RGPVNOTES.IN

Program : **B.E**

Subject Name: **Data Mining**

Subject Code: **CS-8003**

Semester: **8th**

Unit-IV Association Rule Mining:-Introduction, Basic, The Task and a Naïve Algorithm, Apriori Algorithms, Improving the efficiency of the Apriori Algorithm, Apriori-Tid, Direct Hasing and Pruning(DHP),Dynamic Itemset Counting (DIC), Mining Frequent Patterns without Candidate Generation(FP-Growth),Performance Evaluation of Algorithms,.

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."

Association rule mining is the data mining process of finding the rules that may govern associations and causal objects between sets of items.

Association rules analysis is a technique to uncover how items are associated to each other. There are three common ways to measure association.
So in a given transaction with multiple items, it tries to find the rules that govern how or why such items are often bought together.

Measure 1: Support. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. In Table 1 below, the support of {apple} is 4 out of 8, or 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.
Measure 2: Confidence. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. In Table 1, the confidence of {apple -> beer} is 3 out of 4, or 75%.
Measure 3: Lift. This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. In Table 1, the lift of {apple -> beer} is 1,which implies no association between items. A lift value greater than 1 means that item Y is likely to be bought if item X is bought, while a value less than 1 means that item Y is unlikely to be bought if item X is bought.

The main applications of association rule mining:

- Basket data analysis - is to analyze the association of purchased items in a single basket or single purchase as per the examples given above.

- Cross marketing - is to work with other businesses that complement your own, not competitors. For example, vehicle dealerships and manufacturers have cross marketing campaigns with oil and gas companies for obvious reasons.
- Catalog design - the selection of items in a business' catalog are often designed to complement each other so that buying one item will lead to buying of another. So these items are often complements or very related.

Naïve Algorithm [Searching for Patterns (Naive Pattern Searching)]

Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.

Examples:

Input: txt[] = "THIS IS A TEST TEXT"

pat[] = "TEST"

Output: Pattern found at index 10

Input: txt[] = "AABAACAADAABAABA"

pat[] = "AABA"

Output: Pattern found at index 0

Pattern found at index 9

Pattern found at index 12

A **Naive algorithm** is usually the most obvious solution when one is asked a problem. It may not be a smart algorithm but will probably get the job done (...eventually.)

Eg. Trying to search for an element in a sorted array. A Naive algorithm would be to use a Linear Search.

Apriori Algorithms

Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the

database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

Example 1

Consider the following database, where each row is a transaction and each cell is an individual item of the transaction:

alpha	beta	epsilon	
alpha	beta	theta	a
alpha	beta	epsilon	a
alpha	beta	theta	b

The association rules that can be determined from this database are the following:

1. 100% of sets with alpha also contain beta
2. 50% of sets with alpha, beta also have epsilon
3. 50% of sets with alpha, beta also have theta

we can also illustrate this through a variety of examples.

Example 2

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item: each item, such as "butter" or "bread", is identified by a numerical SKU. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together.

Let the database of transactions consist of following itemsets:

Itemsets
{1,2,3,4}

{1,2,4}
{1,2}
{2,3,4}
{2,3}
{3,4}
{2,4}

We will use Apriori to determine the frequent item sets of this database. To do this, we will say that an item set is frequent if it appears in at least 3 transactions of the database: the value 3 is the support threshold.

The first step of Apriori is to count up the number of occurrences, called the support, of each member item separately. By scanning the database for the first time, we obtain the following result

Item	Support
{1}	3
{2}	6
{3}	4
{4}	5

All the itemsets of size 1 have a support of at least 3, so they are all frequent.

The next step is to generate a list of all pairs of the frequent items.

For example, regarding the pair {1,2}: the first table of Example 2 shows items 1 and 2 appearing together in three of the itemsets; therefore, we say item {1,2} has support of three.

