MCA:304A Data Warehousing and Data Mining

Unit III

Association Analysis: Basic Concepts and Algorithms

Prepared by S NOORTAJ
Asst. professor
KMMIPS, TIRUPATHI

Association Analysis

Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently a item set occurs in a transaction. A typical example is a 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.
In association rule mining there are 3 types of algorithms.
1. Apriori algorithm
2. Fp growth algorithm
3. Eclact algorithm.
But here mainly discuss about apriori algorithm and fp growth algorithm.

Frequent Item set

Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset.

Frequent Item set Generation

Frequent Mining shows which items appear together in a transaction or relation.Frequent mining is generation of association rules from a Transactional Dataset.

If there are 2 items X and Y purchased frequently then its good to put them together in stores or provide some discount offer on one item on purchase of other item. This can really increase the sales.
For example it is likely to find that if a customer buys **Milk** and **bread** he/she also buys **Butter**. So the association rule is **['milk]^['bread']=>['butter']**. So seller can suggest the customer to buy butter if he/she buys Milk and Bread.

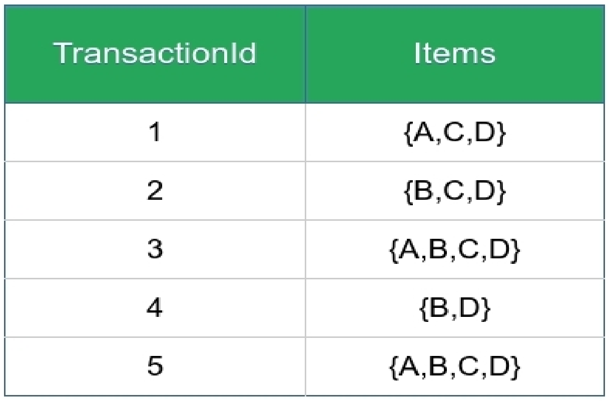
Important Definitions :

- *Support:* It is one of the measure of interestingness. This tells about usefulness and certainty of rules. **5% Support** means total 5% of transactions in database follow the rule.
  Support(A -> B) = Support_count(A ∪ B)
- *Confidence:* A confidence of 60% means that 60% of the customers who purchased a milk and bread also bought butter.
  Confidence(A -> B) = Support_count(A ∪ B) / Support_count(A)

If a rule satisfies both minimum support and minimum confidence, it is a strong rule.

- *Support count(X):* Number of transactions in which X appears. If X is A **union** B then it is the number of transactions in which A and B both are present.
- *Maximal Itemset:* An itemset is maximal frequent if none of its supersets are frequent.
- *Closed Itemset:* An itemset is closed if none of its immediate supersets have same support count same as Itemset.
- *K- Itemset:* Itemset which contains K items is a K-itemset. So it can be said that an itemset is frequent if the corresponding support count is greater than minimum support count.

Example On finding Frequent Itemsets – Consider the given dataset with given transactions.



TransactionId	Items
1	{A,C,D}
2	{B,C,D}
3	{A,B,C,D}
4	{B,D}
5	{A,B,C,D}

- Lets say minimum support count is 3
- Relation hold is maximal frequent => closed => frequent

  1-frequent: *{A} = 3; // not closed due to {A, C} and not maximal {B} = 4; // not closed due to {B, D} and no maximal {C} = 4; // not closed due to {C, D} not maximal {D} = 5; // closed item-set since not immediate super-set has same count. Not maximal*

  2-frequent: *{A, B} = 2 // not frequent because support count < minimum support count so ignore {A, C} = 3 // not closed due to {A, C, D} {A, D} = 3 // not closed due to {A, C, D} {B, C} = 3 // not closed due to {B, C, D} {B, D} = 4 // closed but not maximal due to {B, C, D} {C, D} = 4 // closed but not maximal due to {B, C, D}*

