Improving Efficiency of Apriori Algorithm

Ch.Bhavani, P.Madhavi
Assistant Professors, Department of Computer Science, CVR college of Engineering, Hyderabad, India.

Abstract -- *Apriori algorithm has been vital algorithm in association rule mining. The main idea of this algorithm is to find useful frequent patterns between different set of data. It is a simple and traditional algorithm, Apriori employs an iterative approach known as level wise search. But, this algorithm yet have many drawbacks. Based on this algorithm, this paper indicates the limitation of the original Apriori algorithm of wasting time for scanning the whole database searching on the frequent itemsets, and presents an improvement on Apriori by reducing that wasted time.*

Keywords – apriori algorithm, frequent itemsets, support.

I. INTRODUCTION

Data Mining is a way of obtaining undetected patterns or facts from massive amount of data in a database. Data Mining is also known as knowledge discovery in databases (KDD). Data mining is about solving problems by analyzing the data present in the database and identifying useful patterns. Patterns allow us to make prediction on new database. Data mining is more in demand because it helps to reduce cost and increases the revenues. The various applications of data mining are customer retention, market analysis, production control and fraud detection. Data Mining is designed for different databases such as object-relational databases, relational databases, data warehouses and multimedia databases. Data mining methods can be categorized into classification, clustering, association rule mining, sequential pattern discovery, regression etc. It helps to find the association relationship among the large number of database items and its most typical application is to find the new useful rules in the sales transaction database, which reflects the customer purchasing behaviour patterns, such as the impact on the other goods after buying a certain kind of goods. These rules can be used in many fields, such as customer shopping analysis, additional sales, goods shelves design, storage planning and classifying the users according to the buying patterns, etc. The techniques for discovering association rules from the data have traditionally focused on identifying relationships between items telling some aspect of Human behaviour, usually buying behaviour for determining items that customers buy together. All Rules of this type describe a particular local pattern. The group of association rules can be easily interpreted and communicated. Apriori algorithm is the traditional algorithm used for generating the frequent itemsets from the itemsets in the transactions of the data bases. A basic property of apriori algorithm is “every subset of a frequent item sets is still frequent item set, and every superset of a non-frequent item set is not a frequent item set”. This property is used in apriori algorithm to discover all the frequent item sets. Further in the paper we will see more about the Apriori algorithm steps in detail.

II. TRADITIONAL APRIORI ALGORITHM

Apriori is very much basic algorithm of Association rule mining. It was initially proposed by R. Agrawal and R Srikant for mining frequent item sets. This algorithm uses prior knowledge of frequent item set properties that is why it is named as Apriori algorithm.

Before starting the actual Apriori algorithm, first we will see some the terminologies used in the apriori algorithm.

Itemset **-** Itemset is collection of items in a database which is denoted by
I = {i1, i2,…, in}, where n is the number of items.

Transaction **–** Transaction is a database entry which contains collection of items. Transaction is denoted by T and T □ I. A transaction contains set of items T={i1,i2,..,in}.

Minimum support **–** Minimum support is the condition which should be satisfied by the given items so that further processing of that item can be done. Minimum support can be considered as a condition which helps in removal of the in-frequent items in any database. Usually the Minimum support is given in terms of percentage.

Frequent itemset (Large itemset) **–** The itemsets which satisfies the minimum support criteria are known as frequent itemsets. It is usually denoted by Li where i indicate the i-itemset.

Candidate itemset **–** Candidate itemset are items which are only to be consider for the processing. Candidate itemset are all the possible combination of itemset. It is usually denoted by Ci where i indicate the i-itemset.

Support **–** Usefulness of a rule can be measured with the help of support threshold. Support helps us to measure how many transactions have such itemsets that match both sides of the implication in the association rule.
Consider two items A and B. To calculate support of

A □ B the following formula is used,

Supp(A□ B)=(number of transactions containing both A &B) / (Total number of transactions)

Confidence –Confidence indicates the certainty of the rule. This parameter lets us to count how often a transaction's itemset matches with the left side of the implication with the right side. The itemset which does not satisfies the above condition can be discarded.
Consider two items A and B. To calculate confidence of A □ B the following formula is used,
Conf(A□ B)=(number of transactions containing both A & B)/(Transactions containing only A)
Note: Conf(A□ B) might not be equal to conf(B□ A).

