Discovering Frequent Patterns Mining Procedures

Poonam Sengar, Bhagyshri Lachhwani, Mehul Barot

Abstract— Efficient algorithm to discover frequent pattern are crucial in data mining research. Finding frequent itemsets is computationally the most expensive step in association rule discovery .To address these issues we discuss popular techniques for finding frequent itemsets in efficient way. In this paper we provide the survey list of existing frequent itemsets mining techniques and its analysis.

Keywords— Association rules, data mining, frequent itemsets, FPM, minimum support.

I. INTRODUCTION

Data mining also known as Knowledge Discovery and Database (KDD) has been perceived as a new research is for database research. The aim of data mining is to automate the process of finding interesting patterns and trends from a given data. Discovering of association rules is an important data mining task. A procedure so called mining frequent itemsets is a core step of association rules discovering introduce in [1].

Past transaction analysis is a commonly used approach in order to improve the quality of such decision. As pointed in [2], the progress in bar-code technology has made it possible for retail organizations to collect the store massive amount of sales data so called Basket data that stores item purchased per transaction basis. Most existing algorithm, therefore, are focused on mining frequent itemsets [7]. Several algorithm have been proposed to mine frequent itemsets. A classical algorithm is Apriori introduced in[1]. Variation of Apriori were proposed so far such as hash-based algorithm [3], a dynamic itemset counting technique (DIC) [4] and a Partition algorithm. In addition, many research have been developed algorithms using tree structure, such as H-mine and FP- growth [5].

There are two main strategies for mining frequent itemsets: the candidate generation-and-test approach and the pattern growth approach. Apriori [2] and its several variation belong to the first approach, while FP-growth [5] and H-mine are example of the second. Apriori algorithm [2] suffer from the problem spending much of their time discard the infrequent candidate on each level. The FP-growth (Frequent Pattern -Growth) [5] algorithm differs basically from the level-wise algorithm, that use a "candidate generate and test" approach.

II. PROBLEM STATEMENT

Formally as defined in [1], the problem of mining association rules is started as follows: Let $I = \{i_1, i_2, \ldots . i_m\}$. Let D be a set of transactions, where each transaction T is a set of item such that $T \subseteq I$. Associated with each transaction is unique identifier, called its transaction id is *TID*. A transaction T contains X, a set of some items in I, if $X \subseteq T$ *An association rule is* animplification of the form $X \Rightarrow Y$ where $X \subset I$, $Y \subset I$ and $X \cap Y = \varphi$. . The meaning of such a rule is that transactions in database, which contain the items in X, tend to also contain the items in Y. The rule $X \subseteq Y$ holds in the transaction set D with confidence c if among those transactions that contains X $c\%$ of them also contain Y. The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. The problem of mining association rules that have support and confidence greater than the user-specified minimum support and minimum confidence respectively. Conventionally, the problem of discovering all association rules is composed of the following two steps: 1) Find the large itemsets that have transaction support above a minimum support and 2) From the discovered large itemsets generate the desired association rules. The overall performance of mining association rules can be achieved by using the first step. After the identification of large itemsets, the corresponding association rules can be derived in a straightforward manner. In this paper, we have developed a method to discover large itemsets from the transaction database, thus finding a solution for the first sub problem.

A. Algorithms For Mining Frequent Itemsets

In this each row of database represents a transaction which has a transaction identifier (TID), followed by a set of items. An example of horizontal layout dataset is shown in Table 1. In vertical layout data set, each column corresponds to an item, followed by a TID list, which is the list of rows that the item appears. An example of vertical layout database set is as shown in diagram for the Table 2.

Table 1: Horizontal layout based database

TID	ITEMS
T1	I1, I2, I3, I4, I5, I6
T2	I1, I2, I4, I7
T3	I1, I2, I4, I5, I6
T4	I1, I2, I3
T5	I3, I5

Manuscript received on January, 2013.

Poonam Sengar, ME Computer Engineering, LDRP - ITR, Gandhinagar. India.

Bhagyshri Lachhwani, ME Computer Engineering, LDRP- ITR, Gandhinagar. India.

Prof. Mehul Barot, Computer Department Name, University LDRP - ITR, Gandhinagar. India.

Table 2 : Vertical layout based database

ITEM	TID_list
I1	T1, T2, T3, T4
I2	T1, T2, T3, T4
I3	T1, T4, T5
I4	T1, T2, T3
I5	T1, T3, T5
I6	T1, T3
I7	T2

(1). Apriori Algorithm

The first algorithm for mining all frequent itemsets and strong association rules was the AIS algorithm. Shortly after that, the algorithm was improved and renamed Apriori. Apriori algorithm is, the most classical and important algorithm for mining frequent itemsets. Apriori is used to find all frequent itemsets in a given database DB. The key idea of Apriori algorithm is to make multiple passes over the database. It employs an iterative approach known as a breadth-first search (level-wise search) through the search space, where k-itemsets are used to explore (k+1)-itemsets.