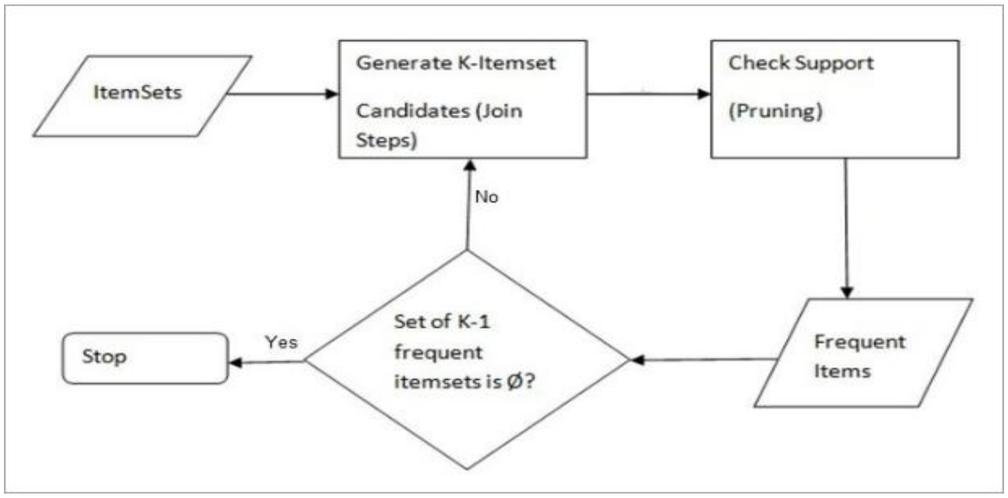
  3-frequent: *{A, B, C} = 2 // ignore not frequent because support count < minimum support count {A, B, D} = 2 // ignore not frequent because support count < minimum support count {A, C, D} = 3 // maximal frequent {B, C, D} = 3 // maximal frequent*

  4-frequent: *{A, B, C, D} = 2 //ignore not frequent.*

Apriori Algorithm:

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a **breadth-first search** and **Hash Tree** to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

FLOW CHART


ItemSets
Generate K-Itemset Candidates (Join Steps)
Check Support (Pruning)
Frequent Items
Set of K-1 frequent itemsets is Ø?
No
Yes
Stop

Apriori algorithm for frequent itemset generation

```
1.  $k = 1$
2.  $F_k = \{i \mid i \in I \wedge \sigma(\{i\}) \geq N \times \min\sup\}$
{Find all frequent 1-itemsets}
3.  repeat
4.      $k = k + 1$
5.      $C_k = \textit{apriori-gen}(F_{k-1})$ {Generate
candidate itemsets}
6.      for each transaction $t \in T$ do
7.          $C_t = \textit{subset}(C_k, t)$
{Identify all candidates that belong to t}
8.          for each candidate itemset
              $c \in C_t$ do
9.          $\sigma(c) = \sigma(c) + 1$
{Increment support count}
10.         end for
11.     end for
```

Steps for Apriori Algorithm:

Below are the steps for the apriori algorithm:

Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.

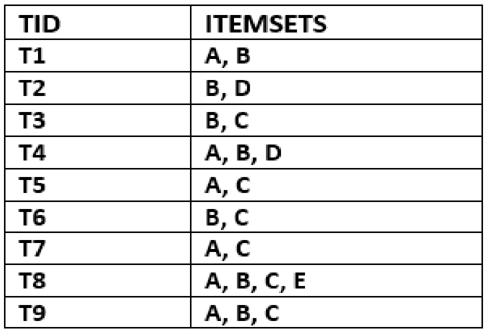
Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

Step-4: Sort the rules as the decreasing order of lift.

Apriori Algorithm Working

We will understand the apriori algorithm using an example and mathematical calculation:

Example: Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:



TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Given: Minimum Support= 2, Minimum Confidence= 50%

Solution:

Step-1: Calculating C1 and L1:

- In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the **Candidate set or C1.**

Item set	Support-count
A	6
B	7
C	5
D	2
E	1

Now, we will take out all the itemsets that have the greater support count that the Minimum Support (2). It will give us the table for the **frequent itemset L1.**

Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.

Step-2: Candidate Generation C2, and L2:

- In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.
- After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2:

Item set	**Support-count**
{A,B}	4
{A,C}	4
{A,D}	1
{B,C}	4
{B,D}	2
{C,D}	0

- Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2.

Item set	**Support-count**
{A,B}	4
{A,C}	4
{B,C}	4
{B,D}	2

Step-3: Candidate generation C3, and L3:

- For C3, we will repeat the same two processes, but now we will form the C3 table with subsets of three itemsets together, and will calculate the support count from the dataset. It will give the below table:

Item set	Support-count
{A,B,C}	2
{B,C,D}	1
{A,C,D}	0
{A,B,D}	0

Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}.**

Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula **sup(A ^B)/A.** After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold(50%).

Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

As the given threshold or minimum confidence is 50%, so the first three rules **A ^B → C, B^C → A, and A^C → B** can be considered as the strong association rules for the given problem.

EX:2: Consider the following dataset and we will find frequent itemsets and generate association rules for them.

TID	items
T1	I1, I2 , I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

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)**

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

(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.

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

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.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

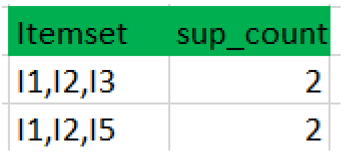
(II) compare candidate (C2) support count with minimum support count(here min_support=2 if support_count of candidate set item is less than min_support then remove those items) this gives us itemset L2.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I2,I5	2

Step-3:

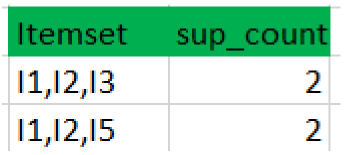
- Generate candidate set C3 using L2 (join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common. So here, for L2, first element should match.
  So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}

- ❖ 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.



Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

(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.



Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

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

Thus, we have discovered all the frequent item-sets. Now generation of strong association rule comes into picture. For that we need to calculate confidence of each rule.

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 of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

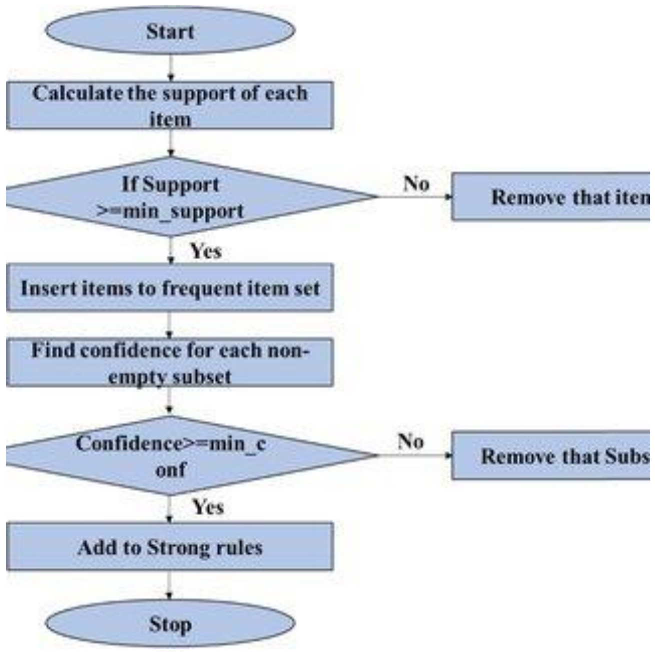
Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.
- The time complexity and space complexity of the apriori algorithm is $O(2^D)$, which is very high. Here D represents the horizontal width present in the database.

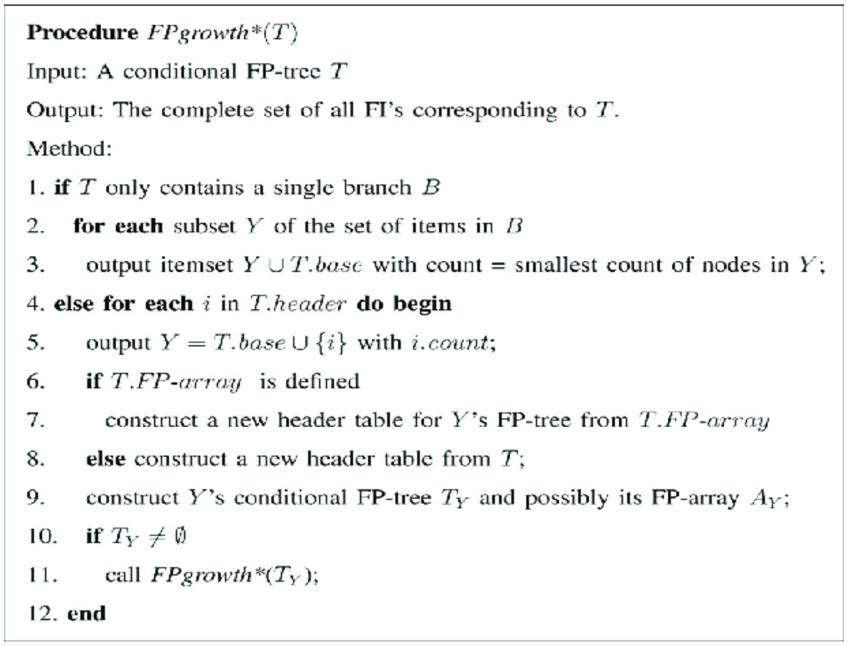
What is FP Growth Algorithm?

