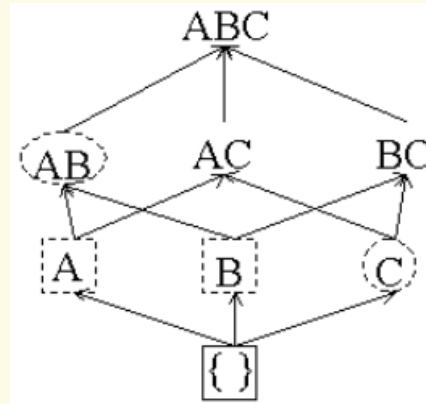
Itemset lattice before any transactions are read:



Counters: $A = 0$, $B = 0$, $C = 0$

Empty itemset is marked with a solid box. All 1-itemsets are marked with dashed circles.

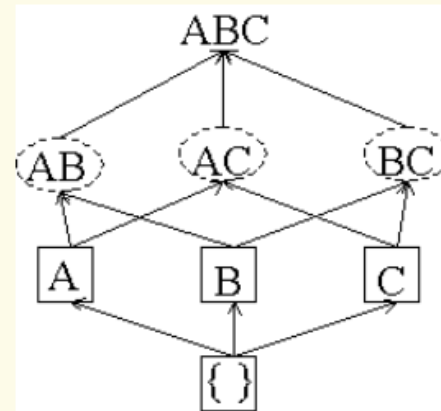
After M transactions are read:



Counters: $A = 2$, $B = 1$, $C = 0$, $AB = 0$

We change A and B to dashed boxes because their counters are greater than minsup (1) and add a counter for AB because both of its subsets are boxes.

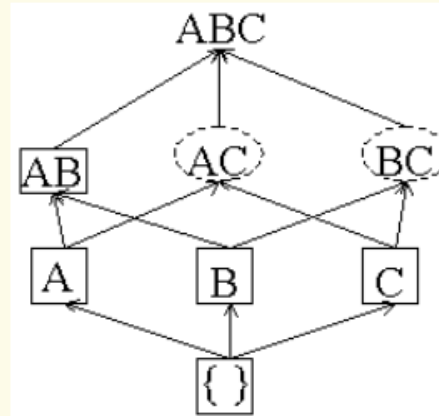
After 2M transactions are read:



Counters: $A = 2$, $B = 2$, $C = 1$, $AB = 0$, $AC = 0$, $BC = 0$

C changes to a square because its counter is greater than minsup. A, B and C have been counted all the way through so we stop counting them and make their boxes solid. Add counters for AC and BC because their subsets are all boxes.

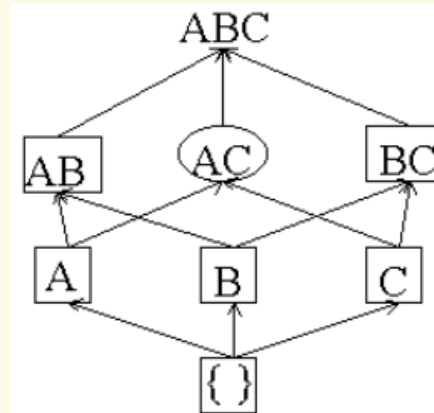
After 3M transactions read:



Counters: $A = 2$, $B = 2$, $C = 1$, $AB = 1$, $AC = 0$, $BC = 0$

AB has been counted all the way through and its counter satisfies minsup so we change it to a solid box. BC changes to a dashed box.

After 4M transactions read:



Counters: $A = 2$, $B = 2$, $C = 1$, $AB = 1$, $AC = 0$, $BC = 1$

AC and BC are counted all the way through. We do not count ABC because one of its subsets is a circle. There are no dashed itemsets left so the algorithm is done.

Mining Frequent Pattern without Candidate Generation (FP Growth)

- Apriori is an algorithm for frequent pattern mining that focuses on generating itemsets and discovering the most frequent itemset. It greatly reduces the size of the itemset in the database, however, Apriori has few shortcomings as well.
- Apriori needs a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.
- Apriori needs multiple scans of the database to check the support of each itemset generated and this leads to high costs.
- These shortcomings can be overcome using the FP growth algorithm.

Frequent Pattern Growth

- A frequent pattern is generated without the need for Candidate Generation. FP growth algorithm represents the database in the form of a tree called a Frequent Pattern Tree (FP Tree).
- Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.
- The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.
- This tree structure maintains the association between the itemsets. The database is fragmented using one frequent item. This fragmented part is called “pattern fragment”. The itemsets of these fragmented patterns are analyzed. Thus, the search for frequent itemsets is reduced comparatively.

Frequent Pattern Algorithm Steps

- 1) Scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency.
- 2) To construct the FP tree, create the root of the tree. The root is represented by null.
- 3) The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. i.e. the branch of tree is constructed with transaction itemsets in descending order of count.
- 4) The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch (for example in the 1st transaction), then this transaction branch would share a common prefix to the root.
This means that the common itemset is linked to the new node of another itemset in this transaction.
- 5) Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions.
- 6) The next step is to mine the created FP Tree. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree. This path(s) are called a conditional pattern base. Conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).
- 7) Construct a Conditional FP Tree, which is formed by a count of itemsets meeting the threshold support.
- 8) Frequent Patterns are generated from the Conditional FP Tree.

*Thank
you*