***Apriori Algorithm* -** It employs an iterative approach known as a breadth-first search (level-wise search) through the search space, where k-itemsets are used to explore (k+1)-itemsets. The working of Apriori algorithm is fairly depends upon the Apriori property which states that" All nonempty subsets of a frequent itemsets must be frequent". It also described the anti monotonic property which says if the system cannot pass the minimum support test, all its supersets will fail to pass the test. Therefore if the one set is infrequent then all its supersets are also frequent and vice versa. This property is used to prune the infrequent candidate elements. In the beginning, the set of frequent 1-itemsets is found. The set of that contains one item, which satisfy the support threshold, is denoted by L. In each subsequent pass, we begin with a seed set of itemsets found to be large in the previous pass. This seed set is used for generating new potentially large itemsets, called candidate itemsets, and count the actual support for these candidate itemsets during The pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large (frequent), and they become the seed for the next pass. Therefore, L is used to find L!, the set of frequent 2-itemsets, which is used to find L , and so on, until no more frequent k-itemsets can be found. The basic steps to mine the frequent elements are as follows: ·

- ***Generate and test:*** In this first find the 1-itemset frequent elements L by scanning the database and removing all those elements from C which cannot satisfy the minimum support criteria.

- ***Join step:*** To attain the next level elements Ck join the previous frequent elements by self join i.e. Lk-1* Lk-1 known as Cartesian product of Lk-1. I.e. This step generates new candidate k-itemsets based on joining Lk-1 with itself which is found in the previous iteration. Let Ck denote candidate k-itemset and Lk be the frequent k-itemset.

- ***Prune step*:** Ck is the superset of Lk so members of Ck may or may not be frequent but all K ' 1 frequent itemsets are included in Ck thus prunes the Ck to find K frequent itemsets with the help of Apriori property. I.e. This step eliminates some of the candidate k-itemsets using the Apriori property A scan of the database to determine the count of each candidate in Ck would result in the determination of Lk (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to Lk). Ck, however, can be huge, and so this could involve grave computation. To shrink the size of Ck, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of candidate k-itemset is not in Lk-1 then the candidate cannot be frequent either and so can be removed from Ck. Step 2 and 3 is repeated until no new candidate set is generated.

Table 1 SAMPLE DATA SET

TID	Items
T1	A, C, D
T2	B, C, E
T3	A, B, C, E
T4	B, E

Performing the first step that is scanning the database to identify the number of occurrences for a particular item. After the first step we will get C1 which is shown in Table 2.

Table 2 C1

Items	Support count
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

The next step is the pruning step in which the itemset support is compared with the minimum support. The itemset which satisfies the minimum support will only be taken further for processing. Assuming minimum support here as 2. We will get L1 from this step.

Table 3 shows the result of pruning.

Table 3 L1

Items	Support count
{A}	2
{B}	3
{C}	3
{E}	3

Now the candidate generation step is carried out in which all possible but unique 2-itemset candidates are created. This table will be denoted by C2. Table 4 shows all the possible combination that can be made from Table 3 itemset

Table 4 C2

Items	Support count
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

Now pruning has to be done on the basis of minimum support criteria. From Table 4 two itemsets will be removed. After pruning we get the following results.

Table 5 L2

Items	Support count
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

The same procedure gets continued till there are no frequent itemsets or candidate set that can be generated. The further processing is described in Table 6 and Table 7.

Table 6 C3

Items	Support count
{A, B, C}	1
{A, B ,E}	1
{B, C, E}	2

Table 7 L3

Items	Support count
{B, C, E}	2

Pseudo Code -

```
$C_k$: Candidate itemset of size k
$L_k$ : frequent itemset of size k
$L_1$ = {frequent items};
for (k = 1; $L_k$ !=□ ; k++)do begin
$C_k$+1 = candidates generated from $L_k$;
for each transaction t in database do increment the count of all candidates in $C_k$+1 that are contained in t
$L_k$+1 = candidates in $C_k$+1 with min_support
end
return□ kLk;
```

It is no doubt that Apriori algorithm successfully finds the frequent elements from the database. But as the dimensionality of the database increase with the number of items then:

- More search space is needed and I/O cost will increase.
- Number of database scan is increased thus candidate generation will increase results in increase in computational cost.

