Frequent Itemset mining over transactional data streams using Item-Order-Tree

Pramod S.
Reader, Information Technology,
M.P.Christian College of Engineering and Tech.,
Bhilai, C.G., INDIA.

O.P. Vyas
Professor,
IIIT Allahabad
U.P., INDIA.

Abstract - The association rule mining is one of the important area for research in data mining. In association rule mining online association rule mining is one of the hottest area due to the reason that the knowledge embedded in the data stream is more likely to be changed as time goes by. This paper proposes an algorithm as well as a data structure for online data mining. In this method the pruning in the data structure as well as the frequent itemset generation will be based on the request. The data structure which we introducing will have the capability to maintain the transactions in the sorted order. Every transaction can be extracted from the item-Order-Tree as by doing the traversal in depth. Frequent itemset can be generated as by do the traversal from the parent node that the user requested for. This ItemOrder-Tree improves the performance of the online association rule mining.

Keywords- Frequent Itemset, Freequent Itemset Mining , Online Data Mining**,** Item-Order-Tree

I. INTRODUCTION

In recent years the size of the database of any transaction storage increased rapidly. This has led to a growing interest in the development of tools capable in the automatic extraction of knowledge from data. The term data mining or knowledge discovery in database has been adopted for a field of research dealing with the automatic discovery of implicit information or knowledge within the databases. The implicit information within databases, mainly the interesting association relationships among sets of objects that lead to association rules may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications.

The problem of mining frequent itemsets came out first as a sub-problem of mining association rules[1]. Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases such as association rules, correlations, sequences, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems. The original motivation for searching association rules came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Association rules describe how often items are purchased together. For example, an association rule states that four out of five customers that bought beer also bought chips. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others.

II. PRILIMINARIES

A. Need of Frequent Itemset Mining

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. Efficient algorithms for mining frequent itemsets are crucial for mining association rules as well as for many other data mining tasks. The major challenge found 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. Consequently, the main aim is 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.

B. Preliminaries of New Approach

Different methods[2,3,4] introduced by different researchers are generated the frequent itemsets by using the candidate generation as well as without candidate generation. The different data structures[1,12,13,14] are used for frequent itemset mining and the methods which they used for input data to the data structure is also different. The new method examines the transactions one by one without the candidate generation and it will keep track of the occurrence count of each item in a monitoring lattice called Item-Order-Tree. The new itemset will be entered in the data structure after sorting the itemset.

The new approach will work as below:

[1] Sort the itemset I.
[2] Insert the items in the itemset in the sorted order and can be accessed in the same order while we traverse in depth. In each node one counter for count the occurrence of item.
[3] Consider all the branches which having the item to be included for finding the frequent itemset.
[4] The total number of nodes available in the ItemOrder-Tree is
[5] $\sum_2^n N$ n is the maximum number of unique item
    a. N is the total number of nodes.
[6] The parameters in the counters in the node are nC_N where N is the item ID of the counter.

[7] There is no need to store the minimum support and the confidence in the lattice as it can store separately.

To design a compact data structure for efficient online data mining let us examine an example. Let the transactions as given below in the first two columns of Table 1. A data structure can be designed based on the observations below:

[1] Since the online transactions are streaming in, it is necessary to store the items in a data structure to save time and space with a frequency count.
[2] If all the items can be stored in some order in a data structure, it may be possible to avoid repeatedly scanning the previous transactions as by storing in database.
[3] If multiple transactions share a set of frequent items, it can be registered as by increment the count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.
[4] If more than one transactions share a common prefix, according to some sorted order frequent items, the shared parts can be merged using one prefix structure as by registering the count properly.

Table 1: A transaction database as example.

TID	Item bought	Items in order
100	f, a, c, d, g, i, m, p	a, c, d, f, g, i, m, p
200	a, b, c, f, l, m, o	a, b, c, l, m, o
300	b, h, f, j, o	b, f, h, j, o
400	k, s, b, p, c	b, c, k, p, s
500	c, e, a, l, m, n, f, p	a, c, e, f, l, m, n, p

[1] The different advantages of the data structure used as below

[2] It will be ordered on transactions received
[3] It will be in ascending order of the number of occurrence of each item in the transactions contain the same itemsets.
[4] Traversal for finding the frequent itemset will be fast.
[5] More frequently occurring items are more likely to be shared and thus they are arranged top of the tree.

