HOW EFFICIENT IS APRIORI: A COMPARATIVE ANALYSIS

[1]Jabeen Sultana, [2]G. Nagalaxmi
[1]Lecturer at IIIT Basar
Email:[1]Jabeens02@gmail.com,[2]nlaxmi.mtech@gmail.com

Abstract
The goal of data mining is to discover knowledge and reveal new, interesting and previously unknown information to the user. Association rule mining forms an important research area in the field of data mining. Association rules discovers patterns and correlations that may be buried deep inside a database. They have therefore become a key data-mining tool and it has been well researched. Many algorithms have been proposed to find association rules in databases with binary and categorical attributes. However many real world databases are in quantitative nature and the current solutions for this kind of databases are so far inadequate. A satisfactory solution would be of great benefit to many fields.

This paper is aimed at proposing new algorithms for mining association rules on the quantitative data. We are proposing a new method for deriving the association rules on the quantitative data by using apriori algorithm, Clustering and fuzzy set concepts. The association rules of quantitative data are represented in two independent ways, one is with statistical terms namely Mean and Standard Deviation, and another way is with fuzzy linguistic terms. The Statistical Association rule consists of quantitative attributes along with the corresponding Mean and Standard Deviation. The representation of the rules was done by means of statistical terms as they can describe the behaviour of quantitative attributes better than the existing methods and the theory of fuzzy sets can be used over relational databases to discover useful, meaningful patterns. We proposed a new approach to infer fuzzy association rules from the quantitative data, which is fully different from the earlier approaches. A case study was done on the Commodity Export data. The data is collected by DGCIS (Directorate General of Commercial Intelligence and Statistics) and available at Reserve Bank of India (RBI) data warehouse, known as Central Database Management System (CDBMS). For our work, we have received the data from CDBMS.

1.1 Introduction

There are many challenging problems in data mining. One of them includes finding association rule mining. Association rule mining includes another problem which is frequent set generation even after existence of different types of algorithms to find out the association rules in given databases. There is a great need of designing efficient and effective algorithms in data mining. What does it mean by Association Rule Mining? Before answering this question. We need to answer another question. What does it exactly mean by frequent set? A frequent set is set which occurs certain number of times in the dataset. After finding all frequent sets we will generate some interesting rule in the form of IF-THEN rules, these rules are noting but association rules. Through which we will generate information from the database or warehouse which we stored over years.

Decision makers use this information to make effective decisions. To make profits by analyzing the existing user data. Our analysis leads to interesting patterns through which decision makers will provide frequently used items to the users through which marketers will end up with the profits. But the main problem comes here is designing effective algorithms which can take large data set as input and provide output within time. So it is the responsibility of the designer to make sure that the algorithms should meet the needs of the decision maker. The main part of the algorithm is to deal with the subsets generated by the algorithm. As the subset generation problems is NP-COMPLETE so it not feasible to design an algorithm which will end up with exponential time complexity. Our algorithm should be optimal in terms of Time complexity and Space complexity and work better over existing algorithms. Even the existing algorithms such as apriori and it's variations will lead to an exponential algorithm. Due to the exponential time complexity of these algorithms we have to go for another algorithm for a better solution. The main objective of this project is to overcome the worst-case exponential time complexity of existing algorithms through designing a new algorithm.

2.1 Background

The concept of Association rule mining is introduced by Apriori Algorithm (Agarwal & Srikanth, 1994). It works following a breadth first search strategy first, for each feature, all frequent similar values (frequent similar subdescriptions with only one feature), are determined first. Combining frequent similar patterns of length 1, candidates to frequent similar patterns of length 2 are obtained. Afterwards, for each pair (Pi,- Pj) of frequent similar subdescriptions with k-1 features, found in the iteration k -1, if Pi and Pj have a common subdescription with k-2 features, they are combined in order to create a new candidate subdescription P* with k features .

