An Analysis on Association Rule Mining Techniques

Ish Nath Jha
Department of Computer Science & Engineering,
Sikkim Manipal Institute of Technology
Majitar, East Sikkim, India, PIN-737136

Samarjeet Borah
Department of Computer Science & Engineering,
Sikkim Manipal Institute of Technology
Majitar, East Sikkim, India, PIN-737136

ABSTRACT
Association rule mining is a subfield of Data mining. It is a popular and widely used method to extract interesting and useful patterns from large sets of data. The first Rule Mining Algorithm was formulated by R. Agrawal in 1993. After the Apriori Algorithm formulated by R. Agrawal, many other algorithms have been proposed. Each of these algorithms has its own advantages and disadvantages over the others. The major issues of concern are the cost efficiency in terms of memory utilization, interestingness of the rules generated, influence of the minimum support level specified on the rules generated, the ability to discover relationships not only quantitatively but also qualitatively and the processing efficiency of the algorithm. This paper provides a comparative analysis on the classical Apriori algorithm along with some other association rule mining algorithms.

Keywords
Association Rule Mining, Support, Confidence, Apriori, AIS, FP-Tree

1. INTRODUCTION
Association rule mining, one of the most important and well researched techniques of data mining, was first introduced in [1][2]. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. 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.

1.1 Association Rule Description
An association rule can be explained as follows: Let $I = \{i_1, i_2, i_3 \ldots i$ be a set of different items, $DB = \{T_1, T_2, T_3 \ldots T,$ be the transaction database (DB) consisting of transactions, where each transaction $T_i = \{i_1, i_2, i_3 \ldots i,$ is a set of elements from . Thus $T_i \subseteq$. An association rule is then specified as $X \Rightarrow$ where $X \subseteq I,\ Y \subseteq$ and $X \cap Y =$. All such rules have two attribute associated with them, i.e. support and confidence. Let be the percentage of transactions in DB which contain $X \cup$ then is known as the support of $X \Rightarrow$. Let be the percentage of transactions in DB containing which also contain then the rule $X \Rightarrow$ holds with confidence . Any statement of the form $X \Rightarrow$ is a rule if and only if the support of and 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 (is greater than or equal to user specified minimum confidence. Given any rule, $X \Rightarrow$, is known as antecedent and is known as consequent.

1.1.1 Support (s)
Support(s) of an association rule is defined as the percentage/fraction of records that contain $X \cup Y$ to the total number of records in the database. The count for each item is increased by one every time the item is encountered in different transaction T in database D during the scanning process. It means the support count does not take the quantity of the item into account. For example in a transaction a customer buys three bottles of beers but we only increase the support count number of beers by one, in another word if a transaction contains a item then the support count of this item is increased by one. Support(s) is calculated by the following formula:

$$\text{Support}(X,Y) = \frac{\text{Support Count of } XY}{\text{Total Number of Transactions in D}}$$

We can see, support of an item is a statistical significance of an association rule. Suppose the support of an item is 0.1%, it means only 0.1 percent of the transaction contain purchasing of this item. The retailer will not pay much attention to such kind of items that are not bought so frequently obviously a high support is desired for more interesting association rules. Before the mining process, users can specify the minimum support as a threshold, which means they are only interested in certain association rules that are generated from those itemsets whose supports exceed that threshold. However, sometimes even the itemsets are not as frequent as defined by the threshold, the association rules generated from them are still important.

1.1.2 Confidence
Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain X Y to the total number of records that contain X, where if the percentage exceeds the threshold of confidence an interesting association rule X→Y can be generated.

$$\text{Confidence } (X|Y) = \frac{\text{Support of } (XY)}{\text{Support of } X}$$

Confidence is a measure of strength of the association rules, suppose the confidence of the association rule X→Y is 80%, it means that 80% of the transactions that contain X also contain Y together, similarly to ensure the interestingness of the rules specified minimum confidence is also pre-defined by users.
The current research trend focuses on developing efficient algorithms for generating the set of all frequent itemsets. The following section contains some popular association rule mining approaches.