The working of Apriori algorithm is fairly depends upon the Apriori property which states that" All nonempty subsets of a frequent itemsets must be frequent" [1]. It also described the anti monotonic property which says if the system cannot pass the minimum support test, all its supersets will fail to pass the test [1],[2]. Therefore if the one set is infrequent then all its supersets are also frequent and vice versa. This property is used to prune the infrequent candidate elements. In the beginning, the set of frequent 1-itemsets is found. The set of that contains one item, which satisfy the support threshold, is denoted by L_1. In each subsequent pass, we begin with a seed set of itemsets found to be large in the previous pass. This seed set is used for generating new potentially large itemsets, called candidate itemsets, and count the actual support for these candidate itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large (frequent), and they become the seed for the next pass. Therefore, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The feature first invented by [1] in Apriori algorithm is used by the many algorithms for frequent pattern generation.

The basic steps to mine the frequent elements are as follows [2]:

- Generate and test: In this first find the 1-itemset frequent elements L_1 by scanning the database and removing all those elements from C which cannot satisfy the minimum support criteria.
- Join step: To attain the next level elements C_K join the previous frequent elements by self join i.e. $L_{K-1} * L_{K-1}$ known as Cartesian product of L_{K-1}. i.e. This step generates new candidate k-itemsets based on joining L_{K-1} with itself which is found in the previous iteration. Let C_K denote candidate k-itemset and L_1 be the frequent k-itemset.
- Prune step: C_K is the superset of L_1 so members of C_K may or may not be frequent but all K-1 frequent itemsets are included in C_K thus prunes the C_K to find K frequent itemsets with the help of Apriori property. i.e. This step eliminates some of the candidate k-itemsets using the Apriori property. A scan of the database to determine the count of each candidate in C_K would result in the determination of L_1 (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_1). C_K, however, can be huge, and so this could involve grave computation. To shrink the size of C_K, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of candidate k-itemset is not in L_{K-1} then the candidate cannot be frequent either and so can be removed from C_K. Step 2 and 3 is repeated until no new candidate set is generated.

To illustrate this, suppose n frequent 1-itemsets and minimum support is 1 then according to Apriori will generate $n^2 + (n\ 2)$ candidate 2-itemset (n 3) candidate 3-itemset and so on. The total number of candidates generated is greater than $\Sigma_{K=1}^{n}$ (n k). Therefore suppose there are 1000 elements then 1499500 candidate are produced in 2-itemset frequent and 166167000 are produced in 3-itemset frequent.

It is no doubt that Apriori algorithm successfully finds the frequent elements from the database. But as the dimensionality of the database increase with the number of items then:

- More search space is needed and I/O cost will increase.
- Number of database scan is increased thus candidate generation will increase results in increase in computational cost.

Therefore many variations have been takes place in the Apriori algorithm to minimize the above limitations arises due to increase in size of database. These subsequently proposed algorithms adopt similar database scan level by level as in Apriori algorithm, while the methods of candidate generation and pruning, support counting and candidate representation may differ.

The algorithms improve the Apriori algorithms by:

- Reduce passes of transaction database scans.
- Shrink number of candidates.
- Facilitate support counting of candidates.

(2). Direct Hashing And Pruning (DHP) Algorithm

It is absorbed that reducing the candidate items from the database is one of the important task for increasing the efficiency. Thus a DHP technique was proposed [3] to reduce the number of candidates in the early passes C_k for $k > 1$ and thus the size of database. In this method, support is counted by mapping the items from the candidate list into the buckets which is divided according to support known as Hash table structure. As the new itemset is encountered if item exist earlier then increase the bucket count else insert into new bucket. Thus in the end the bucket whose support count is less the minimum support is removed from the candidate set.

In this way it reduce the generation of candidate sets in the earlier stages but as the level increase the size of bucket also increase thus difficult to manage hash table as well candidate set.

(3). Eclat Algorithm

It is a set intersection, depth first search algorithm [6], unlike the Apriori. It uses vertical layout database and each item use intersection based approach for finding the support. In this way, the support of an itemset P can be easily computed by simply intersecting of any two subsets Q, R ⊆ P, such that P = Q U R.