Therefore many variations have been taken place in the Apriori algorithm to minimize the above limitations arises due to increase in size of database. These subsequently proposed algorithms makes an improvement over the traditional Apriori Algorithm by

- Reducing the no.of passes of transaction database scans
- Shrink number of candidates
- Facilitate support counting of candidates

III. *Review On Various Improved Apriori Algorithms*

3.1 Improved apriori based on matrix

***Events*:** One transaction of commodity is an event. That is an Event equals one Transaction containing various Items. Event Database (D): An event T in D can be shown as Ti , Where Ti is unique in the whole Database. First step in this improved apriori is to make a Matrix library. The matrix library (mat) contains a binary representation where 1 indicates presence of item in transaction and 0 indicates the absence. Assume that in the event Matrix library of database D, the matrix is A mxn , then the corresponding BOOL data item set of item Ij(1<= j <= n)in Matrix Amxn is the mat of Ij, Mati is items in the mat. Table 8 shows the Sample database and the 3rd column is binary representation of the items in the transaction

Table 8 SAMPLE DATA BASE

TID	List of Items	I1 I2 I3 I4 I5
T1	I1, I3, I4	1 0 1 1 0
T2	I2, I3	0 1 1 0 0
T3	I5, I2	0 1 0 0 1
T4	I2, I3	0 1 1 0 0
T5	I3, I4, I5	0 0 1 1 1
T6	I2, I4	0 1 0 1 0
T7	I4, I5	0 0 0 1 1
T8	I2, I1, I5	1 1 0 0 1
T9	I3, I4, I5	0 0 1 1 1

For 1-itemset matrix represented is used (i.e.)
MAT(I1) = 100000010
MAT(I2) = 011101010
MAT(I3) = 110110001
MAT(I4) = 100011101
MAT(I5) = 001010111

Now by counting the number of 1's in the matrix we can easily find the occurance of that item.
For 2-itemset we can multiply the binary representation of the items to get the occurance of that items together. To find how many times item Ij and Ik are appearing together we have to multiply the MAT(Ij) and MAT(Ik). (i.e) MAT(Ij ,Ik)=MAT(Ij) * MAT(Ik).

MAT(I3,I4) = MAT(I2) * MAT(I4) = 011101010 * 100011101 = 000001000
MAT(I3,I4) = 000001000

Then support of these two items can be calculated as follows:
Support (I3,I4)= (Nos. of times Appearing together/Tot. Transaction) = 1 / 9 Similarly the same procedure can be followed for all possible itemset. This algorithm needs to scan the database only once and also does not require to find the candidate set when searching for frequent itemset.Table 9 provides the computational time of Apriori and improved apriori.

Table 9

Record number	Apriori Computing time(ms)	Improved Apriori Computing time(ms)
500	1787	35
1000	8187	108
1500	44444	178
2000	46288	214
2500	97467	292
3000	199253	407
3500	226558	467
4000	310379	569
5000	155243	470

3.2 Optimized Algorithm

In the Apriori algorithm, C_k-1 is compared with support level once it was found. Item sets less than the support level will be pruned and L_k-1 will come out which will connect with itself and lead to C_k. The optimized algorithm is that, before the candidate item sets C_k come out, further prune L_k-1, count the times of all items occurred in L_k-1, delete item sets with this number less than k-1 in L_k-1. In this way, the number of connecting items sets will decrease, so that the number of candidate items will decline.

