ISSN 2229-5518

Efficient Association Rule Mining Using Improved Apriori Algorithm

Ish Nath Jha, Samarjeet Borah

Abstract— Association rule mining is a data mining technique to extract interesting relationships from large datasets [1, 2]. The efficiency of association rule mining algorithms has been a challenging research area in the domain of data mining [3]. Frequent pattern discovery, the task of finding sets of items that frequently occur together in a dataset is the most resource consuming phase of the rule mining process [4, 5]. Efforts for improvement, in the basic mining techniques are continuously being made. In this paper we take the classical algorithm APRIORI and optimize its performance by applying classification and sorting on the datasets.

Index Terms— Association Rule Mining, Data mining, Apriori Algorithm, Frequent Pattern Discovery, Efficiency.

1 Introduction

Association rule mining is a widely used technique for knowledge discovery from large data warehouses. Rule mining algorithms find all interesting relationships in large datasets by finding those itemsets that often co-occur preferably indicating with what frequency. These frequent itemsets are then used to generate association rules. The task of finding frequent itemsets is most resource consuming phase. The complexity has only increased with the growing size of datasets. The idea of association rules was popularized by the article published in 1993 by Rakesh Aggarwal. Since then, association rule mining techniques have been at the core of research in the area of Data Mining. Rule mining techniques were initially applied for the popular market basket analysis but now find applications in the areas of bioinformatics, geoinformatics, intrusion detection, web usage mining etc. The various areas of challenges in association rule mining are the interestingness of rules discovered, mining rules from incremental database, scalability, memory efficiency, time complexity, etc. Various algorithms have been proposed each with its own merits and demerits over the others. APRIORI, proposed by R. Aggarwal however remains the most popular algorithm and all other algorithms exploit the basic underlying concept of APRIORI. APRIORI has an aggressive search space pruning strategy. Yet the support counting phase has a heuristic approach. It is due to the nature of representation of items and transactions in the database.

In this paper the classical APRIORI has been chosen for the experiment and a unique sorting technique along with clustering is used to arrange the datasets which drastically reduces the number of comparisons made against the datasets to discover the rules. To do so the items and the transactions are assigned numeric attributes which can be used to simply skip some transaction during support counting phase.

- *Ish Nath Jha is currently pursuing M. Tech.in Computer Science & Engineering from Sikkim Manipal University, Sikkim, India-737136. E-mail: ish.jha@gmail.com*
- *Samarjeet Borah is currently working as Associate Professor in the Department of Computer Science & Engineering at Sikkim Manipal Institute of Technology, a Contituent College of Sikkim Manipal University, Sikkim, India-737136. E-mail: samarjeetborah@gmail.com*

2 Background

2.1 Association Rule Description

An association rule can be explained as follows: Let $I = \{i_1, i_2, i_3, \dots, i_n\}$ be a set of n different items, DB $= \{T_1, T_2, T_3, \dots, T_m\}$ be the transaction database consisting of m transactions, where each transaction $T_i = \{i_1, i_2, i_3, \dots, i_k\}$ is a set of k elements from I. Thus $T_i \subseteq I$. An association rule is then specified as $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \Phi$. All such rules have two attributes associated with them, i.e. support and confidence. Let S be the percentage of transactions in DB which contain $X \cup Y$ then S is known as the support of $X \Rightarrow Y$. Let C be the percentage of transactions in DB containing X which also contain Y then the rule $X \Rightarrow Y$, holds with confidence C. Any statement of the form $X \Rightarrow Y$, is a rule if and only if the support of X and Y is greater than or equal to a user specified threshold value known as minimum support as well as the ratio of support $(X \cup Y)$ /support (X) is greater than or equal to user specified minimum confidence. Given any rule, $X \Rightarrow Y$, X is known as antecedent and Y is known as consequent.

2.2 The Classical APRIORI Algorithm

The classical APRIORI algorithm generates association rules in two steps:

i. By scanning the database iteratively to find the support count of each K-itemset where $K = 0,1, \dots p$, such that $p \leq$ maximum number of items in the database. All those itemsets whose support count is greater than or equal to the user specified minimum support is known as a frequent itemset. This phase is most resource consuming.

ii. Generate association rules from the frequent itemsets. For every frequent itemset X, if $\subset X$, $Y \neq \Phi$, and support (X) / support $(Y) \geq$ minimum confidence, then $Y \Rightarrow (X - Y)$.