2. ASSOCIATION RULE MINING APPROACHES

Association rule mining is a well explored research area. In this section some basic and classic approaches for association rule mining will be explored. As stated before, the second subproblem of ARM is straightforward; most of those approaches focus on the first subproblem. The first subproblem can be further divided into two subproblems: candidate large itemsets generation process and frequent itemsets generation process [3]. We call those itemsets whose support exceed the support threshold as large or frequent itemsets, those itemsets that are expected or have the hope to be large or frequent are called candidate itemsets. Most of the algorithms of mining association rules we surveyed are quite similar, the difference is the extent to which certain improvements have been made, so only some of the milestones of association rule mining algorithms are introduced in the following section.

2.1 AIS Algorithm

The AIS (Agrawal, Imielinski, Swami) algorithm was the first algorithm proposed for mining association rule in [Agrawal et. al. 1993]. It focuses on improving the quality of databases together with necessary functionality to process decision support queries. In this algorithm only one item consequent association rules are generated, which means that the consequent of those rules only contain one item, for example we only generate rules like $X \cap Y \Rightarrow Z$ but not those rules as $X \Rightarrow Y \cap Z$. The databases were scanned many times to get the frequent itemsets in AIS. During the first pass over the database, the support count of each individual item was accumulated as shown in Table I (b), suppose the minimal support threshold is 30%, large one items were generated in Table I(c). According to minimal support those items whose support counts are less than 3 (I4 and I6) are eliminated from the list of frequent items. With those frequent 1 items, candidate 2 -itemsets are generated by extending those frequent items with other items in the same transaction. To avoid generating the same itemsets repeatedly the items were ordered, candidate itemsets are generated by joining the large items in previous pass and another item in the transaction, which appears later than the last item in the frequent itemsets. For example, based on transaction T100 I_1, I_2, I_5, according to this specific order we generate candidate 2 -itemsets by extending I1 with only I_2, I_5, similarly I_2 is extended with I_5. The result is shown in Table I (d). During the second pass over the database, the support count of those candidate 2-itemsets are accumulated and checked against the support threshold. Similarly those candidate (k+1)-itemsets are generated by extending frequent k-itemsets with items in the same transaction. All those candidate itemsets generation and frequent itemsets generation process iterate until any one of them becomes empty. The result frequent itemsets includes only one large 3-itemsets $\{I_1, I_2, I_5\}$. To make this algorithm more efficient, an estimation method was introduced to prune those itemsets candidates that have no hope to be large, consequently the unnecessary effort of counting those itemsets can be avoided. Since all the candidate itemsets and frequent itemsets are assumed to be stored in the main memory, memory management is also proposed for AIS when memory is not enough. One approach is to delete candidate itemsets that have never been extended. Another approach is to delete candidate itemsets that have maximal number of items and their siblings, and store this, the parent itemsets in the disk as a seed for the next pass. The detail examples are available in [Agrawal et al. 1993]. The main drawback of the AIS algorithm is too many candidate itemsets that finally turned out to be small are generated, which requires more space and wastes much effort that turned out to be useless. At the same time this algorithm requires too many passes over the whole database.

Table 1 Example of AIS Algorithm

TID	List of items
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3
T000	I1,I2,I5,I6

a) Original Dataset

Items	Count Number
I1	7
I2	8
I3	6
I4	2
I5	3
I6	1

b) C1

Large 1 Items
I1
I2
I3
I5

c) L1

Items	Count Number
I1,I2	5
I1,I5	3
I2,I5	3
I2,I4	2
I2,I3	4
I1,I4	1
….	….

d) C2

Large 2 items
I1,I2
I1,I5
I2,I5
I2,I3
I1,I3

e) L2

Items	Count Number
I1,I2,I5	3
I1,I2,I4	1
I1,I2,I3	2
I1,I2,I6	1
I2,I3,I5	1
I1,I3,I5	1
….	….

f) C3

2.2 Apriori Algorithm

Apriori is a great improvement in the history of association rule mining, Apriori algorithm was first proposed by Agrawal in [Agrawal and Srikant 1994]. The AIS is just a straightforward approach that requires many passes over the database, generating many candidate itemsets and storing counters of each candidate while most of them turn out to be not frequent. Apriori is more efficient during the candidate generation process for two reasons; Apriori employs a different candidate generation method and a new pruning technique.

