IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719
Vol. 3, Issue 1 (Jan. 2013), ||V1|| PP 58-64

Literature Survey on Various Frequent Pattern Mining Algorithm

Varsha Mashoria, Anju Singh
Department of computer science, Barkatullah University Bhopal (M.P) India

Abstract: Data mining discovers hidden pattern in data sets and association between the patterns. In this association rule mining is one of the technique used to achieve the objective of data mining. These rules are effectively used to uncover unknown relationships producing result that can give us a basis for forecasting and decision making. To discover these rules we have to find out the frequent item sets because these item sets are the building blocks to obtain Association rules with a given confidence and support. In this paper we theoretical analyses the extraction for frequent item sets and compare these algorithm.

Keywords **–**association rule mining, confidence, Data mining, frequent item sets, pattern, support.

I. INTRODUCTION

Generally, Data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into knowledgeable information - information that can be used to increase revenue, cuts costs, or both. Data mining, the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses [7]. It allows many users to analyze data from many different sources and dimensions or angles and then categorize it, and summarize the relationships identified. Technically, data mining is the way of finding correlations or patterns among dozens of fields in large relational databases. Data mining, is not only the extraction of hidden predictive information from large databases, it is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining is the tools that can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

II. DATASET ORGANIZATION

Dataset organizations can be processed in two ways they are in horizontally or in vertically. For several decades and especially with the pre-eminence of relational database systems, data is almost always formed into horizontal record structure and then processed vertically. In a horizontal data organization, each transaction contains only items positively associated with a customer purchase. In a horizontal layout, the database is organized as a set of rows, with each row representing a customer's transaction in terms of the items that are purchased in the transaction. There is an alternative approach to this data layout such as vertical layout. It consists of each item associated with a column of values representing the transaction in which it is present. It has smaller effective database size, compact storage of the database and better support of dynamic database.

A market-basket database is a two dimensional matrix where the rows represent individual customer purchase transaction and the columns represent the items on sale.

TID	LIST OF ITEMS
1	A,B,D,E
2	C,D
3	A,D,E
4	A,E
5	B,D,E
6.	B,C

Fig.1 (a)Table 1 Transactions Database

Recently attention has been given to the influence of data organization on the performance of the process of frequent pattern discovery. The discovery of interesting relationships hidden in large datasets is the objective of frequent pattern mining

III. ASSOCIATION RULE MINING

There is many algorithms for finding frequent patterns. Association rule mining first consider by Agrawal has now become one of the main pillars of data mining and knowledge discovery tasks. An association rule is an expression $X \rightarrow Y$, where X is a set of items and Y is usually a single item. It means in the set of transactions, if all

TID	LIST OF ITEMS
1	A,B,D,E
2	C,D
3	A,D,E
4	A,E
5	B,D,E
6.	B,C

Horizontal Layout

LIST OF ITEMS				
A	B	C	D	E
1	1	2	1	1
3	5	6	2	3
5	6		3	4
			5	5

Vertical Layout

Fig.1 (b) Transactions Database

the items in *X* exist in a transaction, then *Y* is also in the transaction with a high probability or in other word It is a method of finding relationships of the form x→y item sets that occur together in a database where X and Y are disjoint item sets [2]. Support and confidence are two key measures for association rule mining. The association rule point that the transactions that contain X, tend to contain Y support. Let I = {I1, I2 …Im} be a set of items. Let D, the task relevant data, be a set of database transactions where each transaction T is a set of items such that T C J.

Support (A=>B) = P (AUB)

Confidence (A=>B) = P (B/A)

IV. ALGORITHMS FOR ASSOCIATION RULES

There are many algorithms for generation association rules. According to the data set organization association rule mining algorithms are also based on these layouts. Some well known algorithms are Apriori[6],partition, Éclat and FP-Growth, but they only do half the job, since they are algorithms for mining frequent item sets.

