Proceedings of GLOGIFT 07
November 15-17, 2007
UP Technical University
Noida, pp. 564-570

EFFICIENT ASSOCIATION RULE MINING FOR MARKET BASKET ANALYSIS

A. Shrivastava* and R. Sahu*

ABSTRACT

Data mining is an attitude that business actions should be based on learning, that informed decisions are better than uninformed decisions, and that measuring results is beneficial to the business. Data mining is also a process and a methodology for applying the tools and techniques. Association rule mining is also one among the most commonly used techniques in Data mining. A typical and the most running example of association rule mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets". The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customer and which items bring them better profits when placed with in close proximity.

The algorithms for single dimensional association rule mining, such as apriori and the FP-tree developed are in a greater use today. However, candidate set generation in apriori is still costly, especially when there exists a large number of patterns and/or long patterns. And both these algorithms prune the itemsets based on their frequencies (i.e., if their frequencies exceed minimum support threshold then they term them as frequent and the rest of them as infrequent). But this pruning technique is insufficient to help market analyst to make decisions such as planning the supermarket's shelf space, changing the layout new store layouts, new product assortments, which products to put on promotion so as to improve their marketing profits. So the focus of this thesis is to enhance these algorithms in a way that it provides frequent profitable patterns which help market analyst to make the best informed decisions for improving their business.

Keywords: Efficient Association, Rule Mining , Market Basket Analysis

Introduction

Database mining is motivated by decision support problem faced by most large retail organizations [4]. Progress in bar code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred to as basket data. A record in such data typically consists of the transaction date and items bought in the transaction. Successful organizations view such databases as important pieces of the marketing infrastructure. They are interested in instituting information-driven marketing processes, managed by database technology, which enables marketers to develop and implement customized marketing programs and strategies [6].

The problem of mining association rules over basket data was introduced in [3]. An example

* DGM, CMC Delhi

** Associate Professor, ABV - Indian Institute of Information Technology & Management, Gwalior

of such a rule might be that 98% of customers those who purchase tires and auto accessories also get automotive services done. Finding all such rules is valuable for cross-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. The databases involved in these applications are very large. It is imperative, therefore, to have fast and algorithms for this task.

The problem of finding association rules falls within the purview of database mining [5] [6][7], also called Knowledge Discovery in databases [8]. Related but not directly applicable, work includes the induction of classification rules. The other works in the machine learning literature is the KID3 algorithm presented in [9]. If used for finding associations this algorithm will make as many passes over the data as the number of combinations of items in the antecedent, which is exponentially large. Related work in the database literature is the work on inferring functional dependencies from data [10].

In this paper identifies the limitations of both apriori developed by Agrawal[1], and FP-tree developed by Han *et al*. (2000) [2] and then propose an algorithm, Frequent Profitable Pattern Tree algorithm, to identify frequent profitable patterns that help a market analyst to take informed decisions to improve business.

The remainder of this paper is organized as follows: section 2 presents the literature survey of single dimensional association rule mining algorithms, followed by the proposed FPP-Tree algorithm in section 3, followed by experimental results and comparison in section 4, and followed by conclusion and future work in section 5.

Literature Survey – Single Dimensional Association Rule Mining Algorithms

There are two main algorithms that are in common use: i) The Apriori Algorithm: mining frequent itemsets using candidate generation [1] and ii) The FP-Growth Algorithm: mining frequent itemsets without candidate generation. [2]

The Apriori Algorithm

The apriori-gen function takes as argument L_{k-1} the set of all large (k-1) itemsets. It returns a superset of the set of all large k-itemsets. The function works in two steps. These two steps are similar to the join and prune steps respectively. However, in general, first step produce a superset of the candidates produced by the join step. The major drawbacks of this approach are:

1. Generating large number of frequent itemsets is expensive: 10^6 frequent 1-itemsets require testing of 5*10^11 candidate 2-itemsets.
2. Not good for long patterns: A frequent itemset of size 100 requires testing of 2^{100} H" 10^{30} smaller candidate itemsets.
3. Repeated scans of database are expensive.
4. The main bottleneck is the candidate generation mechanism.

Mining Frequent Itemsets without Candidate Generation using FP-Tree

"Can we design a method that mines the complete set of frequent itemsets without candidate generation?" An interesting method in this attempt is called frequent pattern growth or simply FP-growth, which adopts a divide-and-conquer strategy as follows: compress the database representing frequent items into frequent-pattern tree or FP-tree, but retain the itemset association information, and then divide such a compressed database into a set of conditional databases (a special kind of projected database), each associated with frequent item, and

mine each such database separately.