There are two processes to find out all the large itemsets from the database in Apriori algorithm. First the candidate itemsets are generated, and then the database is scanned to check the actual support count of the corresponding itemsets. During the first scanning of the database the support count of each item is calculated and the large 1 -itemsets are generated by pruning those itemsets whose supports are below the pre-defined threshold as shown in Table II(a) and (b). In each pass only those candidate itemsets that include the same specified number of items are generated and checked. The candidate k-itemsets are generated after the $(k-1)^{th}$ passes over the database by joining the frequent k-1 -itemsets. All the candidate k-itemsets are pruned by check their sub (k-1)-itemsets, if any of its sub (k-1)-itemsets is not in the list of frequent (k-1)-itemsets, this k-itemsets candidate is pruned out because it has no hope to be frequent according the Apriori property. The Apriori property says that every sub (k-1)-itemsets of the frequent k-itemsets must be frequent. Let us

take the generation of candidate 3-itemsets as an example. First all the candidate itemsets are generated by joining frequent 2-itemsets, which include (I_1, I_2, I_5), (I_1, I_2, I_3), (I_2, I_3, I_5), (I_1, I_3, I_5). Those itemsets are then checked for their sub itemsets, since (I_3, I_5) is not frequent 2-itemsets, the last two 3-itemsets are eliminated from the list of candidate 3-itemsets as shown in Table II(e). All those processes are executed iteratively to find all frequent itemsets until the candidate itemsets or the frequent itemsets become empty. The result is the same as the AIS algorithm.

2.2.1 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 $p \leq maximum\ no.of\ items\ in\ the\ database$. All those itemsets whose support count is greater than or equal to 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 $Y \Rightarrow (X - Y)$.

$C1$={$Candidate\ 1-itemsets$};
$L1$={$c \in C1 | c.count \geq minimum\ support$};
$FOR\ (K = 2;\ L_k - 1 \neq \Phi$: 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
$L_K = \{c \in C_k | c.count \geq minsup\}$
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 K^{th} 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 DB. 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.

In the process of finding frequent itemsets, Apriori avoids the effort wastage of counting the candidate itemsets that are known to be infrequent. The candidates are generated by joining among the frequent itemsets level-wisely, also candidate are pruned according the Apriori property. As a result the number of remaining candidate itemsets ready for further support checking becomes much smaller, which dramatically reduces the computation, I/O cost and memory requirement. Table II shows the process of Apriori algorithm. By comparing Table I and Table II we can see the numbers of candidates changed dramatically.

Apriori algorithm still inherits the drawback of scanning the whole data bases many times. Based on Apriori algorithm, many new algorithms were designed with some modifications or improvements. Generally there were two approaches: one is to reduce the number of passes over the whole database or replacing the whole database with only part of it based on the current frequent itemsets, another approach is to explore different kinds of pruning techniques to make the number of candidate itemsets much smaller. Apriori-TID and Apriori-Hybrid [Agrawal and Srikant 1994] , DHP [Park et al. 1995], SON [Savesere et al. 1995] are modifications of the Apriori algorithm.

Most of the algorithms mentioned above are based on the Apriori algorithm and try to improve the efficiency by making some modifications, such as reducing the number of passes over the database; reducing the size of the database to be scanned in every pass; pruning the candidates by different techniques and using sampling technique. However there are two bottlenecks of the Apriori algorithm. One is the complex candidate generation process that uses most of the time, space and memory. Another bottleneck is the multiple scan of the database.