V. HORIZONTAL LAYOUT ALGORITHM

1. **For each transaction at pass k:**
   -Find all subsets of length k
   -For each subsets, check if s □ C_k
2. **DB scanned entirely for each pass k**

5.1) Apriori algorithm:-

The main motive of Apriori[6] Algorithm is to find associations between different sets of data. It is sometimes referred to as "Market Basket Analysis". Each set of transactions has a number of items. The output of this algorithm is sets of rules that tell us how often items are contained in sets of transactions. There are some key concepts which Apriori i algorithm follow they are as follows:-

•Frequent Item sets: The sets of item which has minimum support (denoted by Lifor ith-Itemset).
•Apriori Property: Any subset of frequent item set must be frequent.
•Join Operation: To find Lk, a set of candidate k-itemsets is generated by joining Lk-1with itself.
•Join Step: Ckis generated by joining Lk-1with itself
•Prune Step: Any (k-1)-item set that is not frequent cannot be a subset of a frequent k-item set
•Pseudo-code:

***Ck*: Candidate item set of size k**
***Lk*: frequent item set of size k**
***L1*= {frequent items};**
for(*k*= 1; *Lk*!=∅; *k*++) do begin
***Ck+1*= candidates generated from *Lk*;**
for each transaction *t*in database do
Increment the count of all candidates in *Ck+1*that are contained in *t*
***Lk+1*= candidates in *Ck+1*with min_support**
end
return∪*kLk*;

This can be better understood by example for which
Consider a database, D in fig 1(a) , 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

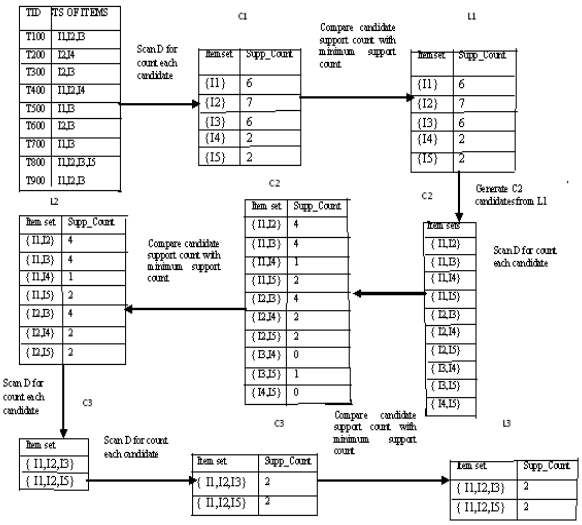

Figure 2 Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

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.
•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 C2is accumulated (as shown in the middle table).
•The set of frequent 2-itemsets, L2, is then determined, consisting of those candidate 2-itemsets in C2having minimum support.
•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 steps 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. How?
•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 C3having minimum support 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. For this refer fig 2. These frequent item sets will be used to generate strong association rules(where strong association rules satisfy both minimum support & minimum confidence).

5.2) F-P Growth algorithm :-

Apriori algorithm is found to be better for association rule mining. Still there are various difficulties faced by apriori algorithm [6].This algorithms suffer from the following two disadvantages

1) It as costly to handle large number of candidate sets.
2) It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching.

Keeping this in mind a new class of algorithm has recently been proposed which avoid the generation of large numbers of candidate sets. we describe one such method called the FP-Tree Growth algorithm. the algorithm involve 2 phases. in 1 phase, it constructs the FP-tree with respect to a given support. the construction of this tree requires 2 phases over the whole database. in phase 2 the algorithm does not use the transaction database any more, but it uses the FP- tree .this can be easily understood by the given example

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
- highly condensed, but complete for frequent pattern mining
- avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
- A divide-and-conquer methodology: decompose mining tasks into smaller ones
- Avoid candidate generation: sub-database test only!
- Consider the same previous example of a database, D refer fig 1(a), consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup = 2/9 = 22 %)
- The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts.

- The set of frequent items is sorted in the order of descending support count.
- The resulting set is denoted as L = {I2:7, I1:6, I3:6, I4:2, I5:2}
- First, create the root of the tree, labeled with "null".
- Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
- The items in each transaction are processed in L order (i.e. sorted order).
- A branch is created for each transaction with items having their support count separated by colon.
- Whenever the same node is encountered in another transaction, we just increment the support count of the common node or Prefix.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.
- Now, The problem of mining frequent patterns in database is transformed to that of mining the FP-Tree.