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.

Flow chart:



Algorithm:



```
Procedure FPgrowth*(T)
Input: A conditional FP-tree T
Output: The complete set of all FI's corresponding to T.
Method:
1. if T only contains a single branch B
2.   for each subset Y of the set of items in B
3.     output itemset $Y \cup T.base$ with count = smallest count of nodes in Y;
4. else for each i in T.header do begin
5.     output $Y = T.base \cup \{i\}$ with i.count;
6.     if T.FP-array is defined
7.       construct a new header table for Y's FP-tree from T.FP-array
8.     else construct a new header table from T;
9.     construct Y's conditional FP-tree $T_Y$ and possibly its FP-array $A_Y$;
10.    if $T_Y \neq \emptyset$
11.      call FPgrowth*($T_Y$);
12. end
```

FP-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.

the FP-tree as the tree structure given below:

1. One root is labelled as "null" with a set of item-prefix subtrees as children and a frequent-item-header table.
2. Each node in the item-prefix subtree consists of three fields:
    - Item-name: registers which item is represented by the node;
    - Count: the number of transactions represented by the portion of the path reaching the node;
    - Node-link: links to the next node in the FP-tree carrying the same item name or null if there is none.
3. Each entry in the frequent-item-header table consists of two fields:
    - Item-name: as the same to the node;
    - Head of node-link: a pointer to the first node in the FP-tree carrying the item name.

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

Solution: Support 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.

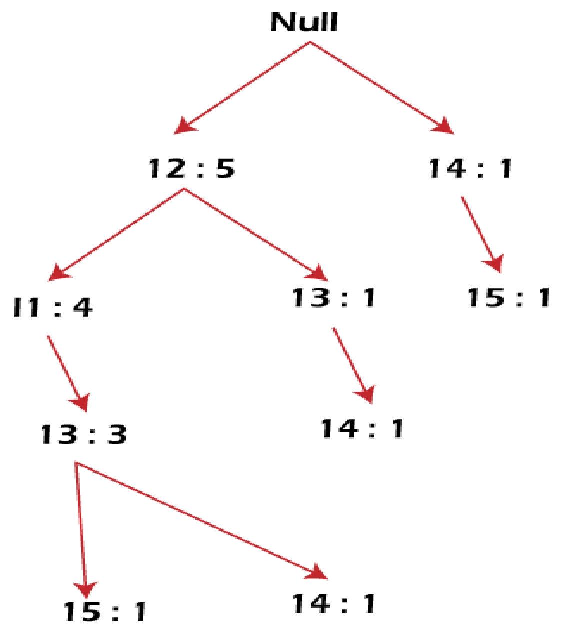
Item	Count
I2	5

I1	4
I3	4
I4	4

Build FP Tree

Let's build the FP tree in the following steps, such as:

1. Considering the root node null.
2. 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.
3. 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.
4. 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}.
5. T3: I4, I5. Similarly, a new branch with I5 is linked to I4 as a child is created.
6. 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}.
7. T5:I1, I2, I3, I5. The sequence will be I2, I1, I3, and I5. Thus {I2:4}, {I1:3}, {I3:2}, {I5:1}.
8. T6: I1, I2, I3, I4. The sequence will be I2, I1, I3, and I4. Thus {I2:5}, {I1:4}, {I3:3}, {I4 1}.

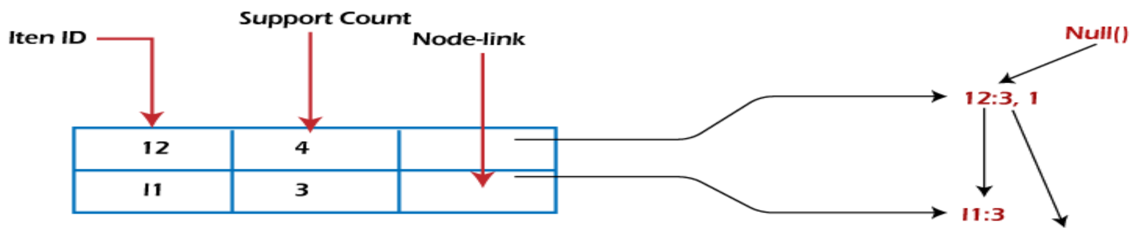


Mining of FP-tree is summarized below:

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}

The diagram given below depicts the conditional FP tree associated with the conditional node I3.