The Realization of Algorithm

According to the properties of frequent item sets, this algorithm declines the number of candidate item sets further. In other words, prune L_k-1 before *Ck* occur using L_k-1. This algorithm can also be described as following:
Count the number of the times of items occur in L_k-1 (this process can be done while scan data *D*);
Delete item sets with this number less than *k*-1 in L_k-1 to get L_k-1. To distinguish, this process is called Prune 1 in this study, which is the prune before candidate item sets occur; the process in Apriori algorithm is called Prune 2, which is the prune after candidate item sets occur. Thus, to find out the k candidate item sets, the following algorithm can be taken:

Prune l(L_k-1), that is executing Prune 1 to L_k-1;
Use L_k-1 to connect with its elements and get the *k*
candidate item sets *Ck*;
Prune 2(*Ck*), that is executing Prune 2 to and finally get the *k* items candidate set which should calculate its support level
(the superset of *k* items frequent set)
The following is the description of the optimized
algorithm:
Input: affairs database *D*: minimum support level threshold is minsup
Output: frequent item sets *L* in *D*

```
1) $L1$=frequent_1-itemsets($D$);
2) For ($k$=2;$Lk$-1$\neq\varphi$;$k$++);
3) Prune$1$($L_k$-1);
4) $Ck$=apriori_gen($L_k$-1;minsup);
5) for all transactions $t \in D$
{
6) $C$= sumset ($Ck$,$t$); find out the subset including $Ck$
7) for all candidates $c \in Ct$
8) { $c$.count ++; }
9) $L_k$ ={$c \in Ck$|$c$.count$\geq$minsup} //result of Prune
2($Ck$) } }
10) Return Answer$\cup k$ $L_k$
```

Algorithm**: **Prune Function:

Input: set *k*-1 frequent items of L_k-1 as input parameter
Output: go back and delete item sets with this number less
than k-1 in L_k-1
Procedure Prune *1*(L_k-1)

```
1) for all itemsets $L1 \in L_k$-1
2) if count($L1$)$\leq k$-1
3) then delete all $Lj$ from $L_k$-1 ($L1 \in L_k$-1)
4) reture $L'_k$-1 // go back and delete item sets with this
number less than $k$-1 in $L_k$-1
```

Chart 3-1 shows the process of the algorithm finding out the frequent item sets, the minimum support level is 2.

TABLE 10: CANDIDATE ITEM SETS *C1*
FREQUENT ITEM SETS *L1*

TID	ITEM LIST
T1	A, B, C
T2	A, B, C, D
T3	A, B, D, E
T4	B, E, F
T5	A, B, D, F
T6	A, C, D, E

From the above dataset, the Candidate Item Set *C*2

Table 11

ITEM SET	SUPPORT LEVEL	ITEM SET	SUPPORT LEVEL
A, B	4	B, F	2
A, C	2	C, D	2
A, D	5	C, E	1
A, E	2	C, F	0
A, F	1	D, E	2
B, C	1	D, F	1
B, D	4	E, F	1
B, E	2		

Occur Frequent Item Set *L*2

Table 12

ITEM SET	SUPPORT LEVEL	ITEM SET	SUPPORT LEVEL
A, B	4	B, E	2
A, C	2	B, F	2
A, D	5	C, D	2
A, E	2	D, E	2
B, D	4		

Now further prune the candidate table, by counting the items and compariong it with the minimum support count.

Table 13

ITEM	SUPPORT LEVEL
A	4
B	4
C	2
D	4
E	3
F	1

As item F has lower support count than the minimum support count, remove the itemsets which contain F in them.

L'$_2$ after further pruning

Table 14

ITEM SET	SUPPORT LEVEL	ITEM SET	SUPPORT LEVEL
A, B	4	B, E	2
A, C	2	C, D	2
A, D	5	D, E	2
A, E	2		
B, D	4		

Occur Candidate Item Set *C*3

Table 15

ITEM SET	SUPPORT LEVEL	ITEM SET	SUPPORT LEVEL
A, B, C	1	A, D, E	2
A, B, D	4	B, D, E	1
A, B, E	1	B, C, D	1
A, C, D	2	C, D, E	1
A, C, E	1		

After pruning

Table 16

ITEM SET	SUPPORT LEVEL
A, B, D	4
A, C, D	2
A, D, E	2

Further pruning, by counting the items after pruning is empty

Table 17

ITEM SET	SUPPORT LEVEL
A	3
B	1
C	1
D	3
E	1

***Advantage of the optimized Algorithm*:** The basic thought of this optimized algorithm is similar with the apirori algorithm, which is they all get the frequent item set *L*1 which has support level larger or equal to the given level of the users via scan the database *D*.