Item	Support
{1,2}	3
{1,3}	1
{1,4}	2
{2,3}	3
{2,4}	4
{3,4}	3

The pairs {1,2}, {2,3}, {2,4}, and {3,4} all meet or exceed the minimum support of 3, so they are frequent. The pairs {1,3} and {1,4} are not. Now, because {1,3} and {1,4} are not frequent, any larger set which contains {1,3} or {1,4} cannot be frequent. In this way, we can prune sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

Item	Support
{2,3,4}	2

in the example, there are no frequent triplets. {2,3,4} is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold.

We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because one of their subsets was already known to be below the threshold.

General Process of the Apriori algorithm

The entire algorithm can be divided into two steps:

Step 1: Apply minimum support to find all the frequent sets with k items in a database.

Step 2: Use the self-join rule to find the frequent sets with k+1 items with the help of frequent k-itemsets. Repeat this process from k=1 to the point when we are unable to apply the self-join rule.

This approach of extending a frequent itemset one at a time is called the "bottom up" approach.

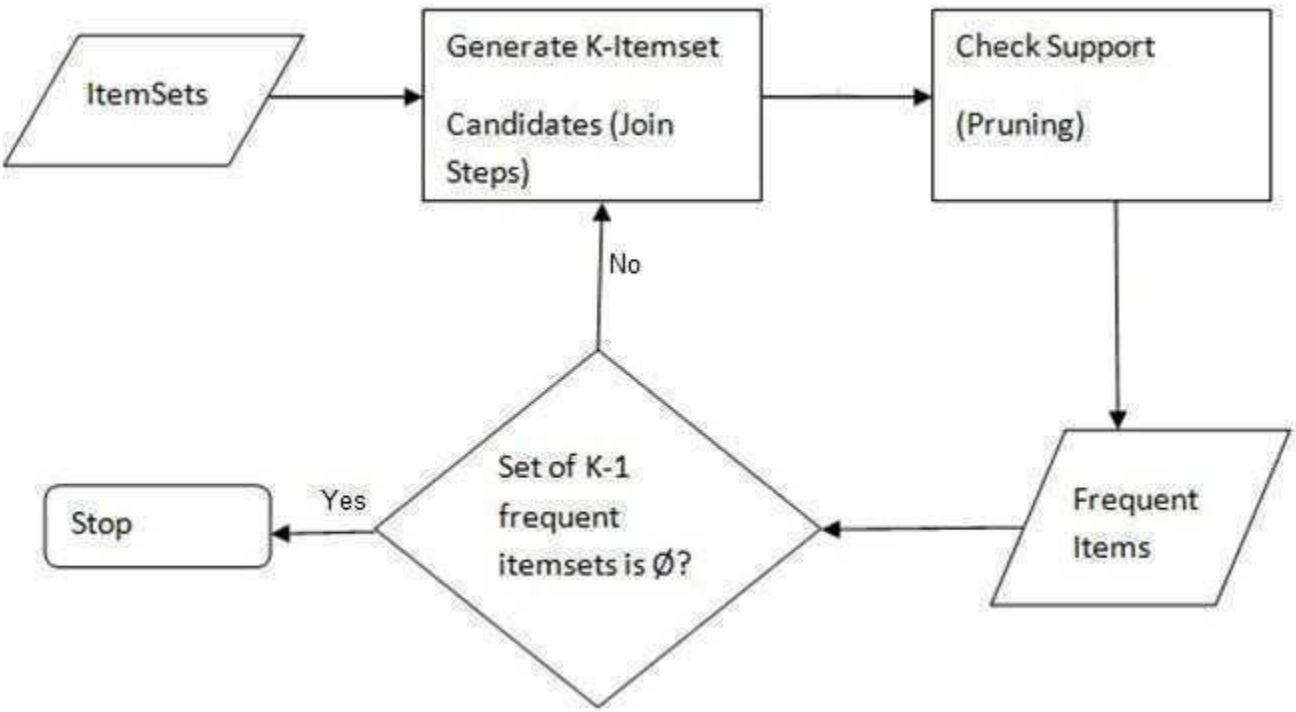

Figure 1: General Process of Apriori Algorithm

Mining Association Rules

Till now, we have looked at the Apriori algorithm with respect to frequent itemset generation. There is another task for which we can use this algorithm, i.e., finding association rules efficiently.