III. ITEM-ORDER-TREE

The ItemOrder-Tree is the tree used as the data structure. The temporary storage of itemset can be reduced by storing of the node item along with the predecessors. This can be achieved by adding an item in the tree immediately after the root node if it is a single item in the transaction and not in the lattice as the first node after the null node. Let there is a single item transaction as {a} and a is already existing in the lattice then no need to add the same item otherwise add that immediately after the root node by creating a new node. If there is another transaction as {d,a,c} then in our new data structure the transaction has to be ordered and after that it can be entered in the Itemset-Tree in the new order, it will be {a},{c},{d} while traverse in depth.

A. Definition 1. Item-Order-Tree

Let T_k be an Item-Order-Tree and let e denote the itemsetand its elements are e= $\{i_1, i_2, i_3 \ldots.\}$. Set of its subsetsareadded in to the tree after reordering the elements in e.

[1] The new node created if $i_1 \subseteq I$ and $i_1 \notin T_k$. From the newly created node i_1 create the child node and the item is i_2.
[2] Any of the node available then its count incremented and create the non existing element node.
[3] Add a transaction under one node along with all the items of its parents. The same transaction with n-1 itemset is in the tree then the n^{th} item to be added and increment the counter for all other.
[4] The concatenation of the parent node items to the new node to reduce the temporary storage.

B. Definition 2:Item-Order-Tree node structures

Given an ItemOrder-Tree T_k. For finding frequent itemsets, a data stream can be defined as follows:

Let $I_1=\{i_1,i_2, \ldots ,i_n\}$ be a set of items in the transaction of any application domain.

[1] An itemset e is a set of items such that $e \in (2^I - \{\emptyset\})$ where 2^I is the power set of I. The length |e| of an itemset e is the number of items that form the itemset and it is denoted by |e|-itemset.
[2] A transaction is a subset of I and each transaction has a unique transaction identifier TID. A transaction generated at the k^{th} turn is denoted by I_k.
[3] The stream D_k is composed of all transactions that have ever been generated so far i.e. $D_k = <T_1, T_2, \ldots , T_k>$ and the total number of transactions in D_k is denoted by $|D|_k$.
[4] When a transactions I_k is generated, the current count $C_k(e)$ of an itemset e is the number of transactions that contain the itemset among the k transactions.

The below given fig:1 is the example of a Item-Order-Tree.

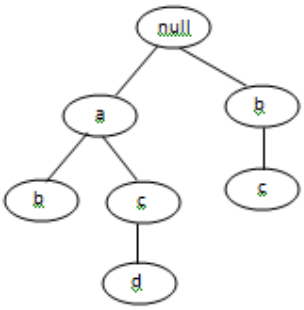

Figure 1: Item-Order-Tree T_k

C. Algorithm 1: (Item-Order-Tree Construction*)*

Input : A transaction T_k in the data stream.

Output: ItemOrder-Tree of the given transactions.

Method: The ItemOrder-Tree is constructed as follows.

[1] Sort the items in the transaction T_k
[2] Create the root node of an Item-Order-Tree, T, and label it as "null". For each transaction do the following.
   a. Select the item from the sorted transaction list of Tk. Let the sorted list in the transaction be [i|I], where i is the first item and I is the remaining items in the Tk. Call the function Insert-ItemOrder-Tree([i|I],T).
   b. The function Insert-ItemOrder-Tree ([i|I],T) is performed as follows. If T has a child N such that
   c. N.item-name=i.item-name then increment count of N by 1;else create a new node N with its count initialized as 1, its parent linked to T, and its node-link linked to the nodes with the same item-name via the node link structure. If I is non empty, call insert-ItemOrder-Tree(I,N) recursively.

IV. FREQUENT ITEM GENERATION

It is working almost the same as the way of the FP-Tree[14] pattern mining process. In a single prefix-path Item-Order-Tree that consists of only a single path starting from the root to the first branching node of the tree where a branching node containing more than one child. In the below example the two separate process. The single prefix-path P= ((a:10),(b:8),(c:7))can be mined by enumeration of all the combinations of the sub paths of P with the support set to the minimum support of the items contained in the sub path. This is because each sub path is distinct and occurs the same number of times as the minimum occurrence frequency among the items in the sub path. Thus the path P generate the following set of frequent patterns. So the frequent pattern set of P is {(a:10), (b:8), (c:7), (ab:8), (ac:7), (abc:7)}. Let Q be the second Item-Order-Tree, and it is rooted with null. The mining of Q can be using the same frequent pattern growth method.

