**Unit-3**

**Apriori Algorithm**

* Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects.
* Apriori algorithm is also called frequent pattern mining.
* This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.
* It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

**Apriori says:**

The probability that item I is not frequent is if:

* P(I) < minimum support threshold, then I is not frequent.
* P (I+A) < minimum support threshold, then I+A is not frequent, where A also belongs to itemset.
* If an itemset set has value less than minimum support then all of its supersets will also fall below min support, and thus can be ignored. This property is called the Antimonotone property.

**What is Apriori Algorithm?**

* Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.
* Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

**Applications Of Apriori Algorithm**

Some fields where Apriori is used:

* **In Education Field:** Extracting association rules in data mining of admitted students through characteristics and specialties.
* **In the Medical field:** For example Analysis of the patient’s database.
* **In Forestry:** Analysis of probability and intensity of forest fire with the forest fire data.

Apriori is used by many companies like Amazon in the Recommender System and by Google for the auto-complete feature.

* **The steps followed in the Apriori Algorithm of data mining are:**
  1. **Join Step**: This step generates (K+1) itemset from K-itemsets by joining each item with itself.
  2. **Prune Step**: This step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.

**The Apriori Algorithm:**

**Step 1:**Start with Single Items: Identify frequent individual items that meet the minimum support.

**Step 2:**Generate Larger Itemsets: Use the frequent itemsets to create new combinations of larger itemsets.

**Step 3:**Check in Database: Go through the database and count how often these larger itemsets appear.

**Step 4:**Remove Infrequent Itemsets\*: Remove any itemsets that don’t meet the minimum support or whose smaller subsets aren’t frequent.

**Step 5**:Repeat: Keep making larger itemsets and checking them until no more frequent itemsets can be found.

**Step 6**:End: When no more itemsets meet the criteria, stop the algorithm.

**Support**

In data mining, support refers to the relative frequency of an item set in a dataset. For example, if an itemset occurs in 5% of the transactions in a dataset, it has a support of 5%. Support is often used as a threshold for identifying frequent item sets in a dataset, which can be used to generate association rules. For example, if we set the support threshold to 5%, then any itemset that occurs in more than 5% of the transactions in the dataset will be considered a frequent itemset.

The support of an itemset is the number of transactions in which the itemset appears, divided by the total number of transactions. For example, suppose we have a dataset of 1000 transactions, and the itemset {milk, bread} appears in 100 of those transactions. The support of the itemset {milk, bread} would be calculated as follows:

Support({milk, bread}) = Number of transactions containing

{milk, bread} / Total number of transactions

= 100 / 1000

= 10%

So the support of the itemset {milk, bread} is 10%. This means that in 10% of the transactions, the items milk and bread were both purchased.

In general, the support of an itemset can be calculated using the following formula:

***Support(X) = (Number of transactions containing X) / (Total number of transactions)***

**Confidence**

In data mining, confidence is a measure of the reliability or support for a given association rule. It is defined as the proportion of cases in which the association rule holds true, or in other words, the percentage of times that the items in the antecedent (the “if” part of the rule) appear in the same transaction as the items in the consequent (the “then” part of the rule).

Confidence is a measure of the likelihood that an itemset will appear if another itemset appears. For example, suppose we have a dataset of 1000 transactions, and the itemset {milk, bread} appears in 100 of those transactions. The itemset {milk} appears in 200 of those transactions. The confidence of the rule “If a customer buys milk, they will also buy bread” would be calculated as follows:

**Confidence("If a customer buys milk, they will also buy bread")**

**= Number of transactions containing**

**{milk, bread} / Number of transactions containing {milk}**

**= 100 / 200**

**= 50%**

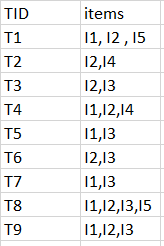
So the confidence of the rule “If a customer buys milk, they will also buy bread” is 50%. This means that in 50% of the transactions where milk was purchased, bread was also purchased.