Table 2 Example of Apriori Mining Process

Items	Count Number
I1	7
I2	8
I3	6
I4	2
I5	3
I6	1

a) C1

Large 1 Items
I1
I2
I3
I5

b) L1

Items	Count Number
I1,I2	5
I1,I3	4
I1,I5	3
I2,I3	4
I2,I5	3
I3,I5	1

c) C2

Large 2 Items
I1,I2
I1,I5
I2,I5
I2,I3
I1,I3

d) L2

Items	Count Number
I1,I2,I5	3
I1,I2,I3	2

e) C3

2.2.2 FP-Tree (Frequent Pattern Tree) Algorithm

To break the two bottlenecks of Apriori series algorithms, some works of association rule mining using tree structure have been designed. FP-Tree [Han et al. 2000], frequent pattern mining, is another milestone in the development of association rule mining, which breaks the two bottlenecks of the Apriori algorithm. The frequent itemsets are generated with only two passes over the database and without any candidate generation process. FP-Tree was introduced by Han et al in [4]. By avoiding the candidate generation process and less passes over the database, FP-Tree is an order of magnitude faster than the Apriori algorithm. The frequent

patterns generation process includes two sub processes: constructing the FP-Tree, and generating frequent patterns from the FP-Tree.

The process of constructing the FP-Tree is as follows:

- The database (Table I (a)) is scanned for the first time. During this scanning the support count of each items are collected. As a result the frequent *1* -itemsets are generated as shown in Table III(a), this process is the same as in Apriori algorithm. Those frequent itemsets are sorted in a descending order of their supports. Also the head table of ordered frequent *1* -itemsets is created as shown in Figure 1.
- Create the root node of the FP-Tree T with a label of Root. The database is scanned again to construct the FP-Tree with the head table, for each transaction the order of frequent items is resorted according to the head table. For example, the first transaction (I_1, I_2, I_5) is transformed to (I_2, I_1, I_5), since I_2 occurs more frequently than I_1 in the database. Let the items in the transaction be [p| P], where p is the first frequent item and P is the remaining items list, and call the function Insert {[p |P]; T}.

The function *Insert* {[p | P]; T} works as follows. If T has a child N such that N.item-name=p.item-name then the count of N is increased by 1, else a new node N is created and N.item-name=p.item-name with a support count of 1. Its parent link be linked to T and its node link is linked to the node with the same item-name via a sub-link. This function *Insert* {P;T} is called recursively until P becomes empty.

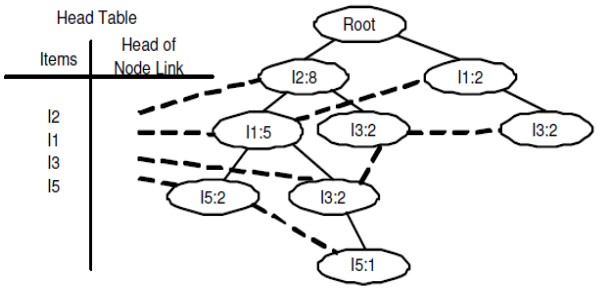

Fig. 1 Result of FP-Tree

Let's take the insertion of first transaction to the FP-Tree as an example to illustrate the *insert* function and construction of FP-Tree we mentioned above. After reorder this transaction is (I_2, I_1, I_5), so p is I_2 in this case, while P is (I_1, I_5). Then we call the function of *insert*, first we search and determine the node I_2 exists in the tree or not, it turns out I_2 is a new node. According to the rules, a new node named I_2 is created with a support count of 1. Since here T is *Root*, node I_2 is linked to Root and call the insert function again. At this time p is I_1, P is I_5, T is I_2. The result of the FP-Tree of the database is shown in Figure 1.