In Agrawal and Srikant (1994) a fast algorithm for generating interesting association rules from frequent patterns is described. This algorithm is based on the following property. For each frequent pattern, if the confidence of the rule $X \rightarrow Y$ obtained by separating its features in two disjoint subsets is less than a specified minimum confidence threshold, then the confidence of all rule $X\text{-}Z \rightarrow YZ$, where Z is a sub pattern of X, is also less than the specified minimum threshold. As a consequence of this property, if a rule $X \rightarrow Y$ is not interesting then all rules $X\text{-} Z \rightarrow YZ$, where Z is a sub pattern of X, are also not interesting, and it is not necessary verifying their confidence. In the fast algorithm, for each frequent pattern, first all rules with only one feature in the consequent and the remaining features in the antecedent are generated. Later, for each rule, the features of the antecedent are recursively moved to the consequent and thus new rules are generated, until the confidence of the rule become less than the minimum confidence threshold.

Other algorithms for generating interesting association rules from frequent patterns have been reported in the literature, however they are designed for specific kinds of associations rules or domains (Ayubi, Muyeba, Baraani, & Keane, 2009; Chen & Wei, 2000; Choi & Hyun, 2010; Li-Min, Shu-Jing, & Don-Lin, 2010;Ya-Han & Yen-Liang, 2006).

2.2 Definitions

Association Rule Mining: Finding interesting relationships in among items

Measures of rule interestingness: association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold.

Let $I = \{I1 , I2 , . . . , Im \}$ be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \varphi$.

Support: Percentage of transaction D that contain

$$A \cup B \quad \text{support}(A \Rightarrow B) = P(A \cup B)$$

Confidence:the percentage of transactions in D containing A that also contain B

$$\frac{\textit{support } A \cup B}{\textit{support } A}$$

Additional interestingness measures can be applied for the discovery of correlation relationships between associated items. Association rule mining can be viewed as a **two-step process**:

1. *Find all frequent itemsets*: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min sup.

2. *Generate strong association rules from the frequent itemsets:* By definition, these rules must satisfy minimum support and minimum confidence.

Classification Of Association Rule Mining:

There are many kinds of frequent patterns, association rules, and correlation relationships. Frequent pattern mining can be classified in various ways, based on the following criteria:

1. Based on the types of values handled in the rule
   - ➢ Boolean association rule
   - ➢ Quantitative association rule
2. Based on the kinds of rules to be mined
   - ➢ Association rules
   - ➢ Correlation rules
   - ➢ Strong gradient relationships [1]
3. Based on the number of data dimensions involved in the rule
   - ➢ Multidimensional association rule
4. Based on the levels of abstraction involved in the rule set
   - ➢ Multilevel association rules
   - ➢ Single-level association rules
5. Based on the kinds of patterns to be mined
   - ➢ Frequent itemset mining
   - ➢ Sequential pattern mining
   - ➢ Structured pattern mining

3. Frequent Subset Generation Problem

The frequent subset generation problem includes given a list of transactions with n distinct items we need to find a frequent set among them. To solve this problem we need to scan through the data set and count how many number of times it is present in dataset and then we need to check that the subsets are satisfying the minimum support constraint are not. If they satisfy the constraint then we can find the frequent set. The main problem comes with the space complexity in this problem is even if we provide the input as all possible combination for a given set of items. We require exponential space. If space complexity is not a problem. Algorithm steps will increase exponentially so finally we end up with a exponential problem. We have different sort of approaches to solve this problem in the next chapter we are going to discuss a brute-force algorithm, apriori Algorithm and several experimental algorithm we've tried to solve the problem. Each Algorithm has been explained with an example. Moreover in designing a solution to this problem we will make sure that the each and every approach we are going to check is mathematically complete and optimal. In the process of designing the algorithm we have encountered many problems. Even then we have improved the performance of the algorithm and designed a remedy for each problem

4. Implementation of Algorithms

There are different types of algorithm to solve the problem of frequent set generation. Apriori and it's variations work effectively in almost all cases. In this chapter before going to the Apriori algorithm we will discuss Brute-force approach to solve the problem. Then after we will go for the apriori algorithm. At the end we will discuss our experimental algorithms.