For finding association rules, we need to find all rules having support greater than the threshold support and confidence greater than the threshold confidence.

But, how do we find these? One possible way is brute force, i.e., to list all the possible association rules and calculate the support and confidence for each rule. Then eliminate the rules that fail the threshold support and confidence. But it is computationally very heavy and prohibitive as the number of all the possible association rules increase exponentially with the number of items.

Given there are n items in the set , the total number of possible association rules is .
We can also use another way, which is called the two-step approach, to find the efficient association rules.

The two-step approach is:

Step 1: Frequent itemset generation: Find all itemsets for which the support is greater than the threshold support following the process we have already seen earlier in this article.

Step 2: Rule generation: Create rules from each frequent itemset using the binary partition of frequent itemsets and look for the ones with high confidence. These rules are called candidate rules.

Direct Hashing and Pruning (DHP)

The Direct Hashing and Pruning (DHP) algorithm is in fact, a variation of Apriori algorithm. Both algorithms generate candidate k+1-itemsets from large k-itemsets, and large k+1-itemsets are found by counting the occurrences of candidate k+1-itemsets in the database. The difference of the DHP algorithm is that, it uses a hashing technique to filter out unnecessary itemsets for the generation of the next set of candidate itemsets

In the DHP algorithm, during the support count of Ck , by scanning the database, the algorithm also accumulates information about candidate k+1 itemsets in advance in such a way that, all possible k+1 subsets of items of each transaction after some pruning are hashed to a hash table. Each entry in the hash table consists of a number of itemsets that have been hashed to this entry thus far. Then, this hash table is used to determine the Ck +1 . In order to find Ck +1 , the algorithm generates all possible k+1 itemsets from Lk as in the case of Apriori, then the algorithm adds a k+1 itemset into Ck +1 only if that k+1 itemset passes the hash filtering, i.e. that the entry for k+1 itemset in the hash table is greater than or equal to the minimum support.

In the DHP algorithm, if we can define a large hash table such that each different itemsets is mapped to different locations in the hash table, then the entries of the hash table gives the actual count of each

itemset in the database. In that case, we do not have any false positives and as a result of this, an extra processing for counting the occurrences of each itemset is eliminated. It has also been showed that, the amount of data that has to be scanned during the large itemset discovery is another performance-related issue. Reducing the number of transactions to be scanned and trimming the number of items in each transaction improves the data mining efficiency in later stages.

Dynamic Itemset Counting (DIC)

Introduction

- Alternative to Apriori Itemset Generation
- 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.

Mining Frequent Patterns without Candidate Generation

Mining frequent patterns in transaction databases, time-series databases, and many other kinds of databases has been studied popularly in data mining research. Most of the previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist a large number of patterns and/or long patterns. In this study, we propose a novel frequent-pattern tree (FP-tree) structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns, and develop an efficient FP-tree-based mining method, FP-growth, for mining the complete set of frequent patterns by pattern fragment growth. Efficiency of mining is achieved with three techniques: (1) a large database is compressed into a condensed, smaller data structure, FP-tree which avoids costly, repeated database scans, (2) our FP-tree-based mining adopts a pattern-fragment growth method to avoid the costly generation of a large number of candidate sets, and (3) a partitioning-based, divide-and-conquer method is used to decompose the mining

task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space. Our performance study shows that the FP-growth method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm and also faster than some recently reported new frequent-pattern mining methods.