In this type of algorithm, for each frequent itemset i new database is created D_i. This can be done by finding j which is frequent corresponding to i together as a set then j is also added to the created database i.e. each frequent item is added to the output set. It uses the join step like the Apriori only for generating the candidate sets but as the items are arranged in ascending order of their support thus less amount of intersection is needed between the sets. It generates the larger amount of candidates then Apriori because it uses only two sets at a time for intersection [6]. There is reordering step takes place at each recursion point for reducing the candidate itemsets.

In this way by using this algorithm there is no need to find the support of itemsets whose count is greater than 1 because Tid-set for each item carry the complete information for the corresponding support. When the database is very large and the itemsets in the database corresponding also very large then it is feasible to handle the Tid list thus it produce good results but for small databases its performance is not up to mark.

(4).FP-Growth Algorithm

FP-tree algorithm [5] is based upon the recursively divide and conquers strategy; first the set of frequent 1-itemset and their counts is discovered. With start from each frequent pattern, construct the conditional pattern base, then its conditional FP-tree is constructed (which is a prefix tree.). Until the resulting FP-tree is empty, or contains only one single path. (Single path will generate all the combinations of its sub-paths, each of which is a frequent pattern). The items in each transaction are processed in L order. (i.e. items in the set were sorted based on their frequencies in the descending order to form a list).

The detail step is as follows: [5]

FP-Growth Method: Construction of FP-tree

- Create root of the tree as a "null".
- After scanning the database D for finding the 1-itemset then process the each transaction in decreasing order of their frequency.
- A new branch is created for each transaction with the corresponding support.
- If same node is encountered in another transaction, just increment the support count by 1 of the common node.
- Each item points to the occurrence in the tree using the chain of node-link by maintaining the header table.

After above process mining of the FP-tree will be done by Creating Conditional (sub) pattern bases:

- Start from node constructs its conditional pattern base.
- Then, Construct its conditional FP-tree & perform mining on such a tree.
- Join the suffix patterns with a frequent pattern generated from a conditional FP-tree for achieving FP-growth.
- The union of all frequent patterns found by above step gives the required frequent itemset.

In this way frequent patterns are mined from the database using FP-tree.

Definition: Conditional pattern base [5]

A "sub database" which consists of the set of prefix paths in the FP-tree co-occurring with suffix pattern.eg for an itemset X, the set of prefix paths of X forms the conditional pattern base of X which co-occurs with X.

Definition: Conditional FP-tree [5]

The FP-tree built for the conditional pattern base X is called conditional FP-tree.

(5). Frequent Item Graph (FIG) Algorithm

The FIG [7] algorithm is a novel method to find all the frequent itemsets quickly. It discovers all the frequent itemsets in only one database scan. The construction of the *graphical structure* is done in two phases, where the first phase requires a full I/O scan of the dataset and the second phase requires only a full scan of frequent 2-itemsets. The first initial scan of the database identifies the frequent 2-itemsets with a minimum support. The goal is to generate an ordered list of frequent 2-itemsets that would be used when building the *graphical structure* in the second phase.

The first phase starts by arranging the entire database in the alphabetical order. During the database scan the number of occurrences of frequent 2-itemsets is determined and infrequent 2-itemsets with the support less than the support threshold are weeded out. Then the frequent 2-itemsets are ordered in the alphabetical order. Phase 2 of constructing the *graphical structure* requires a complete scan of the ordered frequent 2-itemsets. The ordered frequent 2-itemsets are used in constructing the *graphical structure.*

Advantages:

- The quick mining process that does not use candidates.
- Mining the set of all frequent itemsets in the database by scanning the entire database only once.
- I/O costs spent in mining the set of all frequent itemsets will be reduced.

Disadvantage:

- The second phase requires a full scan of frequent 2-itemsets that would be used when building the graphical structure.

(6). MFIPA Algorithm

Mining Frequent Itemsets Based On Projection Array (MFIPA) algorithm [8] of mining frequent itemsets based on projection array. The data structure PArray is generated by using data horizontally and vertically firstly, it stores all the frequent 1-itemsets and those items that co-occurrence with signal frequent item. And the usage of PArray avoids the generation of large numbers of candidate itemsets and redundant detection.

Theorem 1. Let PArray[j].item be a frequent itemset with sup(PArray[j].item) = minsup, then there exists no item PArray[j].item < i and i∉PArray[j].projection, such that PArray[j].item∪ I is a frequent itemset.

Theorem 2. Let item be a single item, the projection of it is *projection(item)* = $a_1,a_2\ldots\ldots a_m$, if item is combined with the 2^m-1 nonempty subsets of $a_1,a_2\ldots\ldots a_m$, the supports of the resulting itemsets are all sup(item).