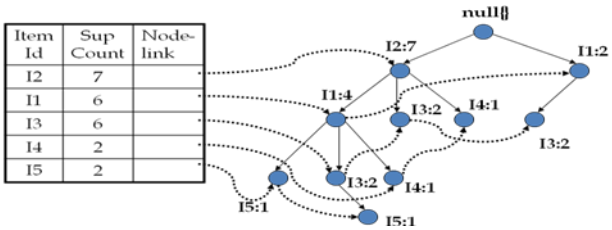

Fig 3 An FP-Tree that registers compressed, frequent pattern information

Steps:

1. Start from each frequent length-1 pattern (as an initial suffix pattern).
2. Construct its conditional pattern base which consists of the set of prefix paths in the FP-Tree co-occurring with suffix pattern.
3. Then, Construct its conditional FP-Tree & perform mining on such a tree.
4. The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-Tree.
5. The union of all frequent patterns (generated by step 4) gives the required frequent itemset.

Now, Following the above mentioned steps:

- Let's start from I5. The I5 is involved in 2 branches namely {I2 I1 I5: 1} and {I2 I1 I3 I5: 1}.
- Therefore considering I5 as suffix, its 2 corresponding prefix paths would be {I2 I1: 1} and {I2 I1 I3: 1}, which forms its conditional pattern base.

Out of these, Only I1 & I2 is selected in the conditional FP-Tree because I3 is not satisfying the minimum support count.

For I1 , support count in conditional pattern base = 1 + 1 = 2

For I2 , support count in conditional pattern base = 1 + 1 = 2

For I3, support count in conditional pattern base = 1

Thus support count for I3 is less than required min_sup which is 2 here. Refer fig 3

Item	Conditional pattern base	Conditional FP-Tree	Frequent pattern generated
I5	{(I2 I1: 1),(I2 I1 I3: 1)}	<I2:2 , I1:2>	I2 I5:2, I1 I5:2, I2 I1 I5: 2
I4	{(I2 I1: 1),(I2: 1)}	<I2: 2>	I2 I4: 2
I3	{(I2 I1: 1),(I2: 2), (I1: 2)}	<I2: 4, I1: 2>,<I1:2>	I2 I3:4, I1, I3: 2 , I2 I1 I3: 2
I2	{(I2: 4)}	<I2: 4>	I2 I1: 4

Fig 4 Mining the FP-tree by creating conditional (sub-)pattern bases.

- Now, we have conditional FP-Tree with us.
- All frequent patterns corresponding to suffix I5 are generated by considering all possible combinations of I5 and conditional FP-Tree.
- The same procedure is applied to suffixes I4, I3 and I1.
- Note: I2 is not taken into consideration for suffix because it doesn't have any prefix at all. for this refer fig 4.
- Performance study shows
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan

- Basic operation is counting and FP-tree building

.

VI. VERTICAL LAYOUT ALGORITHM

1. **Counting item sets AB:intersect{A} with {B}**
2. **No need to scan database in entirely**
- **For item sets C, all relevant t is in {C}**
3. **No need for multiple passes**
- **Intersect larger item sets in pipelined fashion**
- **{(C_{k+1})} >= {(C_k)}**
- **Does not work for 1 item sets.**

6.1) Eclat algorithm:-

Éclat [5, 3, 1] is basically a depth-first search algorithm using set intersection. It partitioned the item sets and after it no communication take place between them.atmost there should be 3scans of local database refer fig 5. It uses a vertical database layout i.e. instead of explicitly listing all transactions; each item is stored together with its cover and uses the intersection based approach to calculate the support of an item set. In this way, the support of an item set X can be easily calculated by simply intersecting the covers of any two subsets Y, Z ⊆ X, such that Y U Z = X. It states that, when the database is stored in the vertical layout, the support of a set can be counted much easier by simply intersecting the covers of two of its subsets that together give the set itself.

Tid	X
100	{a, b, c, d, e, f}
200	{a, b, c, d, e}
300	{a, d}
400	{b, d, f}
500	{a, b, c, e, f}

Fig 5 Table 2: An example preprocessed transaction database

The Éclat algorithm is as given below.