Advantages of FP Growth Algorithm

Here are the following advantages of the FP growth algorithm, such as:

- 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 FP-Growth Algorithm

This algorithm also has some disadvantages, such as:

- 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	**FP 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.	FP-growth generates a conditional FP-Tree for every item in the data.
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.

What is The ECLAT Algorithm?

ECLAT is an acronym for Equivalence Class Clustering and bottom-up Lattice Traversal. ECLAT algorithm is a frequent pattern mining algorithm just like the apriori algorithm. We can say that the ECLAT algorithm is an efficient and scalable version of the apriori algorithm with the following improvements.

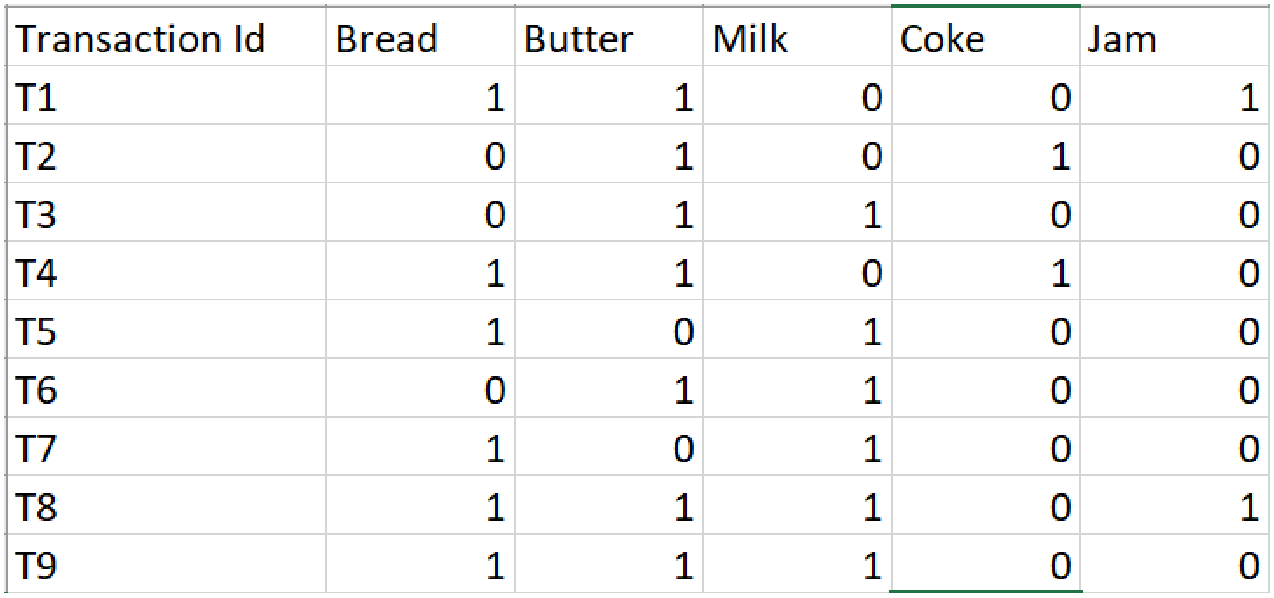
- The apriori algorithm and the fp-growth algorithm work with the horizontal transaction dataset. On the contrary, the ECLAT algorithm works on a vertical data format.
- ECLAT algorithm uses a depth-first search approach to traverse the itemsets. The Apriori algorithm uses a breadth-first approach to traverse the transaction dataset.

How the algorithm work?

The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoiding the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined.

Let us now understand the above stated working with an example:-

Consider the following transactions record:-



Transaction Id	Bread	Butter	Milk	Coke	Jam
T1	1	1	0	0	1
T2	0	1	0	1	0
T3	0	1	1	0	0
T4	1	1	0	1	0
T5	1	0	1	0	0
T6	0	1	1	0	0
T7	1	0	1	0	0
T8	1	1	1	0	1
T9	1	1	1	0	0

The above-given data is a boolean matrix where for each cell (i, j), the value denotes whether the j'th item is included in the i'th transaction or not. 1 means true while 0 means false.