According to Theorem 1 and 2, find some frequent itemsets which have the same support as single frequent item by connecting the single frequent item with every nonempty subsets of its projection, then the frequent itemsets are extended by using depth-first search strategy when the support of single frequent item is greater than minsup. The support of resulting itemsets after extending can be figured out by intersection operation of *tidset(items)*, if the support is greater than minsup, the resulting itemsets will be reserved, and on the contrary, the resulting itemsets will be pruned.

III. COMPARISION OF DIFFERENT ALGORITMS FOR FPM

- All the algorithms produce frequent itemsets on the basis of minimum support.
- Apriori algorithm is quite successful for market based analysis in which transactions are large but frequent items generated is small in number.
- Apriori works well in sparse data sets since most of the candidates that Apriori generates turn out to be frequent patterns.
- But in sparse data sets, FP-tree cannot compress data effectively as what it does on dense data sets. Constructing FP-tree over sparse data sets recursively has its overhead.
- The memory usage of Apriori does not scale well as the support threshold goes down because it needs to store level-wise frequent pattern and generate next level candidates. But in FP-tree, do not need to store any frequent patterns or candidates. Once a pattern is found, it is output immediately and never read back.
- For FP-Tree performs better than all discussed above algorithms because of no generation of candidate sets but the pointes needed to store in memory require large memory space.
- DHP works well at early stages and performance deteriorates in later stages and also results in I/O overhead. In DHP algorithm colliding problem is encounter. At the same bucket two different entry is encountered.
- Execution time of the first pass of DHP is slightly larger than Apriori due to extra overhead required for generating H_2.DHP incurs significantly smaller execution times than Apriori in later passes.
- Apriori Scans full database for every pass whereas DHP scan full database for early two passes and then scan reduced database D_k thereafter.
- Vertical Layout based algorithms claims to be faster than Apriori but require larger memory space then horizontal layout based because they needs to load candidate, database and TID list in main memory.
- For FIG algorithm is a quick mining process that does not use candidates but requires a full scan of frequent 2-itemsets that would be used when building the graphical structure in second phase.
- For MFIPA algorithm works well for dense datasets but performance decreases for sparse datasets.

Therefore these algorithms are not sufficient for mining the frequent itemsets for large transactional database.

IV. CONCLUSION

Studies of Frequent Itemset (or pattern) Mining is acknowledged in the data mining field because of its broad applications in mining association rules, correlations, and graph pattern constraint based on frequent patterns, sequential patterns, and many other data mining tasks. The major challenge we found by studying different algorithms present in this paper in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of itemsets are generated. Therefore, pruning unimportant patterns can be done effectively in mining process and that becomes one of the main topics in frequent pattern mining. There are many algorithms were found for mining frequent itemsets but there are all having some disadvantage. We try to optimize the process of finding patterns which should be efficient, scalable and can detect the important patterns which can be used in various ways in future.

REFERENCES

1. Agrawal, R. Imielinski, T. and Swami. *Mining Association Rules between Sets of Items in Large Databases, Proc. of ACM SIGMOD*, Washington DC, 22:207-216, ACM Press
2. Agrawal, R. and Srikant, R. *Fast Algorithms for Mining Association Rules in Large Databases, Proc. 20th Intl Conf. Very Large Data Bases*, pp. 478-499, Sept. 1944.
3. Park, J.S. Chan, M. and Yu, P.S. *An Effective Hash-based Algorithm for Mining Association Rules, In Proc. of ACM SIGMOD*, pp. 175-186. ACM, May 1995.
4. Brin, S. Motwani, R. Ullman, J. and Tsur, S. *Dynamic itemset counting and implication rules for market basket data, In Proc. of ACM SIGMO*, pp.225-264, 1997.
5. Han, J. Pei, J. and Yin, Y. *Mining Frequent Patterns without Candidate Generation, Proc. of ACM SIGMOD Conf.*, Dallas, TX, 2000
6. C.Borgelt; , "Efficient Implementations of Apriori and Eclat," *In Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations, CEUR Workshop Proceedings 90, Aachen, Germany*, 2003
7. Kumar, A.V.S.; Wahidabanu, R.S.D.; , "A Frequent Item Graph Approach for Discovering Frequent Itemsets," *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on* , vol., no., pp.952-956, 20-22 Dec. 2008
8. Hai – Tao He; Hai - Yan Cao; Rui-Xia Yao; Jia-Dong Ren; Chang-Zhen Hu; , "Mining frequent itemsets based on projection array," *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on* , vol.1, no., pp.454-459, 6-14 July 2010