4.1 The Brute-force Approach:

In this approach all possible subsets of set of distinct items in a transaction set. And check if any one them is frequent by comparing with each and every transaction. As it is known already that the brute-force approaches lead to exponential time it is not preferable.

Pseudo-code for Brute-force approache:

```
input:Transaction T, minimum support
min_sup
output:Frequent set FS
begin
 1.    Generate all possible subsets S of
       Transaction T.
 2.    while each subset s ϵ S
 3.              sup=support(s)
 4.              ifsup>=min_sup
 5.    add s to FS
 6.              elsediscard the subset
 7.    end while
 8.    Find Maximal Frequent Item Sets[2] from
       FS
 9.    Generate Rules for Maximal Frequent
       Item Set
 end
```

Drawbacks:

In this method if d distinct items are present in transaction then 2^d -1 Subsets need to be generated for which computation time increases exponentially with increase in d. This is the main pitfall of this method.

4.2 The Apriori Algorithm:

The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties. Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted Lv1 . Next, L1 is used to find L2 , the set of frequent 2-itemsets, which is used to find L3 , and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an impor-tant property called the Apriori property is used to reduce the search space. We will first describe this property, and then show an example illustrating its use.

Apriori property: All nonempty subsets of a frequent itemset must also be frequent. This property is used in the algorithm by using two step process:

1. *The join step*:To find Lk , a set of candidate k-itemsets is generated by joining Lk −1 with itself. This set of candidates is denoted Ck .
2. The prune step: Ck is a superset of Lk , that is, its members may or may not be frequent, but all of the frequent k-itemsets are included inCk all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to Lk .

Pseudo-Code of Apriori Algorithm:

```
F1 = {frequent 1-itemsets}
For (k = 2; Fk-1 ≠ 0; k++) loop
    Ck = generate (Fk-1);
    For all transactions x ∈ D loop
        Cx = generate_subset (Ck, x);
        //candidate generation;
        For all candidates c ∈ Cx loop
            c.count++
        end loop;
    end loop;
    Fk = { c ∈ Ck | c.count ≥ minsup}
end loop;
Return ∪k {Fk}
```

To improve the efficiency of apriori algorithm there are certain variations

1.Hash-based technique:A hash-based technique can be used to reduce the size of the candidate k-itemsets, Ck , for k > 1. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1 , from the candidate 1-itemsets in C1 , we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts . A 2-itemset whose corresponding bucket count in the hash table is below the support

2.Transaction reduction :A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k + 1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for j-itemsets, where j > k, will not require it.

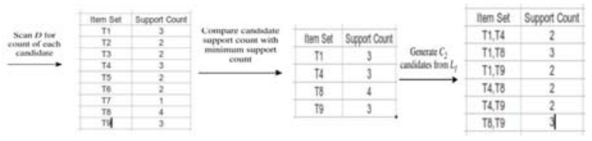
3.Partitioning :A local frequent itemset may or may not be frequent with respect to the entire database, D. Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions. Therefore, all local frequent itemsets are candidate itemsets with respect to D. The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to D. In Phase II, a second scan of D is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

4.Sampling: The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, we trade off some degree of accuracy against efficiency. The sample size of S is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. Because we are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets.

5.Dynamic itemset counting :A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires fewer database scans than Apriori.

Example 1.1:	
Item Sets	TID sets
1	T1,T4,T5,T8,T9
2	T1,T2,T3,T4,T6,T8,T9
3	T3, T5, T6, T7,T8, T9
4	T2,T4
5	T1,T8