In general, the confidence of a rule can be calculated using the following formula**:**

***Confidence(X => Y) = (Number of transactions containing X and Y) / (Number of transactions containing X)***

**PROBLEM:**

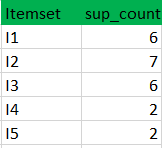
**Example:**



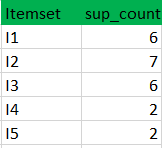
Minimum support count is 2

Minimum confidence is 60%

**Step-1:**K=1  
(I) Create a table containing support count of each item present in dataset – Called **C1(candidate set)**

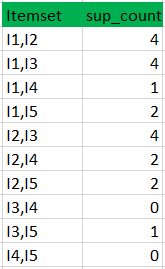


**II**) compare candidate set item’s support count with minimum support count(here min\_support=2 if support\_count of candidate set items is less than min\_support then remove those items). This gives us itemset L1.

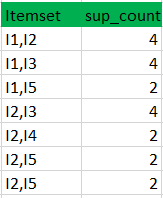
****

**Step-2:** K=2

* Generate candidate set C2 using L1 (this is called join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common.
* Check all subsets of an itemset are frequent or not and if not frequent remove that itemset.(Example subset of{I1, I2} are {I1}, {I2} they are frequent.Check for each itemset)
* Now find support count of these itemsets by searching in dataset.

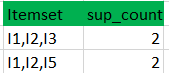
****

**(II)**Compare candidate (C2) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L2.

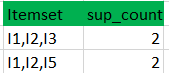
****

**Step-3:**

* Generate candidate set C3 using L2 (join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common. So here, for L2, first element should match.  
  So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}
* Check if all subsets of these itemsets are frequent or not and if not, then remove that itemset.(Here subset of {I1, I2, I3} are {I1, I2},{I2, I3},{I1, I3} which are frequent. For {I2, I3, I4}, subset {I3, I4} is not frequent so remove it. Similarly check for every itemset)
* find support count of these remaining itemset by searching in dataset.

****

**(II)** Compare candidate (C3) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L3.

****

**Step-4:**

Generate candidate set C4 using L3 (join step). Condition of joining Lk-1 and Lk-1 (K=4) is that, they should have (K-2) elements in common. So here, for L3, first 2 elements (items) should match.

Check all subsets of these itemsets are frequent or not (Here itemset formed by joining L3 is {I1, I2, I3, I5} so its subset contains {I1, I3, I5}, which is not frequent). So no itemset in C4

We stop here because no frequent itemsets are found further

**Confidence**   
A confidence of 60% means that 60% of the customers, who purchased milk and bread also bought butter.

* Confidence(A->B)=Support\_count(A∪B)/Support\_count(A)
* So here, by taking an example of any frequent itemset, we will show the rule generation.  
  Itemset {I1, I2, I3} //from L3  
  SO rules can be  
  [I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4\*100=50%  
  [I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/4\*100=50%  
  [I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4\*100=50%  
  [I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/6\*100=33%  
  [I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7\*100=28%  
  [I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/6\*100=33%
* So if minimum confidence is 50%, then first 3 rules can be considered as strong association rules.

**Advantages**

* Easy to understand algorithm
* Join and Prune steps are easy to implement on large itemsets in large databases

**Disadvantages**

* It requires high computation if the itemsets are very large and the minimum support is kept very low.
* The entire database needs to be scanned.

**Frequent Pattern Growth Algorithm**

* This algorithm is an improvement to the Apriori method. 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 or FP tree.
* This tree structure will maintain 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 with this method, the search for frequent itemsets is reduced comparatively.

**What is Frequent Pattern Growth Algorithm?**

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

**Frequent-Pattern Tree**

The frequent-pattern tree (FP-tree) is a compact data structure that stores quantitative information about frequent patterns in a database. Each transaction is read and then mapped onto a path in the FP-tree. This is done until all transactions have been read. Different transactions with common subsets allow the tree to remain compact because their paths overlap.

A frequent Pattern Tree is made with the initial item sets 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 item set.