We now call the function for the first time and arrange each item with it's tidset in a tabular fashion:-

k = 1, minimum support = 2

Item	**Tidset**
Bread	{T1, T4, T5, T7, T8, T9}
Butter	{T1, T2, T3, T4, T6, T8, T9}
Milk	{T3, T5, T6, T7, T8, T9}
Coke	{T2, T4}
Jam	{T1, T8}

We now recursively call the function till no more item-tidset pairs can be combined:-

k = 2

Item	**Tidset**
{Bread, Butter}	{T1, T4, T8, T9}
{Bread, Milk}	{T5, T7, T8, T9}
{Bread, Coke}	{T4}
{Bread, Jam}	{T1, T8}
{Butter, Milk}	{T3, T6, T8, T9}
{Butter, Coke}	{T2, T4}
{Butter, Jam}	{T1, T8}
{Milk, Jam}	{T8}

k = 3

Item	**Tidset**
{Bread, Butter, Milk}	{T8, T9}
{Bread, Butter, Jam}	{T1, T8}

k = 4

Item	**Tidset**
{Bread, Butter, Milk, Jam}	{T8}

We stop at k = 4 because there are no more item-tidset pairs to combine.

Since minimum support = 2, we conclude the following rules from the given dataset:-

Items Bought	**Recommended Products**
Bread	Butter
Bread	Milk
Bread	Jam
Butter	Milk
Butter	Coke
Butter	Jam
Bread and Butter	Milk
Bread and Butter	Jam

Example 2:

Step-By-Step ECLAT Algorithm Explanation

To perform association rule mining using the ECLAT algorithm, we first define the minimum support, confidence, and lift. After this, we will convert the transaction dataset to vertical format if it isn’t already so. Next, we perform candidate generation, pruning, database scan, and rule generation to create association rules. These steps are almost similar to the apriori algorithm.

Step 1: Convert Transaction Data to Vertical Format

Normally, the transactions in a dataset are stored in horizontal format. It means that each row in the dataset contains a transaction ID and the corresponding items in the transaction as shown below.

Transaction ID	Items
T1	I1, I3, I4
T2	I2, I3, I5, I6
T3	I1, I2, I3, I5
T4	I2, I5
T5	I1, I3, I5

The dataset in Horizontal Format

In vertical format, the rows in the transaction data contain an item and the corresponding transactions in which the item is present. The dataset in the vertical format looks as follows.

Items	Transaction IDs
I1	T1,T3,T5
I2	T2,T3,T4
I3	T1,T2,T3,T5
I4	T1
I5	T2,T3,T4,T5
I6	T2

The dataset in Vertical Format

Step 2: Candidate Generation From the Dataset

After transforming the dataset into the vertical format, we use the candidate generation step to generate itemsets that can possibly be frequent itemsets. For this, we start by creating sets containing single items. If there are N items in the dataset, we create N candidate sets.

After creating the candidate sets, we use the minimum support count to select frequent itemsets containing one item. Once we get the frequent itemsets with one item, we iteratively join them to create larger sets containing 2, 3, 4, 5, or more items.

In the candidate generation process, we generate the candidate itemsets containing k items by joining the frequent itemsets with k-1 items in common. This process is repeated until no new frequent itemsets can be generated.

Step 3: Pruning the Candidate Itemsets

The pruning step in the ECLAT algorithm is derived from the apriori algorithm. It is based on the concept that a subset of a frequent itemset must also be a frequent itemset. In other words, if we have an itemset having a subset that is not a frequent itemset, the itemset cannot be a frequent itemset.

We use pruning to remove the candidate sets before even scanning the dataset to calculate the support count and minimize the time taken in executing the algorithm. After creating itemsets of K items, we use the following steps to prune the candidate set.

For each candidate set having k items, we check if each of its subsets having k-1 is a frequent itemset or not. If yes, the candidate set is considered for generating frequent itemsets. Even if we find a single subset of the candidate set that is not a frequent itemset, we reject or prune the itemsets.

Step 4: Frequent Itemset Generation

After Pruning, we check the support count of the remaining candidate itemsets. For this, we scan the transaction dataset to find the support of each frequent itemset.

After calculating the support count of each candidate itemset, we drop the itemsets having a support count less than the minimum support count from the candidate list. The rest of the itemsets are considered frequent itemsets.

After generating the frequent itemsets having k items, we create candidate itemsets having k+1 items, perform pruning, database scan, and then frequent itemset generation to generate frequent itemsets having k+1 items.