C_1={$Candidate\ 1-itemsets$};
L_1={$c \in C_1 | c.count \geq minimum\ support$ };
$FOR\ (K = 2;\ L_k - 1 \neq \Phi : K++)\ DO\ BEGIN$
$C_k = apriori - gen(L_k - 1)$;
FOR all transaction $T_i \in$ DB $DO\ BEGIN$

$C_t = subset(C_k, t);$
FOR all candidates $c \in C_t$ DO
$c.count++;$
END
$L_k = \{c \in C_k \mid c.count \geq minsup\}$
END

$ANSWER = \cup L_k;$

The above stated algorithm can be explained as follows:

At first all the frequent 1-itemsets are found by simply counting the support of each individual item in the transaction database. This set is denoted by L_1. L_1 is used to find L_2, the set of all frequent 2-itemsets. This cycle continues until no more frequent k-itemsets are found. At this stage the first step of APRIORI algorithm stops. During every Kth cycle a set of candidate K-itemsets, denoted by C_k is generated at first. Each itemset in C_k is generated by joining two frequent itemsets from L_{k-1} which have only one different item. The itemsets in C_k are candidates for frequent K-itemset in L_k. Thus L_k is always a subset of C_k. The set C_k is pruned to retain those elements whose support count should be verified by scanning the database. Pruning is an efficient method of removing all those elements of C_k which can be declared a non frequent itemset without scanning the DB. Pruning removes all those itemsets of C_k whose any of the subset is not an element of L_{k-1}. This is done on the basis that if some superset is frequent then all its subset must be frequent as well.

The current research trend focuses on developing efficient algorithms for generating the set of all frequent itemsets. In the following section we discuss the problem of the existing algorithm and the proposed improvement in the basic APRIORI algorithm.

2.3 Related Work

Different optimization methods for association rule mining have been proposed. The process is too resource-consuming, especially when there is not enough available physical memory for the whole database. A solution to encounter this problem is to use evolutionary algorithms, which reduce both cost and time of rule discovery. Yan proposed a method based on genetic algorithm without considering minimum support [7]. The method uses an extension of elaborate encoding while relative confidence is the fitness function. A public search is performed based on genetic algorithm. As the method does not use minimum support, a system automation procedure is used instead. It can be extended for quantitative-valued association rule mining. In order to improve algorithm's efficiency, it uses a generalized FP-tree. Evaluation of the algorithm shows a considerable reduction in computational cost. Kaya proposed genetic clustering method in [8]. Hong proposed a cluster based method for mining generalized fuzzy association rules [9]. Chen proposed a cluster-based fuzzy-genetic mining method for association rules and membership functions [10]. In many of these works clustering has been used to gain speedup and improve the efficiency of the algorithm. In the proposed work the idea of clustering is used without any evolutionary algorithm.

2.4 Recent Advances

Since the publication of APRIORI many subsequent ideas have been proposed and FP-Growth being one of them has gained a lot of popularity. In the following section some literature survey is presented to explain why APRIORI was chosen over FP-Growth for this work. Han introduces a quite novel algorithm to solve the frequent itemset mining problem in [4]. They adapt the idea of a trie to the set of transactions rather than candidates. In so doing, they effectively compress the dataset D with the hope that it will fit entirely in main memory. The data structure appears to eliminate the construction of candidates entirely. Experimental results have demonstrated consistently that it significantly outperforms A Priori. However, once the trie no longer fits in memory it suffers exactly the same consequences as in [5]. Even building the trie becomes extremely costly, to the point that in [6] it is remarked that the dominant percentage of execution time is that of constructing the trie. Consequently, on truly large datasets, the FPGrowth algorithm fails even to initialize.

When first introduced, it was remarked that the algorithm scales quite elegantly. Indeed, if one has already constructed a trie, then the cost of mining it is roughly the same independent of the support threshold (except that the recursion produces more intermediate trees). However, one must be careful here. FPGrowth has a preprocessing step that prunes out all infrequent 1-itemsets prior to building the trie. Consequently, it does not scale as claimed because as the support threshold is lowered, the number of items pruned from the dataset decreases—and each of these newly unpruned items needs appear in the trie. So the trie needs to be reconstructed and it grows. How much it grows is dependent on the distribution of the dataset and the amount by which the support threshold is reduced. This growth can be several orders of magnitude for relatively small decreases in support threshold.

Furthermore, despite the claim that FPGrowth does not produce any candidates, Goethals demonstrates in [11] that it can, in fact, be considered a candidate-based algorithm and Dexters later show that the probability of any particular candidate being generated is actually higher in FPGrowth than in the classical A Priori algorithm [3].