Then repeat that process and get *L*2 , *L*3……*Lk*.

But they also have differences. The optimized algorithm prunes *Lk*-1 before *Ck* is consisted. In other words, count the frequent item set *Lk*-1 which is going to connect. According to the result delete item sets with this number less than *k*-1 in *Lk*-1 to decrease the number of the connecting item set and remove some element that is not satisfied the conditions. This will decrease the possibility of combination, decline the number of candidate item sets in *Ck*, and reduce the times to repeat the process. For large database, this algorithm can obviously save time cost and increase the efficiency of data mining. This is what Apriori algorithm do not have.

Although this process can decline the number of candidate item sets in *Ck* and reduce time cost of data mining, the price of it is pruning frequent item set, which could cost certain time. For dense database (such as, telecom, population census, etc.), as large amounts of long forms occur, the efficiency of this algorithm is higher than Apriori obviously.

3.3 Matrix based Algorithm

The method we propose involves the mapping of the In items and Tm transaction from the database into a matrix A with size mxn. The rows of the matrix represent the transaction and the columns of the matrix represent the items. The elements of matrix A are:

A= [aij] = 1, if transaction i has item j
= 0, otherwise

We assume that minimum support and minimum confidence is provided beforehand.

In matrix A, The sum of the jth column vector gives the support of j thitem.
And the sum of the ith row vector gives the S-O-T, that is, size of ith transaction (no. of items in the transaction).

Now we generate the item sets.

For, 1–frequent item set, we check if the column sum of each column is greater than minimum support. If not, the column is deleted. All rows with rowsum=1 (S-O-T) are also deleted. Resultant matrix will represent the 1- frequent item set.

Now, to find 2-frequent itemsets, columns are merged by AND-ing their values. The resultant matrix will have only those columns whose columnsum>=min_support. Additionally, all rows with rowsum=2 are deleted. Similarly the kth frequent item is found by merging columns and deleting all resultant columns with columnsum <min_support and rowsum=k.When matrix A has 1 column remaining, that will give the kth frequent item set

Algorithm

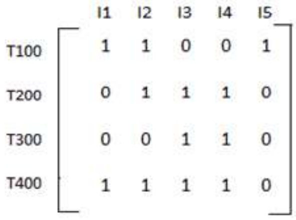
```
1. Create matrix A
2. Set n=1
3. While(n<=k)
           If(columnsum(colj<min_support)
                    If(rowsum(row i)==n)
                              Delete row i;
                         Merge(col j, col j+1)
                         n=n+1
4. end while
5. display A
```

Consider the following example:

Table 18

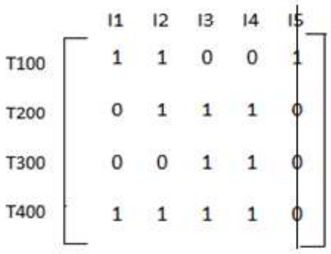
Transactions	Items
T1	A, B, E
T2	B, C, D
T3	C, D
T4	A, B, C, D

The above example shows the number of transactions and items in table. Consider minimum support to be given as 2. Now, we will draw the matrix from above table to show the occurrence of each item in particular transaction, i.e.:



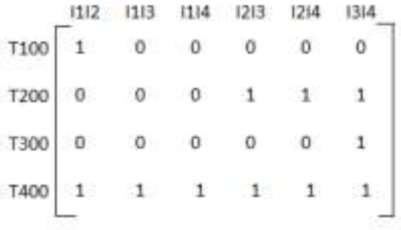
$$\begin{array}{c|ccccc} & I1 & I2 & I3 & I4 & I5 \\ \hline T100 & 1 & 1 & 0 & 0 & 1 \\ T200 & 0 & 1 & 1 & 1 & 0 \\ T300 & 0 & 0 & 1 & 1 & 0 \\ T400 & 1 & 1 & 1 & 1 & 0 \end{array}$$

Now, to find 1-frequent item set, remove those columns whose sum is less than minimum support i.e. 2 and those rows that sum is equal to finding frequent item set which is 1 for above case. So, the matrix after removing particular row and column would be:



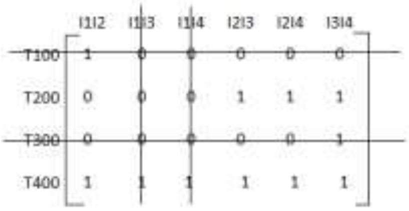
$$\begin{array}{c|ccccc} & I1 & I2 & I3 & I4 & I5 \\ \hline T100 & 1 & 1 & 0 & 0 & 1 \\ T200 & 0 & 1 & 1 & 1 & 0 \\ T300 & 0 & 0 & 1 & 1 & 0 \\ T400 & 1 & 1 & 1 & 1 & 0 \end{array}$$

So, the above matrix represents the items present in 1-freq item set. Combine the item by taking AND to get matrix of 2-freq item set, which can be represented as:



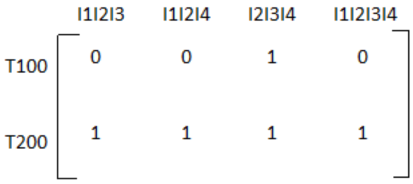
	I1I2	I1I3	I1I4	I2I3	I2I4	I3I4
T100	1	0	0	0	0	0
T200	0	0	0	1	1	1
T300	0	0	0	0	0	1
T400	1	1	1	1	1	1

Now, after removing rows and columns following the above method, the reduced matrix would be like:



	I1I2	I1I3	I1I4	I2I3	I2I4	I3I4
~~T100~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
T200	0	~~0~~	~~0~~	1	1	1
~~T300~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
T400	1	~~1~~	~~1~~	1	1	1

For finding 3-frequent set, follow the same procedure and combine item sets as follow:



	I1I2I3	I1I2I4	I2I3I4	I1I2I3I4
T100	0	0	1	0
T200	1	1	1	1

Remove those columns whose sum is less then 2(min support) and those rows whose sum is less than 3, so the reduced matrix is:So, this is the final reduced matrix for above given example. The final frequent item set (3-freq item set) is **BCD**.

IV Conclusion and Future scope

In this paper, Apriori algorithm is improved based on the properties of cutting database. The typical Apriori algorithm has performance bottleneck in the massive data processing so that we need to optimize the algorithm with variety of methods. The improved algorithm we proposed in this paper not only optimizes the algorithm of reducing the size of the candidate set of k-itemsets, but also reduce the I / O spending by cutting down transaction records in the database. The performance of Apriori algorithm is optimized so that we can mine association information from massive data faster and better. Although this improved algorithm has optimized and efficient but it has overhead to manage the new database after every generation of Matrix. So, there should be some approach which has very less number of scans of database. Another solution might be division of large database among processors.

REFERENCES

[1] “Data Mining - concepts and techniques” by Jiawei Han and MichelineKamber.

[2] “Improved Apriori Algorithm using logarithmic decoding and pruning” paper published by SuhaniNagpal, Department of Computer Science and Information Technology, Lovely Professional University, Punjab (INDIA).

[3] “Improving Efficiency of Apriori Algorithm Using Transaction Reduction” by Jaishree Singh*, Hari Ram**, Dr. J.S. Sodhi, Department of Computer Science & Engineering, Amity School of Engineering and Technology, Amity University, Sec-125 NOIDA, (U.P.),India.

[4] Wanjun Yu; Xiaochun Wang; Erkang Wang; Bowen Chen;, “The research of improved apriori algorithm for mining association rules,” Communication Technology, 2008. ICCT 2008 11th IEEE International Conference on, vol., no.,pp.513-516, 10-12 Nov. 2008.

[5] Luo Ke,Wu Jie.Apriori algorithm based on the improved techniques. Computer Engineering and application ,2001,20:20.

[6]Li Xiaohong,Shang Jin.An improvement of the new Apriori algorithm [J].Computer science, 2007,34 (4) :196-198. 2007.

[7] Sheng Chai, Jia Yang, and Yang Cheng, “The Research of Improved Apriori Algorithm for Mining A
ssociation Rules” Proceedings of the Service Systems and Servic Management ,2007.