An Approach of Improvisation in Efficiency of Apriori Algorithm

Sakshi Aggarwal[1], Ritu Sindhu[2]

[1] SGT Institute of Engineering & Technology,
Gurgaon, Haryana
sakshii.26@gmail.com
[2] SGT Institute of Engineering & Technology,
Gurgaon, Haryana
ritu.sindhu2628@gmail.com

Abstract. Association rule mining has a great importance in data mining. Apriori is the key algorithm in association rule mining. Many approaches are proposed in past to improve Apriori but the core concept of the algorithm is same i.e. support and confidence of itemsets and previous studies finds that classical Apriori is inefficient due to many scans on database. In this paper, we are proposing a method to improve Apriori algorithm efficiency by reducing the database size as well as reducing the time wasted on scanning the transactions.

Keywords: Apriori algorithm, Support, Frequent Itemset, Association rules, Candidate Item Sets.

1. INTRODUCTION

Extracting relevant information by exploitation of data is called Data Mining. There is an increasing need to extract valid and useful information by business people from large datasets [2]; here data mining achieves its goal. Thus, data mining has its importance to discover hidden patterns from huge data stored in databases, OLAP (Online Analytical Process), data warehouse etc. [5]. This is the only reason why data mining is also known as KDD (Knowledge Discovery in Databases). [4] KDD's techniques are used to extract the interesting patterns. Steps of KDD process are cleaning of data (data cleaning), selecting relevant data, transformation of data, data pre-processing, mining and pattern evaluation.

2. ASSOCIATION RULE MINING

Association rule mining has its importance in fields of artificial intelligence, information science, database and many others. Data volumes are dramatically increasing by day-to-day activities. Therefore, mining the association rules from massive data is in the interest for many industries as theses rules help in decision-making processes, market basket analysis and cross marketing etc.

Association rule problems are in discussion from 1993 and many researchers have worked on it to optimize the original algorithm such as doing random sampling, declining rules, changing storing framework etc. [1]. We find association rules from a huge amount of data to identify the relationships in items which tells about human behavior of buying set of items. There is always a particular pattern followed by humans during buying the set of items.

In data mining, unknown dependency in data is found in association rule mining and then rules between the items are found [3]. Association rule mining problem is defined as follows.

DBT = $\{T_1, T_2... T_N\}$ is a database of N T transactions.

Each transaction consists of I, where I= $\{i_1, i_2, i_3....i_N\}$ is a set of all items. An association rule is of the form $A \Rightarrow B$, where A and B are item sets, $A \subseteq I$, $B \subseteq I$, $A \cap B = \emptyset$. The whole point of an algorithm is to extract the useful information from these transactions.

For example: Consider below table containing some transactions:

Table 1. Example of transactions in a database

TID	Items
1	CPU, Monitor
2	CPU, Keyboard, Mouse, UPS
3	Monitor, Keyboard, Mouse, Motherboard
4	CPU, Monitor, Keyboard, Mouse
5	CPU, Monitor, Keyboard, Motherboard

Example of Association Rules:
{Keyboard} → {Mouse},
{CPU, Monitor} → {UPS, Motherboard},
{CPU, Mouse} → {Monitor},
A →B is an association rule (A and B are itemsets).
Example: {Monitor, Keyboard} → {Mouse}

Rule Evaluation:

Support: It is defined as rate of occurrence of an itemset in a transaction database.
Support (Keyboard → Mouse) =

$$\frac{\text{No. Of transactions containing both Keyboard and Mouse}}{\text{No. Of total transactions}}$$

Confidence: For all transactions, it defines the ratio of data items which contains Y in the items that contains X.
Confidence (Keyboard → Mouse) =

$$\frac{\text{No. Of transactions containing Keyboard and Mouse}}{\text{No. Of transactions (containing Keyboard)}}$$

Itemset: One or more items collectively is called an itemset. Example: {Monitor, Keyboard, Mouse}. K-itemset contains k-items.

Frequent Itemset: For a frequent item set:

$$S_I >= min\_sup$$

where I is an itemset, min_sup is minimum support threshold and S represent the support for an itemset.