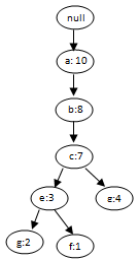

Fig 2: Single Prefix-path Tree

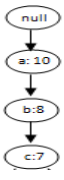

Fig 3: Single-path P

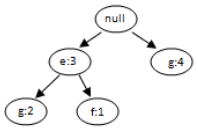

Fig 2: Multi-path Tree Q

Frequent Item Generation Algorithm

Input : An Item-Order-Tree T constructed according to Algorithm 1 and a minimum threshold 'mt'.

Output : The complete set of frequent patterns.

Method : Frequent_Item_Generation(Item-Order-Tree, null).

{
If Item-Order-Tree contains a single prefix path then
{
let P be the single prefix-path of Item-Order-Tree;
let Q be the multipath with the top branching node replaced by a null root;
for each combination(denoted β) of the nodes in the path P do
generate pattern β U α with support=minimum support of nodes in β;
let frequent_Item_set(P) be the set of patternsgenerated;}
else let Q be Item-Order-Tree;
for each item i in Q do{
generate pattern β= i U α with support= i.support;
construct β's conditional pattern-base and then β's conditional Item-Order-Tree$_\beta$;
if Item-Order-Tree$_\beta \neq$ □;
then call Freequent_Item_Generation(Item-Order-Tree$_\beta$);
let frequent_Item_set(Q) be the set of patterns so generated;}
return(frequent_Pattern_set(P) U frequent_Pattern_set(Q) U (frequent_Pattern_set(P)
X frequent_Pattern_set(Q)))}

V. CONCLUSION

Determining frequent objects is one of the most important fields in data mining. It is well known that the way candidates are defined have great effect on running time and memory need, and this is the reason for the large number of algorithms. It is also clear that the applied data structure also influences efficiency parameters. In this paper we presented an implementation that solved frequent itemset mining problem in a way by reducing the number of transactions which we saved as space in the Item-Order-Tree. Considering the continuity of a data stream, the general definition of finding frequent itemsets used in conventional data mining methodology may not be valid in a data stream. This is because the old information of a data stream may be no longer useful or possibly incorrect at present. In this algorithm we have proved the change in the item storage can improve the performance.

REFERENCES

[1] R. Agrawal, T. Imielienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. Proc. Conf. on

Management of Data, 207–216. ACM Press, New York, NY, USA 1993.

[2] H. Chang and W.S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *Proc.of the 9th ACM SIGKDD*, pp. 487 - 492, 2003.

[3] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pages 255 -264, Tucson, AZ, May1997.

[4] R. C. Agarwal, C. C. Aggarwal and V.V.V. Prasad. Depth first generation of long patterns. In *Proc. of the 6th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 108-118, Boston, MA, Sept. 2000.

[5] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of SIGMOD, 1993.

[6] M. Garofalakis, J. Gehrke and R. Rastogi. Querying and mining data streams: you only get one look. In *the tutorial notes of the 28th Int'l Conference on Very Large Databases*.

[7] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.

[8] M. Charikar, K. Chen and M. Farach-Colton. Finding frequent items in data streams. In *Proc. of the 29th Int'l Colloq. on Automata, Language and Programming*, 2002.

[9] C.-H. Lee, C.-R. Lin and M.-S. Chen. Sliding-window filtering: An efficient algorithm for incremental mining. In *Proc. of the 10th Int'l Conference on Information and Knowledge Management*, pages 263-270, Atlanta, GE, Nov. 2001.

[10] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris Online data mining for co-evolving time sequences. In *Proc. of the 16th Int'l Conference on Data Engineering*, pages 13-22, San Diego, CA, Feb. 2000.

[11] H. S. Javitz and A. Valdes. The NIDES statistical component description and justification. Annual report, March 1994.

[12] C. Hidber. Online association rule mining. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pages 145-156, Philadelphia, PA, May 1999.

[13] R. Agrawal, and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, Sept. 1994.

[14] Han, J., PEI, J., And YIN, Y.2000. Mining frequent patterns without candidate generation. In 2000 ACM SIGMOD Intl. Conference on Management of Data, W. Chen, J. Naughton, and P. A. Bernstein, Eds. ACM Press, 1-12.