The FP-tree construction takes exactly two scans of the transaction database: The first scan collects the set of frequent items, and the second scan constructs the FP-tree. The cost of inserting a transaction *Trans* into the FP-tree is $O(|freq(Trans)|)$, where *freq*(*Trans*) is the set of frequent items in *Trans*. We will show that the FP-tree contains the complete information for frequent-pattern mining.

Why FP-Tree is Better Over Apriori Algorithm?

Frequent pattern tree (FP-tree) is novel data structure, for storing compressed, crucial information about frequent patterns, and developed a pattern growth method, *FP-growth*, for efficient mining of frequent patterns in large databases. There are several advantages of *FP-growth* over other approaches:

1. It constructs a highly compact FP-tree, which is usually substantially smaller than the original database and thus saves the costly database scans in the subsequent mining processes.
2. It applies a pattern growth method which avoids costly candidate generation and test by successively concatenating frequent 1-itemset found in the (conditional) FP-trees. This ensures that it never generates any combinations of new candidate sets which are not in the database because the itemset in any transaction is always encoded in the corresponding path of the FP-trees. In this context, mining is not *Apriori*-like (*restricted*) *generation-and-test* but *frequent pattern* (*fragment*) *growth only*. The major operations of mining are count accumulation and prefix path count adjustment, which are much less costlier than candidate generation and pattern matching operations performed in most *Apriori*-like algorithms.
3. It applies a partitioning-based divide-and-conquer method which dramatically reduces the size of the subsequent conditional pattern bases and conditional FP-tree.

Common Limitations of Apriori and FP-Tree algorithms

Both of the above algorithms are really efficient in mining association rules. These algorithms prune the itemsets based on their frequencies (i.e., if their frequencies exceed minimum support they term them as frequent and the rest as infrequent). But this pruning based on frequencies, alone, cannot really help the market analyst to make decisions such as planning the supermarket's shelf space, changing the layout new store layouts, new product assortments, which products to put on promotion so as to improve their sales and marketing profits. For this we need to give the specific concern over profitable frequent patterns. These give a quick idea of which associations give them profit and which do not. And in some cases it may happen that, frequent patterns which are generated by the above algorithms may not be as profitable as a pruned-out pattern.

This may happen like, if a pattern a '! b '! c which is pruned-out as infrequent by both apriori and FP-tree, is giving a better profit than a frequent pattern such as b '! c '! f. And thus by taking into an account this result, the market analyst may commit mistake in arranging b, c, and f in one shelf (in close proximity) instead of arranging a, b, and c (which are actually profitable) into one.

Frequent Profitable Pattern Tree with Out Candidate Generation

This algorithm mines two databases, one is a transactions database D shown in Table 2, and the other is a profits database P which holds the profits of all the items as shown in the Table 1. The first scan of the database D is the same as Apriori, which derives the set of frequent

items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The items whose supports are greater than minimum support are selected as frequent. The set of frequent items is sorted in the order of descending support count. This resulting set or list is denoted by L. Thus, we have L=ác:4, f:4, a:3, b:3, m:3, p:3 ñ from the Table 3. A tree is constructed as follows. First create the root of the tree labeled with "null". Scan the database D second time. The items in each transaction are processed in L order (i.e., sorted according to the descending support count) and a branch is created for each transaction.

Table 1: The Profit Database

itm_name	Itm_profit
A	4
B	3
C	5
D	2
E	4
F	3
G	3
H	2
I	5
J	3
K	4
L	6
m	3
N	2
O	5
P	3
S	2

Table 2: The Transaction Database.

TID	item1	item2	item3	item4	item5	item6	item7	item8
1	f	a	c	d	g	i	m	p
2	a	b	c	f	l	m	o	
3	b	f	h	j	o	w		
4	b	c	k	s	p			
5	a	f	c	e	l	p	m	n

For example, the scan of the first transaction," TID: c, f, a, m, p" which contains five items in L order, leads to the construction of the first branch of the tree with five nodes: áác:4ñ áf:3ñ áa:3ñ ám:2ñ áp:2ññ, where c is linked as a child of the root, 'f' is linked to c, 'a' is linked to 'f', m is linked to 'a', and 'p' is linked to 'm'. However, this branch would share a common prefix ácñ, with the existing path for TID 100. Therefore, we instead increment the count of 'c' by 1, and create a new node (f: 3), which is linked as a child of (c:4). In general when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and the nodes of the prefix for the next match sequentially either till the leaf nose of until there are no matches, for each match increment the count value by 1. And still if there are any items in the transaction that are to be added into the tree, start inserting from the end node of the search. Now scan the profit database, which has the list of item-profits Páa:4, b:3,c:5,……ñ and assign the individual profit to each node in the tree correspondingly, then calculate the joint-profits of each node in the tree by using the formula,

Joint-profit of item X_i = count of Xi * (profit (Xi) + profit (parent (Xi) +

Σ profit (ancestors (Xi)).