Implementation of apriori algorithm:
minimum support threshold=3
C1 l1 C2

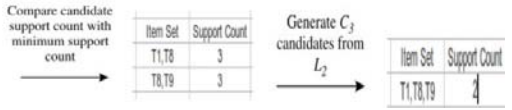

Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1

Item Set	Support Count
T1	3
T2	2
T3	2
T4	3
T5	2
T6	2
T7	1
T8	4
T9	3

Item Set	Support Count
T1	3
T4	3
T8	4
T9	3

Item Set	Support Count
T1,T4	2
T1,T8	3
T1,T9	2
T4,T8	2
T4,T9	2
T8,T9	3

L2 C3


Compare candidate support count with minimum support count
Generate C_3 candidates from L_2

Item Set	Support Count
T1,T8	3
T8,T9	3

Item Set	Support Count
T1,T8,T9	2

Figure 4.1 Apriori Example

There are no C3 items satisfying minimum support threshold we stop with sets {T1,T8} , {T8,T9}.

Disadvantages:

This algorithm will lead to a exponetial time complexity in worst-case and it will generate 2^d-1 subsets which is not optimal.

Solution to This Problem:

Rather than generating the frequent sets sequentially start searching the frequent set by using bisection method which will be completed in log n steps.

Bisection Method:

This method logic is similar to Binary search method. In a sorted sequence if we would like to find out the a number we directly go to middle of array and we compare with the middle element and if it is greater than searching element we proceed further if not we proceed lower half. This will Take log n steps. Based on method we would like to present

4.3 Experimental Algorithm 1:

The main motivation of this algorithm is to overcome the problem of exponential time complexity through reducing the number of steps. As power set size is n the there will be $2^{\wedge n}$ sub sets we can align the increasing order with respective their size.

Ex: consider a 5 element set and Tree generated through brute-force algorithm to find all sub sets of the set. The tree generated has 32 nodes.

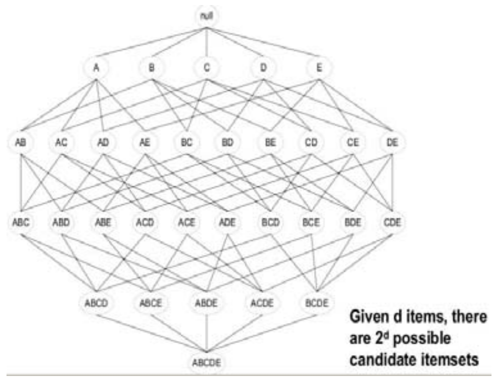


Figure 4.2 Power Set

In this algorithm we use bisection method to find the frequent set in generated tree. The advantage of this algorithm of it will find the frequent set in log n number of steps through either selecting only upper portion of the tree or lower portion of the tree. The bisection method is mathematical method to find roots this method is also used in binary search method that why the binary search time complexity is logn.

Input: Dataset, min_support
Output: Frequent Set **begin:**

```
1. N= no. of distinct items
2. find substest of size-1
3. support-1= find support of size-1 subsets
4. if max(support-1)>=min_support
5. S=subset of size-N
6.     if (support(S)>=min_support)
7.         Frequent Set = S
8.     else
```

```
9.    low=1
10.       high=N
11.       mid= ┌(low+high)/2┐
12    for i in range 1 to logN
13.       S=subsets of size-mid
14.       support= support(S)
15.       if max(support)<min_support
16.           high=mid
17.           mid= ┌(low+high)/2┐
18.       else
19.           low=mid;
20.           mid= ┌(low+high)/2┐
              Index=index of subset with
21.   max(support) in S
22.           Frequent Set=S(index)
23.       end
24.     end
25.   end
26.   end
27.   end
```

size 1	
Item	Support
T1	3
T2	2
T3	2
T4	3
T5	2
T6	2
T7	1
T8	4
T9	3

Size 2	
Items	Sup port
T1,T4	2
T1,T8	3
T1,T9	2
T4,T8	2
T4,T9	2
T8,T9	3

Sub set size	No of subsets
1	5
2	10
3	10
4	5
5	1

Size n	
Items	support
T1,T2,T3,T4	1
,T6,T8,T9	

Subsets	Support
T1,T4,T8	2
T1,T4,T9	2
T4,T8,T9	2
T1,T8,T9	1

Fig. 4.3:subsets in tree

In the above example consist of five distinct elements and there will five different size sub sets in tree. We need to count the number of different subsets present in tree.