The root node represents null, while the lower nodes represent the item sets. The associations of the nodes with the lower nodes, that is, the item sets with the other item sets, are maintained while forming the tree.

**Frequent Pattern Growth Algorithm**

**Step 1**: Create an FP-Tree: Organize the data into a tree based on item frequency.

**Step 2:** Check for Single Path: If the tree has just one path, combine items along that path to form frequent patterns.

**Step 3:**Process Multiple Paths:

- If the tree has multiple branches, for each item:

- Combine the item with the current pattern.

- Create a smaller tree (conditional FP-tree) for that item.

- Repeat the process on this smaller tree.

**Step 4.** Gather Patterns: Collect all the frequent patterns found in the process.

**Step 5.** Stop: When no more patterns can be found, stop.

**Problem**

**Example**

Support threshold=50%, Confidence= 60%

**Table 1:**

|  |  |
| --- | --- |
| **Transaction** | **List of items** |
| **T1** | **I1,I2,I3** |
| **T2** | **I2,I3,I4** |
| **T3** | **I4,I5** |
| **T4** | **I1,I2,I4** |
| **T5** | **I1,I2,I3,I5** |
| **T6** | **I1,I2,I3,I4** |

threshold=50% => 0.5\*6= 3 => min\_sup=3

**Table 2: Count of each item**

|  |  |
| --- | --- |
| **Item** | **Count** |
| **I1** | **4** |
| **I2** | **5** |
| **I3** | **4** |
| **I4** | **4** |
| **I5** | **2** |

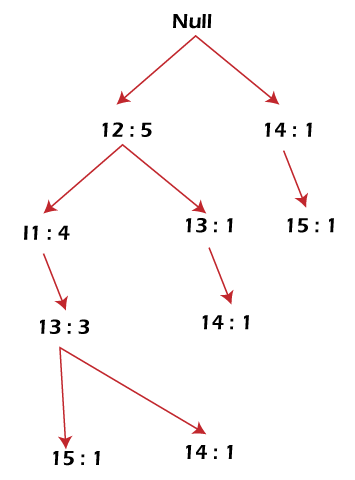
**Table 3: Sort the itemset in descending order.**

|  |  |
| --- | --- |
| **Item** | **Count** |
| **I2** | **5** |
| **I1** | **4** |
| **I3** | **4** |
| **I4** | **4** |

**Build FP Tree**

Considering the root node null.

1. The first scan of Transaction T1: I1, I2, I3 contains three items {I1:1}, {I2:1}, {I3:1}, where I2 is linked as a child, I1 is linked to I2 and I3 is linked to I1.
2. T2: I2, I3, and I4 contain I2, I3, and I4, where I2 is linked to root, I3 is linked to I2 and I4 is linked to I3. But this branch would share the I2 node as common as it is already used in T1.
3. Increment the count of I2 by 1, and I3 is linked as a child to I2, and I4 is linked as a child to I3. The count is {I2:2}, {I3:1}, {I4:1}.
4. T3: I4, I5. Similarly, a new branch with I5 is linked to I4 as a child is created.
5. T4: I1, I2, I4. The sequence will be I2, I1, and I4. I2 is already linked to the root node. Hence it will be incremented by 1. Similarly I1 will be incremented by 1 as it is already linked with I2 in T1, thus {I2:3}, {I1:2}, {I4:1}.
6. T5:I1, I2, I3, I5. The sequence will be I2, I1, I3, and I5. Thus {I2:4}, {I1:3}, {I3:2}, {I5:1}.
7. T6: I1, I2, I3, I4. The sequence will be I2, I1, I3, and I4. Thus {I2:5}, {I1:4}, {I3:3}, {I4 1}.