For example for the branch ác, f, a, m, pñ,

Joint–profit of c = 4*(5+0) =20. (Since profit of root is 0.)

Joint–profit of f= 3*(3+5+0) =24.

The main concept behind doing this is if 'f' is purchased by customers for three times, then it also implies that 'c' is purchased by those customers for same number of times. So their joint-profit is calculated, since the profit acquired by selling f three times automatically includes the profit of selling 'c' for three times.

Similarly the joint profits of all nodes are calculated. The constructed tree is shown in the Figure 1. Now scan the tree for all the profitable-frequent patterns, whose joint-profit is more than the minimum joint profit (threshold given by the user). This gives the items which are profitable and also frequent for each prefix node (direct child to the root node). Now the market analyst can arrange the items in a more profitable way.

The FPP-Tree Algorithm design

Let $I = \{a_1, a_2, \ldots, a_m\}$ be a set of items, a *transaction database* $D = \langle T_1, T_2, \ldots, T_n \rangle$ where T_i ($i \in [1 \ldots n]$) is a transaction which contains a set of items in *I,* and $P = \langle p_1, p_2, p_3, \ldots \rangle$. The *support* (or occurrence frequency) of a *pattern A*, where *A* is a set of items, is the number of transactions containing *A* in *D*. A pattern *A* is *frequent* if *A*'s support is no less than a predefined *minimum support threshold*, ξ .

Given a transaction database *DB,* a profit database P which give the items profits, minimum joint profit threshold min-joint-profit *y*, and a minimum support threshold ξ , then we need to find *the complete set of profitable frequent patterns* using the algorithm given below.

Algorithm

***Input*:** A Transaction database D, Profit database P, minimum support threshold, minimum joint profit.

***Output*:** The complete set of profitable frequent patterns.

***Method*:**

The Profitable Frequent Pattern Tree is constructed in the following steps.

1. Scan the transaction database D once. Collect the set of frequent itemsets F, their corresponding supports.
2. Sort F in the descending order as L, the list of frequent items.
3. Create the root of a profitable-frequent-pattern tree, and label it as "null". For each transaction Trans in D do the following:
   a. Select and sort the frequent items in Trans according to the order of L.
   b. Start inserting the sorted frequent items to the tree. If the item is the first item in the transaction, then search the direct children (i.e., children at the first next level) of the root node for the item you are inserting.
   o If the item is found then, increment the count of that node by 1, and traverse the successive nodes of that node, searching for the next succeeding items in the Trans one by ne, until either no match is found or till the leaf node is reached. In this search, for each match increment the matched node count value by 1.And if there are any unmatched elements in the same transaction, start inserting from the node where the search is stopped. And while inserting increment the count value by 1.

- o If the item is not found then, insert the items one by one following the node-link right from the root node. And for every new node increment the value of count by one

4. After constructing the tree assign the profits to each node by accessing the profit database P accordingly.
5. Now calculate the joint-profit for each node by using the formula,

   Joint-profit of item X_i = count of Xi * (profit (Xi) + profit (parent (Xi) +

   Σ profit (ancestors (Xi)).
6. Scan and print out all the frequent items whose joint-profits are greater than min-joint-profit *y* for each prefix path.