Apriori vs FP-Growth for Frequent Item Set Mining

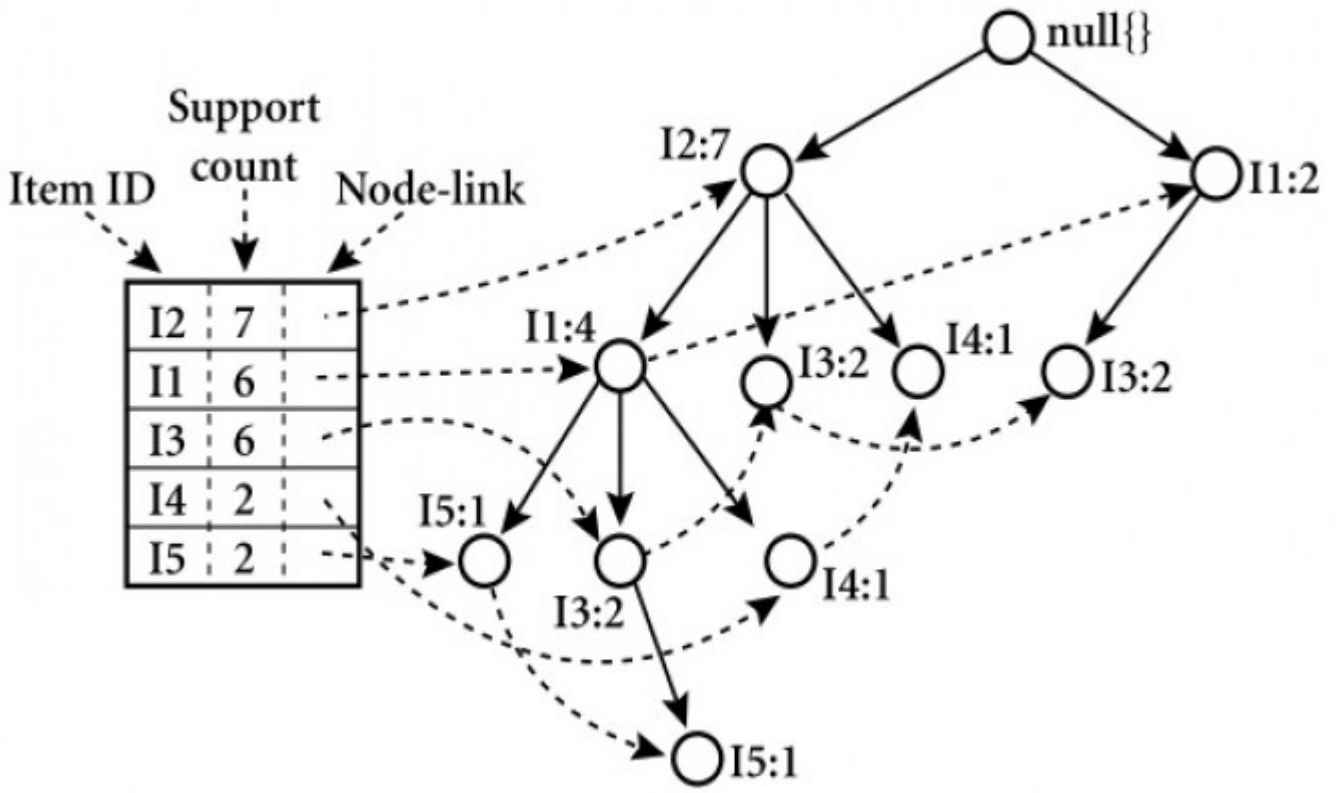

Figure 2: FP Growth

Apriori vs FP-Growth

Algorithm	Technique	Runtime	Memory usage	Parallelizability
Apriori	Generate singletons, pairs, triplets, etc.	Candidate generation is extremely slow. Runtime increases exponentially depending on the number of different items.	Saves singletons, pairs, triplets, etc.	Candidate generation is very parallelizable
FP-Growth	Insert sorted items by frequency into	Runtime increases linearly, depending on the number of	Stores a compact version	Data are very inter dependent, each node

a pattern tree	transactions and items	of the database.	needs the root.

Performance Evaluation of Algorithms

Case Study:

http://shodhganga.inflibnet.ac.in/bitstream/10603/7989/14/14_chapter%205.pdf

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in