Input: D, K, i ⊆ I
Output: F[I](D, K)
1: F[I] :={}
2: for all I L I occurring in D do
3: F[I] := F[I] ∪ {I ∪ {i}}
4: // Create Di
5: Di: = {}
6: for all j L I occurring in D such that j>I do
7: C := cover({i}) O cover({j})
8: if |C| >= K then
9: Di: = Di ∪ {(j, C)}
10: end if
11: end for
12: //Depth-first recursion
13: Compute F[I ∪ {i}](Di, K)
14: F[I] := F[I] ∪ F[I ∪ {i}]
15: end for

In this algorithm each frequent item is added with the output set. The frequent item sets are determined using simple tid-list-intersection in a depth first graph. After that, for every such frequent item i the i-projected database. Di is created. all this is done by first finding every item j that frequently occur with i The support of this set {i, j} is computed by intersecting the covers of both items. If {i, j} is frequent, then j is inserted into Di together with its cover. The rearrangement is performed at every recursion step of the algorithm .then the algorithm is called recursively to find all the frequent item sets in the new database. Di . it essentially generates the candidate item sets using only the join step from Apriori. Again all the items in the database is rearranged in ascending order of support to reduce the number of candidate item sets that is generated, and hence reduce the number of intersections that need to be computed and the total size of the covers of all generated item sets. Since the algorithm doesn't fully exploit the monotonicity property but generates a candidate item sets based on only two of its subsets, the number of candidate item sets that are generated is much larger as compared to a breadth-first approach such as Apriori. As a comparison, Éclat Essentially generates candidate item sets using only the join step from Apriori [2], since the item sets necessary for the prune step are not available. A technique that is regularly used is to reorder the items in

Support ascending order to reduce the number of candidate item sets that is generated. In Éclat, such reordering can be performed at every recursion step in the algorithm. Also that at a certain depth d,the covers of at most all k-item sets with the same k-1 prefix are stored in main memory with k<d Because of the item reordering, this number is kept small.

6.1) SaM algorithm:-

The SaM (Split and Merge) algorithm established by [4] is a Simplification of the already fairly simple RElim (Recursive Elimination) algorithm. While RElim represents a (conditional) database by storing one transaction list for each item (partially vertical representation), the split and merge algorithm employs only a single transaction list refer fig 6 (purely horizontal representation), stored as an array. This array is processed with a simple split and merge scheme, which computes a conditional database, processes this conditional database recursively, and finally eliminates the split item from the original (conditional) database. SaM preprocesses a given transaction following the steps below:

1. The transaction database is taken in its original form. 2. The Frequencies of individual items are determined from this input in order to be able to discard infrequent items immediately. 3. The (frequent) items in each transaction are sorted according to their frequency in the transaction database, since it is well known that processing the items in the order of increasing frequency usually leads to the shortest execution times. 4. The transactions are sorted lexicographically into descending order, with item comparisons again being decided by the item frequencies; here the item with the higher frequency precedes the item with the lower frequency. 5. The data structure on which SaM operates is built by combining equal transactions and setting up an array, in which each element consists of two fields: **.**

An occurrence counter and a pointer to the sorted transaction (array of contained items). This data structure is then processed recursively to find the frequent item sets. The basic operations of the recursive processing are based on depth-first/divide-and conquer scheme. In the split step the given array is split with respect to the leading item of the first transaction. All array elements referring to transactions starting with this item are transferred to a new array. The new array created in the split step and the rest of the original arrays are combined with a procedure that is almost identical to one phase of the well-known merge sort algorithm. The main reason for the merge operation in SaM is to keep the list sorted, so that: 1.All transactions with the same leading item are grouped together and 2.Equal transactions (or transaction suffixes) can be combined, thus reducing the number of objects to process.

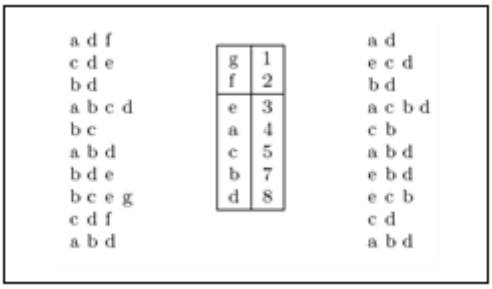

Figure 6: Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted accordingly with respect to their frequency (right).