Another general problem with the FPGrowth algorithm is that it lacks the incremental behavior of A Priori, something that builds fault tolerance into the algorithm. Should a machine running A Priori fail or shut down after producing, say, its frequent 5-itemsets, the algorithm can be easily restarted from that point by beginning with the construction of candidate 6-itemsets, rather than starting from the beginning. However, because FPGrowth operates by means of recursion, there are very few points at which the program can save state in anticipation of failure.

3 Improved Algorithm

Problem Description: In order to confirm whether an itemset of C_k after pruning is frequent or not its support is counted by scanning the entire transaction database. This is a heuristic approach. Such a heuristic approach is a necessity because of

the representation of the transactions in the DB. Moreover this task has to be repeated for all the remaining itemsets of C_k after pruning during each K-th cycle.

Basic Principle of improvement: We propose that each of the element of the set $I = \{i_1, i_2, i_3, \dots, i_n\}$ be assigned a unique integer value in increasing order of enumeration so that each transaction $T_i = \{i_1, i_2, i_3, \dots, i_k\}$ can be given a value, say total_sum which would be equal to the sum total of values of each individual element contained in T_i. At the same time during the 1st cycle while counting the support of 1-itemsets we increase the number of items represented by item_count of each transaction by one if an item is present in that transaction. Thus by the end of 1st cycle we will have two attributes associated with each transaction namely total_sum representing a numeric value associated with each transaction and item_count representing the number of items contained by that transaction. Now all these transactions which have got same item_count are grouped into one. So we get n-groups of transactions where $n \geq 1$. Further the transactions within a particular group are sorted on the basis of their total_sum value. The basic idea behind these computations is that any k-itemset can be found in some transactions whose size is minimum k. So while counting the support for any k-itemset we will look into only those transactions belonging to some group whose item_count is greater than or equal to the k and other groups are ignored. The advantage of keeping the transactions within a group in sorted order is that when we look for the presence of an itemset within any group of transactions we can ignore all those transactions whose total_sum value is less than the sum total value of individual elements of the itemset. The logic behind this is that we can also associate total_sum as an attribute of any k-itemset of k items which will be equal to the sum total of the numeric value of each individual item of that set. And if some transaction of size m, where $m \geq k$ contains this k-itemset than the total_sum value of that transaction will be greater than or equal to the total_sum value of the k-itemset.

These computations performed on the dataset will incur cost in terms of execution time however it is needed only once during the 1st cycle and for all the subsequent cycle of support counting its benefit is evident and can be summarized as follows.

Let TS_i represent the total_sum value of transaction T_i , C_i be the total_sum value of some itemset and N be the size of maximum sized transaction then the total number of comparisons for finding the support count of that itemset is:

$$\sum_{i=size(itemset)}^{N}[size(Group(i))\text{-}\{Count(T_i): TS_i < C_i)\}] \quad (1)$$

4 Implementation and Results

APRIORI and the proposed modification both have been implemented and then tested using a randomly generated synthetic dataset. To generate a sample dataset, we have filled a market basket dataset such a way that the value of each field in each transaction is randomly generated. So, each field has a 50% chance to be true (i.e. a 50% chance to occur in each transaction).

To evaluate the support count of an itemset, the groups having itemsets of size less than the number of items in the itemset are ignored. The ratio of ignored transactions to total number of transactions depends on the size of the itemset. This ratio plays an important role in the performance (execution time) of the algorithm. The higher the ratio better is the performance. Further some transactions within a particular group are ignored if their total_sum value is less than that of the itemset which adds to the performance of the algorithm. It is verified with repeated experiments that for smaller support threshold the new algorithm performs better. It is because the probability of itemsets with larger number of items being frequent increases and so the number of comparison increases for the original APRIORI whereas in the modified APRIORI with increased size of itemsets the chances of ignored groups of itemsets also increases.

Fig. 1 shows a comparison between the numbers of comparison needed by APRIORI and modified APRIORI algorithm for different number of transactions in the randomly generated dataset. Fig. 2 compares the execution time of APRIORI and modified APRIORI algorithm for different number of transactions in the randomly generated dataset. Fig. 3 compares the execution time for both the algorithms on a dataset with 120000 transactions with different support threshold values.