Table 3 Example of FP-Tree Algorithm

Large 1 Items	Support
I1	7
I2	8
I3	6
I5	3

TID	Ordered Large Items
T100	I2,I1,I5
T200	I2
T300	I2,I3
T400	I2,I1
T500	I1,I3
T600	I2,I3
……	……

a) L1 **b) Transformed Data**

The frequent patterns are generated from the FP-Tree by the procedure named FP-growth [Han and Pei 2000]. Based on the head table and the FP-Tree, frequent patterns can be generated easily. It works as follows:

Input: A transactional database *DB* and a minimum support threshold ξ.
Output: Its frequent pattern tree, FP-tree
Method: The FP-tree is constructed in the following steps:

1. Scan the transaction database *DB* once. Collect the set of frequent items *F* and their supports. Sort *F* in support descending order as *L*, the *list* of frequent items.
2. Create the root of an FP-tree, *T*, and label it as "root". For each transaction *Trans* in *DB* do the following.
   a. Select and sort the frequent items in *Trans* according to the order of *L*. Let the sorted frequent item list in *Trans* be [*p* | *P*], where *p* is the first element and *P* is the remaining list. Call *insert_tree*([*p* | *P*], *T*).
   b. The function *insert_tree*([*p* | *P*], *T*) is performed as follows. If *T* has a child *N* such that *N.item-name = p.item-name*, then increment *N*'s count by 1; else create a new node *N*, and let its count be 1, its parent link be linked to *T*, and its node-link be linked to the nodes with the same *item-name* via the node-link structure. If *P* is nonempty, call *insert_tree*(*P*, *N*) recursively.

The FP-growth algorithm for mining frequent patterns with FP-tree by pattern fragment growth is:

Input: a *FP-tree* constructed with the above mentioned algorithm;
D – transaction database;
s – minimum support threshold.
Output: The complete set of frequent patterns.
Method:
call FP-growth(*FP-tree*, *null*).
Procedure FP-growth(*Tree*, *A*)
{
  if *Tree* contains a single path *P*
  then for each combination (denoted as *B*) of the nodes in the path *P* do
  generate pattern $B \cup A$ with *support=minimum support of nodes in B*
  else for each *ai* in the header of the *Tree* do
  {
    generate pattern $B = a_i \cup A$ with *support* = a_i*.support*;
    construct *B*'s conditional pattern base and *B*'s conditional *FP-tree*
    TreeB;
    if *TreeB* ≠ Ø
    then call FP-growth(*TreeB*, *B*)
  }
}

---

3. ANALYSIS

Recently FP-Growth algorithm has gained a lot of popularity and grabbed the focus of research. In the following section some limitations of FP-Growth is discussed to support the choice of Apriori over FP-Growth for this work. In [4], Han et al. introduce a quite novel algorithm to solve the frequent itemset mining problem. 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.

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.

3.1 Increasing the Efficiency of Association Rules Algorithms

The computational cost of association rules mining can be reduced in many ways, e. g.:

- By reducing the number of passes over the database
- By sampling the database
- By adding extra constraints on the structure of patterns
- Through parallelization.

In recent years much progress has been made in all these directions. Some of the related approaches towards the efficiency enhancement of Association Rule Mining algorithms are given below:

3.2 GA based Association Rule Mining

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. Genetic algorithm, colony algorithm, evolutionary algorithm and particle swarm algorithm are instances of single objective association rule mining algorithms. A few of these algorithms has been used for multi objectives. 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. Just interesting rules with constant length are discovered. In this method, the genes contain rank of fields. Final chromosome should be the best one and the process stops if it reaches the predefined number of iterations or the result is not improved. The fitness function is defined such a way that it stays in local optimum and causes many rules to be generated. Kaya proposed genetic clustering method [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]. Dehuri et al proposed a rule mining method using multi objectives called multi objective genetic algorithm (MOGA) [11]. Later, they improved the performance by parallel association rule [12]. Gilan Atlas et al proposed a multi-objective differential evolution algorithm for mining numeric association rules. Later, they proposed another numeric association rule mining method using rough particle swarm algorithm which had some improvements in performance and precision compared to the previous one.