We iterate through steps 2 to 4 until we cannot generate more frequent itemsets.

Step 4: Association Rule Generation

After creating frequent itemsets, we generate association rules. If we have a frequent itemset {I}, we can create association rules in the form of {S}-> {I-S}. Here {S} is a subset of the frequent itemset {I}.

ECLAT Algorithm Numerical Example

To explain the ECLAT algorithm using the numerical example, we will use the following dataset.

Transaction ID	Items
T1	I1, I3, I4
T2	I2, I3, I5, I6
T3	I1, I2, I3, I5
T4	I2, I5
T5	I1, I3, I5

The dataset in Horizontal format

The above transaction dataset is in horizontal format. It contains five transactions having transaction IDs T1, T2, T3, T4, and T5. The dataset contains six different items namely I1, I2, I3, I4, I5, and I6.

Convert Transaction Data to Vertical Format

To proceed with the explanation of the ECLAT algorithm using numerical examples, we need to represent the dataset in vertical format. In vertical format, each row of the dataset represents an item and all the transactions in which the item is present. The transformed dataset looks as follows.

Items	Transaction IDs
I1	T1,T3,T5
I2	T2,T3,T4
I3	T1,T2,T3,T5
I4	T1
I5	T2,T3,T4,T5
I6	T2

the dataset in vertical format

The above dataset for the ECLAT algorithm numerical example contains five transactions having transaction IDs T1, T2, T3, T4, and T5. In the transactions, it contains six different items namely I1, I2, I3, I4, I5, and I6.

Let us now use the Eclat algorithm to find association rules from the above dataset. For our numerical example, we will use the minimum support count of 2 and minimum confidence of 75 percent. To help us calculate the support of the itemsets, we will create a matrix representing the presence of items in a transaction as shown below.

	T1	T2	T3	T4	T5
I1	1	0	1	0	1
I2	0	1	1	1	0
I3	1	1	1	0	1
I4	1	0	0	0	0
I5	0	1	1	1	1
I6	0	1	0	0	0

Transaction matrix

The above matrix contains Items on the vertical axis and transaction IDs on the horizontal axis. If an item is present in a transaction, the corresponding cell is set to 1. Otherwise, it is set to 0. We will use this matrix to calculate the support count of itemsets as it is easier to scan this matrix compared to the transaction dataset.

To calculate the support count of any given itemset, we will find the number of columns in which all the items in the given itemset are set to 1 in the above matrix.

Create Frequent Itemsets With 1 Item

The ECLAT algorithm starts by creating candidate itemsets with one item. For this, let us calculate the support count of each item.

Itemset	**Transactions**	**Support Count**
{I1}	T1, T3, T5	3
{I2}	T2, T3, T4	3
{I3}	T1,T2,T3,T5	4
{I4}	T1	1
{I5}	T2,T3,T4,T5	4
{I6}	T2	1

candidate itemset with one item

The above table contains the support count of candidate itemsets with one item. Here, you can observe that the itemsets {I4} and {I6} have support count 1 which is less than the minimum support count 2. Hence, we will omit these itemsets from the candidate table. After this, we will get the table containing frequent itemsets with a single item as shown below.

Itemset	**Transactions**	**Support Count**
{I1}	T1, T3, T5	3
{I2}	T2, T3, T4	3
{I3}	T1,T2,T3,T5	4
{I5}	T2,T3,T4,T5	4

Frequent itemset with one item

In the above table, we have created frequent itemsets containing a single item. Now, we will calculate the frequent itemsets with two items.

Create Frequent Itemsets With 2 Items

To create frequent itemsets with two items, we will first create the candidate itemset with two items. For this, we will join all the frequent itemsets with one item with each other. After joining, we will get the following item sets.

{I1,I2}, {I1,I3}, {I1,I5},{I2,I3},{I2,I5}, and {I3,I5}

After creating the itemsets with two items, we need to prune the itemsets having subsets that are not frequent itemsets. As the {I1}, {I2}, {I3}, and {I5} all are frequent itemsets, no itemsets will be removed from the above list.

As the next step, we will calculate the support count of each itemset having two items to create the candidate itemset. The result is tabulated below.