Each transaction is represented as a simple array of item identifiers (which are integer numbers). The transaction list is prepared which are stored in a simple array, each element of which contains a support counter and a pointer to the head of the list. The list elements themselves consist only of a successor pointer and a pointer to the transaction. The transactions are inserted one by one into this structure by simply using their leading item as an index. Refer fig 7.

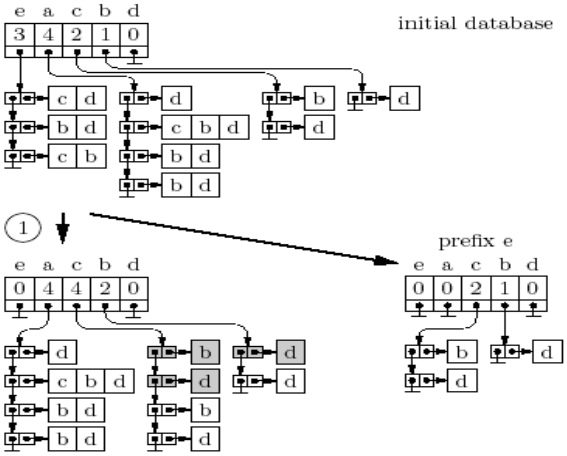

Figure 7 SaM algorithm

VII. CONCLUSION

In this paper We summarize and organize recent research results in a new way that integrates and adds understanding to work in the field. Firstly We survey the two horizontal layout of data's algorithm for association rule mining then I survey the vertical layout data's algorithm for association rule mining. From this I found that in horizontal layout FP-tree works better than the Apriori algorithm because Apriori generates candidate sets and FP-Growth uses specialized data structures (no candidate sets). Apriori visits each transaction when generating a new candidate sets; whereas FP-Growth does not and in vertical layout SaM algorithm works better than Eclat. Because For higher support threshold execution time of SaM is comparatively less.

REFERENCE

[1]. C.Borgelt. Efficient Implementations of Apriori and Eclat. Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL).CEUR Workshop Proceedings 90, Aachen, Germany 2003.

[2]. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 307–328. MIT Press, 1996.

[3]. M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. NewAlgorithms for Fast Discovery of Association Rules. Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97), 283–296. AAAI Press, Menlo Park, CA, USA 1997.
[4]. C. Borgelt. SaM: Simple Algorithms for Frequent Item Set Mining. IFSA/EUSFLAT 2009 conference- 2009.
[5]. J. Han, and M. Kamber, 2000. Data Mining Concepts and Techniques. Morgan Kanufmann.
[6]. Dr. M. Dhanabhakyam , Dr. M. Punithavalli, A survey on Data mining algorithm for market basket analysis. Global Journal of Computer Science and Technology (Volume 11 Issue 11 Version 1.0 July 2011 Type: Double Blind Peer Reviewed International Research Journal) Publisher: Global Journals Inc. (USA).
[7]. http://datawarehousingtips.blogspot.in/2009/01/ data-mining-intro.html.
[8]. Klong Luang, Pathumthani Application of a Mining Algorithm to Finding Frequent Patterns in a Text Corpus: A Case Study of the Arabic International Journal of Software Engineering and Its Applications (Vol. 6, No. 3, July, 2012).
[9]. K.Lakshmi, Dr. T. Meyyappan., FREQUENT SUBGRAPH MINING ALGORITHMS - A SURVEY AND FRAMEWORK FOR CLASSIFICATION.
[10]. Byung Joon Park, Sang Young Kim, and Lynn Choi , Efficient Frequent Pattern Mining Based on a Condensed Tree Structure .
[11]. Sunil Joshi,Dr.R.S.Jadon,and Dr.R.C.Jain , An implementation of pattern mining algorithm using dynamic functio.International journal of computer application(vol 9-No.9.November 2010).
[12]. S. M. FAKHRAHMAD AND GH. DASTGHAIBYFARD An Efficient Frequent Pattern Mining Method and its Parallelization in Transactional Databases JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 511-525 (2011).