3. CLASSICAL APRIORI ALGORITHM

Using an iterative approach, in each iteration Apriori algorithm generates candidate item-sets by using large itemsets of a previous iteration. [2]. Basic concept of this iterative approach is as follows:

Algorithm Apriori_algo(L_k)

```
1. $L_1$= {frequent-1 item-sets};
2. for (k=2; Lk-1$\neq\Phi$; k++) {
3. $C_k$= generate_Apriori($L_{k-1}$); //New candidates
4. forall transactions t $\epsilon$ D do begin
5. $C_t$=subset($C_k$,t); //Candidates contained in t
6. forall candidates c $\epsilon$ $C_t$ do
7.  c.count++;
8.  }
9.  $L_k$={c $\epsilon$ $C_k$ | c.count$\geq$minsup}
10. end for
11.  Answer=$U_k L_k$
```

Algorithm. 1. Apriori Algorithm[6]

Above algorithm is the apriori algorithm. In above, database is scanned to find frequent 1-itemsets along with the count of each item. Frequent itemset L_1 is created from candidate item set where each item satisfies minimum support. In next each iteration, set of item sets is used as a seed which is used to generate next set of large itemsets i.e candidate item sets (candidate generation) using generate_Apriori function.

L_{k-1} is input to generate_Apriori function and returns C_k. Join step joins L_{k-1} with another L_{k-1} and in prune step, item sets c ϵ C_k are deleted such that (k-1) is the subset of "c" but not in L_{k-1} of C_{k-1}.

Algorithm generate_Apriori (L_k)

```
1. insert into $C_k$
2. p =$L_{k-1}$ ,q= $L_{k-1}$
```

```
3. select $p.I_1, p.I_2, ..... p.I_{k-1}, q.I_{k-1}$ from p, q where $p.I_1 = q.I_1 ... p.I_{k-2} = q.I_{k-2}, p.I_{k-1} < q.I_{k-1}$;
4. forall itemsets c ϵ $C_k$ do
5. forall { s ⊃ (k-1) of c) do
6. if (s ∉ $L_{k-1}$) then
7. from $C_k$ , delete c
```

Algorithm. 2. Apriori-Gen Algorithm[6]

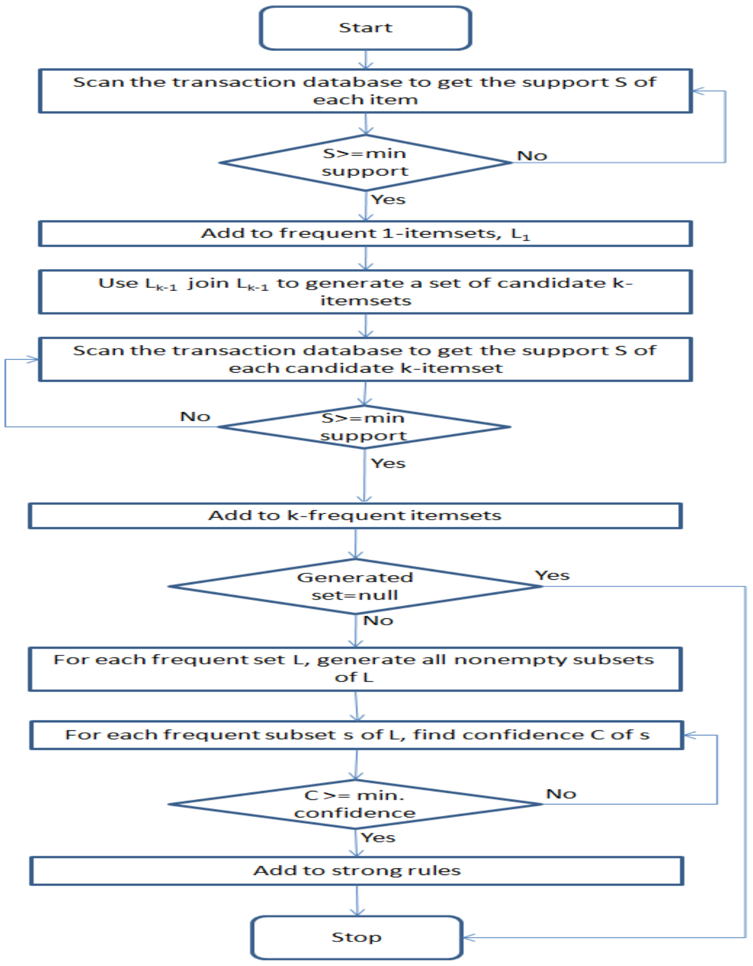
Start
Scan the transaction database to get the support S of each item
S>=min support
No
Yes
Add to frequent 1-itemsets, L_1
Use L_{k-1} join L_{k-1} to generate a set of candidate k-itemsets
Scan the transaction database to get the support S of each candidate k-itemset
No
S>=min support
Yes
Add to k-frequent itemsets
Generated set=null
Yes
No
For each frequent set L, generate all nonempty subsets of L
For each frequent subset s of L, find confidence C of s
C >= min. confidence
No
Yes
Add to strong rules
Stop