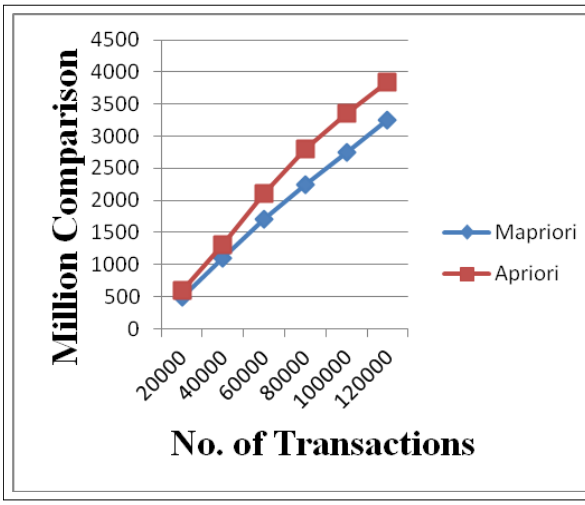

Fig. 1. Total Comparison with Different Number of Transaction in the Randomly Generated Dataset

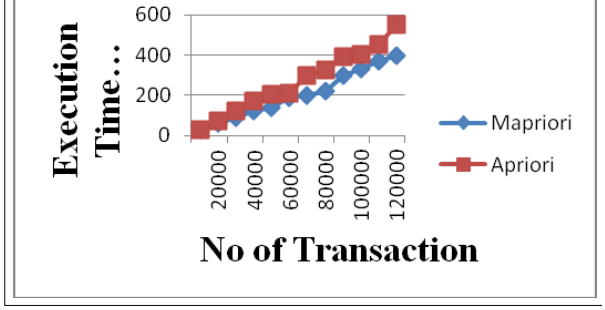

Fig. 2. Execution Time with Different Number of Transaction in the Randomly Generated Dataset

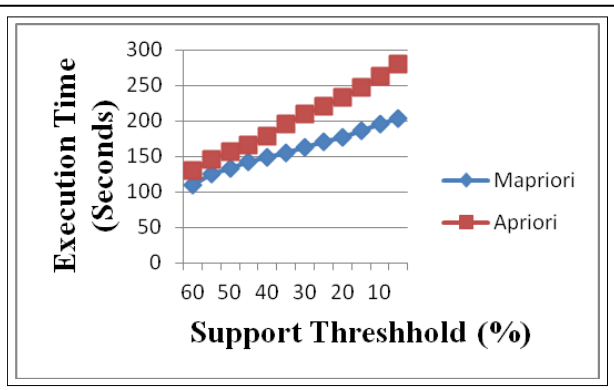

Fig. 3. Execution Time with Different Threshold Values 120000 Transactions from the Randomly Generated Dataset

4 Conclusion

In this work the basic APRIORI has been optimized for discovering association rules from large data warehouses. The proposed method is to cluster transactions with equal number of items into one and further sort the transactions within each cluster. This approach leads to avoiding of some dispensable comparisons against the database. It is due to the fact that certain itemsets are impossible to occur in many clusters and even in many transactions. We obtained a speedup of 12.9% and a decrease of 11% in the number of comparison. This work can further be extended to obtain negative association rules, and some computational method can be created to avoid making comparisons at all for generating rules from the data warehouses.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993. ACM Press, 1993, pp. 207–216.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in Proc. of VLDB, 1994, pp. 487–499.

[3] N. Dexters, P. W. Purdom, and D. Van Gucht, "A probability analysis for candidate-based frequent itemset algorithms," in SAC '06. New York, NY, USA: ACM, 2006, pp. 541–545.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in SIGMOD Conference. ACM, 2000, pp. 1–12.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," SIGMOD Rec., Volume. 26, no. 2, pp. 255–264, 1997.

[6] G. Buehrer, S. Parthasarathy, and A. Ghoting, "Out-of-core frequent pattern mining on a commodity pc," in KDD '06: Proceedings of the 12th ACM SIGKDD international confer- ence on Knowledge discovery and data mining. New York, NY, USA: ACM, 2006, pp. 86–95.

[7] X. Yan, "Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support", Expert Systems with Applications, Volume 36, Issue 2, pp: 3066-3076 (2008).

[8] M. Kaya, R. Alhajj, "Genetic algorithm based framework for mining fuzzy association rules", Fuzzy Sets and Systems 152:3, pp 587-601 (2005).

[9] B. C. Chien, Z. L. Lin and T. P. Hong, "An efficient clustering algorithm for mining fuzzy quantitative association rules", The Ninth International Fuzzy Systems Association World Congress, pp. 1306-1311 (2001).

[10] C.H. Chen, V.S. Tseng, T.P. Hong, "Cluster-Based Evaluation in Fuzzy-Genetic Data Mining", IEEE T. Fuzzy Systems 16(1): 249-262 (2008).

[11] B. Goethals, "Efficient frequent pattern mining," Ph.D. dis-sertation, transnationale Universiteit Limburg, 2002.

[12] Sean Chester, Ian Sandler, Alex Thomo: Scalable APRIORIBased Frequent Pattern Discovery. CSE (1) 2009: 48-55