3.3 Fuzzy Association Rule Mining

Most work in fuzzy ARM is directed towards theoretical aspects of finding ways to discover better fuzzy association rules, both positive as well as negative ones. [13], [14], [15] discuss in detail about fuzzy implicators and t-norms for discovering fuzzy association rules, especially negative association rules, from datasets. [15], actually talks in depth about new measures of rule quality which would help the overall fuzzy ARM process. In fact, Hüllermeier et. al. [17], [18] make a very detailed analysis of t-norms and implicators with respect to fuzzy partitions, and provide a firm theoretical basis for their conclusions. [20], describes in great detail the general model and application of fuzzy association rules. Fuzzy Apriori, the de-facto algorithm used for fuzzy association rule mining, is used in [14], [15], [16], [19], [21]. Verlinde et al [16] describe in a fair amount of detail as to how fuzzy Apriori can be used to generate fuzzy association rules. Fuzzy Apriori, like Apriori, uses a record-by-record counting approach albeit the major difference being that it takes into account the fuzzy membership of an itemset in each record in order to calculate its overall count. [16], also very briefly describes a pre-processing technique, to obtain fuzzy attributes from numerical attributes, using FCM. Last, in [19] Hüllermeier and Yi justify the relevance of fuzzy logic being applied to association rule mining in today's data-mining setup.

Apart from these, much work is also done on trying to address the problem of scalability, mining rules qualitatively rather than quantitatively, and mining rules without specifying fixed

minimum support etc. But, since these areas of research are not within the scope of this work, so discussion on those works is omitted over here.

4. CONCLUSION

There are various algorithms for ARM. Almost all of them exploit the basic underlying concept of Apriori. Each of them has some merits and demerits over the other. Apriori however remains one of the most popular algorithms. In this paper different algorithms have been compared and the observations are stated in the analysis section of this paper. Though, Apriori has an aggressive search space pruning strategy, still the support count of candidate itemsets has a heuristic approach. 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.

5. 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. Dissertation, Transnationale Universiteit Limburg, 2002.

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

[13] De Cock, M., Cornelis, C., Kerre, E.E.: Fuzzy Association Rules: A Two-Sided Approach. In: FIP, pp 385-390 (2003).

[14] Yan, P., Chen, G., Cornelis, C., De Cock, M., Kerre, E.E.: Mining Positive and Negative Fuzzy Association Rules. In: KES, pp. 270-276. Springer (2004).

[15] De Cock, M., Cornelis, C., Kerre, E.E.: Elicitation of fuzzy association rules from positive and negative examples. Fuzzy Sets and Systems, 149, 73–85 (2005).

[16] Verlinde, H., De Cock, M., Boute, R.: Fuzzy Versus Quantitative Association Rules: A Fair Data-Driven Comparison. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, 36, 679-683 (2006).

[17] Dubois, D., Hüllermeier, E., Prade, H.: A systematic approach to the assessment of fuzzy association rules. Data Min. Knowl. Discov., 13, 167-192 (2006).

[18] Dubois, D., Hüllermeier, E., Prade, H.: A Note on Quality Measures for Fuzzy Association Rules. In: IFSA, pp. 346-353. Springer-Verlag (2003).

[19] Hüllermeier, E., Yi, Y.: In Defense of Fuzzy Association Analysis. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, 37, 1039- 1043 (2007).

[20] Delgado, M., Marin, N., Sanchez, D., Vila, M. A.: Fuzzy Association Rules: General Model and Applications. IEEE Transactions on Fuzzy Systems 11, 214-225 (2003).

[21] Shu-Yue, J., Tsang, E., Yenng, D., Daming, S.: Mining fuzzy association rules with weighted items. In: IEEE International Conference on SMC, pp. 1906-1911, IEEE (2000).