Itemset	**Transactions**	**Support Count**
{I1,I2}	T3	1
{I1,I3}	T1, T3, T5	3
{I1,I5}	T3, T5	2
{I2,I3}	T2,T3	2
{I2,I5}	T2, T3, T4	3
{I3,I5}	T2, T3, T5	3

Candidate itemset with 2 items

In the above candidate itemset, you can observe that the itemset {I1, I2} has the support count 1 which is less than the minimum support count. Hence, we will remove the above itemset from the table and obtain the table containing frequent itemsets with two items as shown below.

Itemset	**Transactions**	**Support Count**
{I1,I3}	T1, T3, T5	3
{I1,I5}	T3, T5	2
{I2,I3}	T2,T3	2
{I2,I5}	T2, T3, T4	3
{I3,I5}	T2, T3, T5	3

Frequent Itemsets with 2 items

Here, we have obtained frequent itemsets with two items. Let us now calculate the frequent itemsets with three items.

Calculate Frequent Itemsets With Three Items

To calculate the frequent itemsets with three items, we first need to calculate the candidate set. For this, let us first join the frequent itemsets with two items and create the following itemsets with three items.

{I1, I3, I5}, {I1, I2, I3}, {I1, I2, I5}, {I2, I3, I5}

On the above itemsets, we will perform pruning to remove any itemset that has a subset that is not a frequent itemset. For this, we will create subsets of 2 items for each itemset and check if they are frequent itemsets or not. All the subsets of the above itemsets are tabulated below.

Itemset	**Subsets**	**All the subsets are frequent itemsets?**
{I1, I3, I5}	{I1, I3},{I1, I5},{I3, I5}	Yes
{I1, I2, I3}	{I1, I2}, {I1, I3}, {I2, I3}	No
{I1, I2, I5}	{I1, I2},{I1, I5},{I2, I5}	No
{I2, I3, I5}	{I2, I3}, {I2, I5}, {I3, I5}	Yes

Pruning the Itemsets with three items

In the table, you can observe that the itemset {I1,I2 , I3} and {I1, I2, I5} contain the itemset {I1, I2} which is not a frequent itemset. Hence, we will prune the itemsets {I1, I2, I3} and {I1, I2, I5}. After this, we will get the itemsets {I1, I3, I5} and {I2, I3, I5} as candidate itemsets for the itemsets having three items. Let us calculate their support count.

Itemset	Transactions	Support Count
{I2, I3, I5}	T2, T3	2
{I1, I3, I5}	T3,T5	2

Candidate itemsets with 3 items

In the above table, both itemsets have a support count of 2 which is equal to the minimum support count. Hence, both itemsets will be considered frequent itemsets.

Itemset	Transactions	Support Count
{I2, I3, I5}	T2, T3	2
{I1, I3, I5}	T3,T5	2

Frequent Itemsets with 3 items

Calculate Frequent Itemsets With Three Items.

Now, we will calculate the frequent itemsets with four items. For this, we will first join the items in the frequent itemsets with three items to create itemsets with four items. We will get only one item set as shown below.

{I1, I2, I3, I5}

Now, the above itemset has four subsets with three elements i.e. {I2, I3, I5},{I1, I3, I5}, {I1, I2, I5}, {I1, I2, I3}. In these itemsets, {I1, I2, I5} and {I1, I2, I3} are not frequent itemsets. Hence, we will prune the itemset {I1, I2, I3, I5}. Thus, we have no candidate set for itemsets with 4 items. Hence, the process of frequent itemset generation stops here.

Now, let us tabulate all the frequent itemsets created in this numerical example on the ECLAT algorithm.

Itemset	Support Count
{I1}	3
{I2}	3
{I3}	4
{I5}	4
{I1,I3}	3
{I1,I5}	2
{I2,I3}	2
{I2,I5}	3
{I3,I5}	3
{I2, I3, I5}	2
{I1, I3, I5}	2

Frequent Itemset Lattice
The above table contains all the frequent itemsets in the given transaction data. This table is also called the itemset lattice. We often store this itemset lattice in the form of a tree in the memory. Then, the tree is used to generate the association rules.

ECLAT vs Apriori algorithm:

1. Apriori algorithm is a classical algorithm used to mining the frequent item sets in a given dataset.
2. Coming to Eclat algorithm also mining the frequent itemsets but in vertical manner and it follows the depth first search of a graph.
3. As per the speed,Eclat is fast than the Apriori algorithm.
4. Apriori works on larger datasets where as Eclat algorithm works on smaller datasets.
5. **Memory Requirements:** Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm.
6. **Number of Computations:** The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.