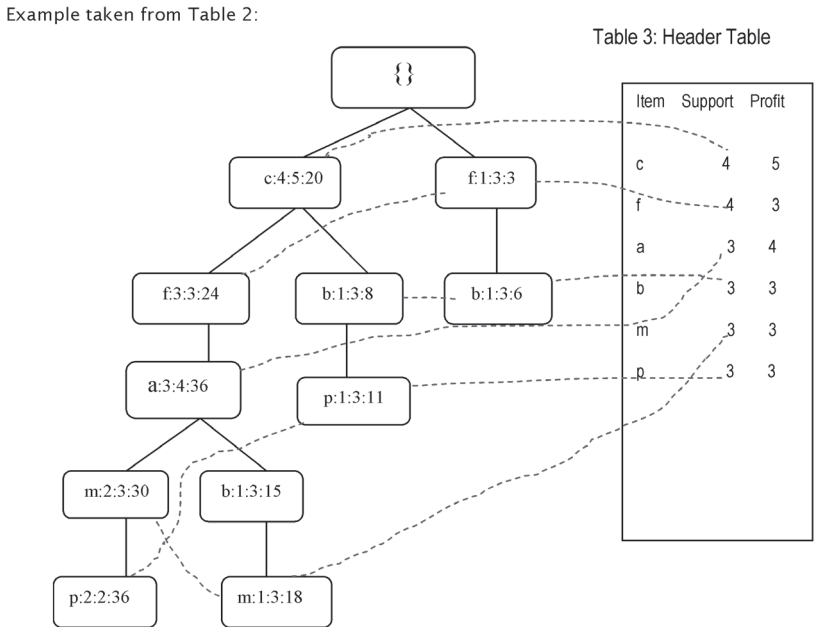

Figure 1: Frequent Profitable Pattern-Tree

Syntax of a node: *item-name: count: individual-profit: joint-profit*

The paper uses the database having the transactions table and item profits table, to predict the outcomes using the apriori, fp-tree and our proposed algorithm. It has been proved that our approach performs better results than the rest. It is observed that both apriori, and FP-tree algorithms gave only the frequent patterns as their outcomes (f, c, a), but the proposed algorithm gave frequent profitable patterns (f, c, a, m, p) as shown in the Table 4. The proposed algorithm says that m and p are giving profits which are comparatively same as f, c, and a. And from the obtained results, we may say that frequent items which are pruned out (i.e., like m, and p) while generating frequent patterns are profitable. Based on this idea our proposed algorithm gives profitable frequent patterns. So that we never prune out the profitable items in pruning stage of association rule mining while giving frequent profitable patterns as outcome. Note these values may change based on the minimum support threshold value, given by the user.

Table 4: Results from FTTP vis-à-vis Apriori and FP-Tree

Algorithm	Apriori	FP-Tree	FTTP
Results	*f, c, a*	*f, c, a*	*f, c, a, m, p*

Conclusion

One appeal of association rules is the clarity and utility of the results, which are in the form of rules about groups of products. There is an intuitive appeal to an association rule because it expresses how tangible products and services group together. A rule like, "*if a customer purchases three-way calling, then that customer will also purchase call waiting,*" is clear. Even better, it might suggest a specific course of action, such as bundling three-way calling with call waiting into a single service package. Keeping this in mind our proposed algorithm has given profitable frequent patterns with out generating any candidate itemsets. The proposed algorithm efficiently makes use of the knowledge acquired from both the "apriori" developed by Agrawal and Srikanth [1] and "FP-tree" in Han et.al (2000)[2].

The Frequent Profitable Pattern Tree constructed in the section 3 gives the *frequent profitable patterns* which help the market analyst to arrange the shelf space in a more profitable way, which improves their business. And since this FPP-tree is also being constructed without candidate generation, it inherits all the advantages of FP-tree, such as: i. it constructs a highly compact FPP-tree, which is usually substantially smaller than the original database and thus saves the costly database scans in the subsequent mining processes, and ii. it doesn't generate any candidate itemsets, so the calculation and tree building process speeds up.

References

1. Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94),* Santiago, Chile, pp. 487–499.
2. Han, J., Pei, J., and Yin, Y. 2000. Mining frequent patterns without candidate generation. *In Proc. 2000 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'00*), Dallas, TX, pp. 1–12.
3. Rakesh Agrawal, Tomasz Imielinski, and Arun swami. Mining Association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD conference on Management of data, pages 207-216, Washington, D.C., May 1993*
4. Rakesh Agrawal, Christos Faloutsos, and Arun swami. Efficient similarity search in sequence Databases. In *Proc of the Fourth International Conference on Foundations of Data Organization and Algorithms,Chicago,October 1993.*
5. Rakesh Agrawal, Tomasz Imielinski, and Arun swami. Database Mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering,* 5(6):914-925, December 1993.*Special Issue on Learning and Discovery in Knowledge Based Databases.*
6. R.S.Michalski, L.Kerschberg, K.A.Kaufman, and J.S.Ribeiro. Mining for knowledge in databases: The INLEN architecture, initial implementation, and first results. *Journal of Intelligent Information* Systems,1:85-113,1992.
7. Jiawei Han, Yandong Cai, and Nick Cercone. Knowledge discovery in databases: An attribute oriented approach. *In Proc. of the VLDB* Conference, pages 547-559, Van-couver, British Columbia, Canada,1992.
8. David J.Lubinsky . Discovery from databases: A review of AI and statistical techniques. *In IJCAI- 89 workshop on knowledge discovery in databases*, pages 204-218, Detroit, August 1989.
9. Heikki Mannila and Kari-Jouku Raiha. Dependecy inference. In Proc. of the VLDB Conference, pages 155-158,Brighton, England,1987.
10. Heikki Mannila , Hannu Toivonen, and A.Inkeriverkamo. Efficient algorithms for discovering association rules. In KDD-94: *AAAI Workshop on Knowledge Discovery in Databases*, July 1994.