Step 1: compute support of size 1 and size n Size 1 is satisfying the minimum support but not size n=7 is not satisfying the minimum support.

Step2: as n=7 is not satisfying the support we will jump to n=7/2=3.5 we need to take lower bound on this. Then we will get 3. The data Items that satisfying the minimum support are T1,T4,T8,T9 we need form all the subsets of size three with these Items They are

Step 3: We didn't find a set which can satisfy our minimum support with size three.then we need to average 1 and 3 which is 2. We need to find out all subsets of size two and check which are satisfying the minimum support

There are only two sets which can satisfy the {T1,T8},{T8,T9} which can satisfy the minimum support. So {T1,T8},{T8,T9} are the frequnet sets.

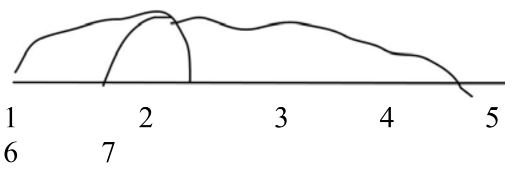


Figure 4.4 Eperimental Algorithm-1	
Step 1: We've compared 1,7	Implemenation
Step 2: We've checked at 3	
Step 3: At 2 we found our Frequent subsets.	
Advantages:	

1.We can complete this algorithm by searching at all log n subsets.

Disadvantages:

1.If total size of the transaction is greater than 10 then it will be difficult to apply this algorithm as it is difficult to find out when n>10. It is NP-COMPLETE to find out.

Solution To This problem:

We know it optimal to find all possible subsets for n=10 then we divide each transcation such that we will get atmost ten elements elements in partition.

4.4. Experimental Algorithm-2:

To over come the problem in Algorithm 1 we will proceed with another procedure. Which uses partion.

Input: Dataset, min_support
Output: Frequent Set **begin**

1. N=no. Of partitions
2. C=no. Of colums in Dataset
3. X=N/C
4. R=No. Of transactions
5. part=divide the dataset vertically into partitions of size R*X

```
6.  D=no. Distinct elements in Data Set
7.  bol=boolean matrix representing DataSet
1.  count=the no. Transaction in which each element present
2.  place it in the respective index of matrix bol
3.  for each partition in part set
4.      n_d=no. Of distinct elements in partition
5.      for i in range 1 to X
6.          find subset of size-i
7.          find support of subsets
8.          if support>min_support
16.             store the subsets
17.         end
18.       end
19.     end
20. end
```

In this algorithm the main step is to divide the transcation in such a way that every partion will get atmost ten distinct elements.

Ex: Fig 4.5

Tid	Items
T1	1 3 9 13 23 25 34 36 38 40 52 54 59 63 67 76 85 86 90 93 98 107 113
T2	2 3 9 14 23 26 34 36 39 40 52 55 59 63 67 76 85 86 90 93 99 108 114
T3	2 4 9 15 23 27 34 36 39 41 52 55 59 63 67 76 85 86 90 93 99 108 115
T4	1 3 10 15 23 25 34 36 38 41 52 54 59 63 67 76 85 86 90 93 98 107 113
T5	2 3 9 16 24 28 34 37 39 40 53 54 59 63 67 76 85 86 90 94 99 109 114
T6	2 3 10 14 23 26 34 36 39 41 52 55 59 63 67 76 85 86 90 93 98 108 114
T7	2 4 9 15 23 26 34 36 39 42 52 55 59 63 67 76 85 86 90 93 98 108 115

Step 1:

Max Tranasaction size =23

Each partion size=23/10=2.3 (Upper bound is 3)

	part1	part2	part3	part 4	p5	part 6	part7	p8	p9	part 10	p11	p12
T1	1 3 9	13	23 25	34 36 38	40	52 54 59		76	85 86	90 93 98	107	113
T2	2 3 9	14	23 26	34 36 39	40	52 55 59	63 67	76	85 86	90 93 99	108	114
T3	2 4 9	15	23 27	34 36 39	41	52 55 59	63 67	76	85 86	90 93 99	108	115
T4	1 3	10 15	23 25	34 36 38	41	52 54 59	63 67	76	85 86	90 93 98	107	113
T5	2 3 9	16	24 28	34 37 39	40	53 54 59	63 67	76	85 86	90 94 99	109	114
T6	2 3	10 14	23 26	34 36 39	41	52 55 59	63 67	76	85 86	90 93 98	108	114
T7	2 4 9	15	23 26	34 36 39	42	52 55 59	63 67	76	85 86	90 93 98	108	115

Step 2:

In above partion we can generalize one thing that each partion is having at most of size is 3. So we need to find only $^{10}C3$. This will be an advantage and it will decrease time too. We need to solve this partions independently to form a frequent subset.

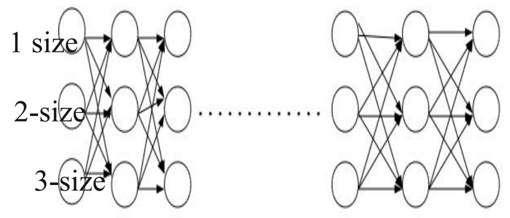


Figure 4.6 Formation of subset with size >=3

Disadvantage

Due to this approach the is a problem after partioning the input we will have size1,size 2,size3 subsets we need to form subsets with this partitions to find the frequent subsets. The formation of all subsets will lead to an exponetial algorithm again which is non-optimal.

Now we would like to present a completely a defferent algorithm which will not exponential time complexity.

4.5 Experimental Algorithm- 3:

Input: Dataset, min_support

Output: Frequent Set

begin

Data=Transactions r=no.of rows in Data

c=no.of columns in Data

m=maximum no. in transactions

count[1,m]=count of each item in Datafreq=no.

of items which are satisfying minimum support
size=no.of items in freq set

```
while(done==1)
{       freq_check(frq);
        if (cheek==1)
        {
        print(freq is frequent) done=0;
        }
        else
        {
        calc_prob(freq); remove_item(freq);

        }
}end

frq_check(array a) #checks whether the all
elements are satisfying the minimum_support
are not
calc_prob(freq)
{prob[i]= count[i]/no.of transactions}

remove_item(freq)
        {   remove();#removes the item with
        less probablity if(remove_items>2)
        {
        form no of freq sets by removing the
each element once
        }
        ** each item size should get decreased
by 1
}
```

Ex:: min_sup=3 Boolean Matrix:
FC: Fig 4.7

freq_count After sorting:			
	Count	index	Sum
1	4	1	22
2	4	2	22
3	2	3	17
4	1	4	4
5	2	5	9

Item Sets	TID sets
1	T1,T4,T5,T8, T9
2	T1,T2,T3,T4, T6,T8,T9
3	T3, T5, T6, T7,T8, T9
4	T2,T4
5	T1,T8

Count set number	4	2	1
count	2	2	1

Count of everyFrequent count
As the maximum count set is 4 and which is not satisfying the minimum support we reduce the size of the set by using probability. We will calculate the probability of every item then remove the item with less probability. This step is repeated whenever we need to reduce size of the set.

Maximum Count set:{T1,T4,T8,T9}
P(T1)=3/13=0.23
P(T4)=3/13=0.23
P(T8)=4/13==0.31
P(T9)=3/13=0.23

	Count	index	sum
1	4	1	22
2	4	2	22
3	2	3	17
4	2	5	9
5	1	4	4

Item	count
T1	3
T4	3
T8	4
T9	3
	13

Fig:4.8

Here T1,T4,T9 are having the less probability so we will form sub sets by removing each element.

As size 3 is not satisfying the minimum support we need to go for size 2. Now we need to findout the sets with size 2 by using same formula.