**Mining of FP-tree is summarized below:**

1. The lowest node item, I5, is not considered as it does not have a min support count. Hence it is deleted.
2. The next lower node is I4. I4 occurs in 2 branches , {I2,I1,I3:,I41},{I2,I3,I4:1}. Therefore considering I4 as suffix the prefix paths will be {I2, I1, I3:1}, {I2, I3: 1} this forms the conditional pattern base.
3. The conditional pattern base is considered a transaction database, and an FP tree is constructed. This will contain {I2:2, I3:2}, I1 is not considered as it does not meet the min support count.
4. This path will generate all combinations of frequent patterns : {I2,I4:2},{I3,I4:2},{I2,I3,I4:2}
5. For I3, the prefix path would be: {I2,I1:3},{I2:1}, this will generate a 2 node FP-tree : {I2:4, I1:3} and frequent patterns are generated: {I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3}.
6. For I1, the prefix path would be: {I2:4} this will generate a single node FP-tree: {I2:4} and frequent patterns are generated: {I2, I1:4}.

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Conditional Pattern Base | Conditional FP-tree | Frequent Patterns Generated |
| I4 | {I2,I1,I3:1},{I2,I3:1} | {I2:2, I3:2} | {I2,I4:2},{I3,I4:2},{I2,I3,I4:2} |
| I3 | {I2,I1:3},{I2:1} | {I2:4, I1:3} | {I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3} |
| I1 | {I2:4} | {I2:4} | {I2,I1:4} |

**Advantages of Frequent Patterns Growth Algorithm:**

* This algorithm needs to scan the database twice when compared to Apriori, which scans the transactions for each iteration.
* The pairing of items is not done in this algorithm, making it faster.
* The database is stored in a compact version in memory.
* It is efficient and scalable for mining both long and short frequent patterns.

**Disadvantages of Frequent Patterns Growth Algorithm**

* FP Tree is more cumbersome and difficult to build than Apriori.
* It may be expensive.
* The algorithm may not fit in the shared memory when the database is large.

### **Difference between Apriori and FP Growth Algorithm**

|  |  |
| --- | --- |
| **Apriori** | **Frequent Pattern Growth** |
| Apriori generates frequent patterns by making the itemsets using pairings such as single item set, double itemset, and triple itemset. | FP Growth generates an FP-Tree for making frequent patterns. |
| Apriori uses candidate generation where frequent subsets are extended one item at a time. | Apriori uses candidate generation where frequent subsets are extended one item at a time. |
| Since apriori scans the database in each step, it becomes time-consuming for data where the number of items is larger. | FP-tree requires only one database scan in its beginning steps, so it consumes less time |
| A converted version of the database is saved in the memory | A set of conditional FP-tree for every item is saved in the memory |
| It uses a breadth-first search | It uses a depth-first search. |

**MULTIDIMENSIONAL ASSOCIATION RULES**

* Multidimensional association rule comprises of more than one aspect
* Numeric attributes should be discretized.
* Attributes can be unmitigated or quantitative.
* Quantitative characteristics are numeric and consolidate pecking order.

**Approaches in mining multi dimensional affiliation rules :**  
Three approaches in mining multi dimensional affiliation rules are as following.

1. **Using static discretization of quantitative qualities :**
   * Discretization is static and happens preceding mining.
   * Discretized ascribes are treated as unmitigated.
   * Use apriori calculation to locate all k-regular predicate sets(this requires k or k+1 table outputs). Each subset of regular predicate set should be continuous.

**Example –**  
If in an information block the 3D cuboid (age, pay, purchases) is continuous suggests (age, pay), (age, purchases), (pay, purchases) are likewise regular.

**Note –**  
Information blocks are appropriate for mining since they make mining quicker. The cells of an n-dimensional information cuboid relate to the predicate cells.

1. **Using powerful discretization of quantitative traits :**
   * Known as mining Quantitative Association Rules.
   * Numeric properties are progressively discretized.

**Example –**:

age(X, "20..25") Λ income(X, "30K..41K")buys ( X, "Laptop Computer")

1. **Grid FOR TUPLES :**  
   **Using distance based discretization with bunching –**  
   This id dynamic discretization measure that considers the distance between information focuses. It includes a two stage mining measure as following.
   * Perform bunching to discover the time period included.
   * Get affiliation rules via looking for gatherings of groups that happen together.