P(T1)=3/13=0.23
P(T4)=3/13=0.23
P(T8)=4/13==0.31
P(T9)=3/13=0.23

T4,T8,T9			T1,T8,T9		T1,T4,T8		
T4,T8		2	T1,T8	3	T1,T8		3
T9,T8		3	T8,T9	3	T4,T8		3

Fig:4.9
Now we've ended with {T1,T8},{T8,T9},{T4,T8} which are frequent sets.

Disadvantage:

This algorithm is not complete due to dependency on the probability. As we are leaving a item in decreasing the size of the frequent set. That item may be in a frequent set.

5.Results

The presented algorithms in the chapter 4 are Brute-force Algorithm, Apriori Algorithm, Experimental Algorithm 1, Experimental Algorithm 2, Experimental Algorithm 3. Brute-force Algorithm does not work well as the number of distinct elements increases and it is having exponential time complexity.

Apriori Algorithm has many more advantages than any existing algorithms for finding a frequent set and association rules. But it requires as many scans as the number of subsets in each stage. Join step will take much time in worst-case and leads to an Exponential time complexity.

Experimental Algorithm 1 which depicts the way of binary search. It works efficiently and optimally when number of distinct elements <10.Even it search half of the tree. It will not work good when the number of distinct elements increases. Finally it will become an exponetial algorithm and consume much time in generating the subsets.

Experimental Algorithm 2 it is designed to over come the problems which are faced by the Experimental Algorithm 1. In This Algorithm we've divided each transaction and computed independently. Even then it lead us to an exponential algorithm in joining and forming the subsets greater than 3. It is just because we need to take all combinations.

Experimental Algorithm 3 it is completely different approach than the existing algorithms it works better than the rest of two experimental algorithms. It is not complete because it finds the frequent sets through probability.

Fig:4.10

Transaction 1		
	Item set	count
	T4,T8, T9	2
	T1,T8, T9	1
	T1,T4, T8	2

If we do the same for 2nd transaction too		
	Item set	count
	T4,T8,T9	2
	T1,T8,T9	1
	T1,T4,T8	2

6.Conclusions and Scope for Future Work

In this papers we've tried to design new algorithm to find out the frequent set. Comparing with the other algorithms the experimental algrorithm-3 will work efficiently than the rest of the two algorithms but it is not complete because it works on probability. In future we will try to improve the existing algorithm i.e experimental algorithm-3 to make it work completely on any given data.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In 20th International Conference on Very Large Databases (VLDB). 1994.

[2]The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns Guizhen Yang Department of Computer Science and Engineering University at Buffalo, The State University of New York Buffalo, NY 14260-2000 gzyang@cse.buffalo.edu

[3]Proposing an Efficient Method for Frequent Pattern Mining Vaibhav Kant Singh, Vijay Shah, Yogendra Kumar Jain, Anupam Shukla, A.S. Thoke, Vinay Kumar Singh, Chhaya Dule, Vivek Parganiha

[4]An Information-Theoretic Approach to Quantitative Association Rule Mining Yiping Ke James Cheng Wilfred Ng Department of Computer Science Hong Kong University of Science and Technology Clear Water Bay, Kowloon, Hong Kong, China {keyiping, csjames, wilfred}@cs.ust.hk

[5]Mining frequent patterns and association rules using similarities Ansel Y. Rodríguez-González a,⥣, José Fco. Martínez-Trinidad a, Jesús A. Carrasco-Ochoa a, José Ruiz-Shulcloper b

[6]An Efficient Approach of Association Rule Mining on Distributed Database Algorithm Neha Saxena1, Rakhi Arora2, Ranjana Sikarwar3 and Ashika Gupta4

[7]A Review Approach on various form of Apriori with Association Rule Mining Ms. Pooja Agrawal1, Mr. Suresh kashyap2, Mr.Vikas Chandra Pandey3, Mr. Suraj Prasad Keshri4 [8]Improving Efficiency of Apriori Algorithm Using Transaction Reduction Jaishree Singh*, Hari Ram**, Dr. J.S. Sodhi***