Fig.1. Apriori Algorithm Steps

3.1. Limitations of Apriori Algorithm

- Large number of candidate and frequent item sets are to be handled and results in increased cost and waste of time.
  Example: if number of frequent (k-1) items is 10^4 then almost 10^7 C_k need to be generated and tested [2]. So scanning of a database is done many times to find C_k

- Apriori is inefficient in terms of memory requirement when large numbers of transactions are in consideration.

4. PROPOSED ENHANCEMENT IN EXISTING APRIORI ALGORITHM

Below section will give an idea to improve apriori efficiency along with example and algorithm.

4.1. Improvement of Apriori

In this approach to improve apriori algorithm efficiency, we focus on reducing the time consumed for Ck generation.

In the process to find frequent item sets, first size of a transaction (S_T) is found for each transaction in DB and maintained. Now, find L_1 containing set of items, support value for each item and transaction ids containing the item. Use L_1 to generate L_2, L_3… along with decreasing the database size so that time reduces to scan the transaction from the database.

To generate $C_2(x,y)$ (items in C_k are x and y), do L(k-1) * L(k-1) . To find L_2 from C_2, instead of scanning complete database and all transactions, we remove transaction where $S_T < k$ (where k is 2, 3…) and also remove the deleted transaction from L_1 as well. This helps in reducing the time to scan the infrequent transactions from the database.

Find minimum support from x and y and get transaction ids of minimum support count item from L_1. Now, Ck is scanned for specific transactions only (obtained above) and from decreased DB size. Then, L_2 is generated by C_2 where support of C_k >= min_supp.

$C_3(x,y,z)$, L_3 and so on is generated repeating above steps until no frequent items sets can be discovered.

Algorithm Apriori
Input: transactions database, D
        Minimum support, min_sup

```
Output $L_k$: frequent itemsets in D
1. find $S_T$ //for each transaction in DB
2. $L_1$=find frequent_1_itemset (D)
3. $L_1$= find frequent_1_itemset (D)
4. $L_1$+=get_txn_ids(D)
5. for (k=2;$L_{k-1}\neq\Phi$ ; k++){
6. $C_k$=generate_candidate ($L_{k-1}$)
7. x= item_min_sup($C_k$, $L_1$) //find item from $C_k$(a,b) which has minimum support
   using $L_1$
8. target =get_txn_ids(x)  //get transactions for each item
9. foreach (txn t in tgt) do{
10. $C_k$.count++
11. $L_k$=(items in $C_k$>=min_sup)
12.} //end foreach
13. foreach(txn in D){
14. if($S_T$=(k-1))
15. txn_set+=txn
16. //end foreach
17. delete_txn_DB(txn_set)  //reduce DB size
18. delete_txn_$L_1$(txn_set,$L_1$)  //reduce transaction size in $L_1$
19.} //end for
```

Algorithm. 3. Proposed Apriori Algorithm

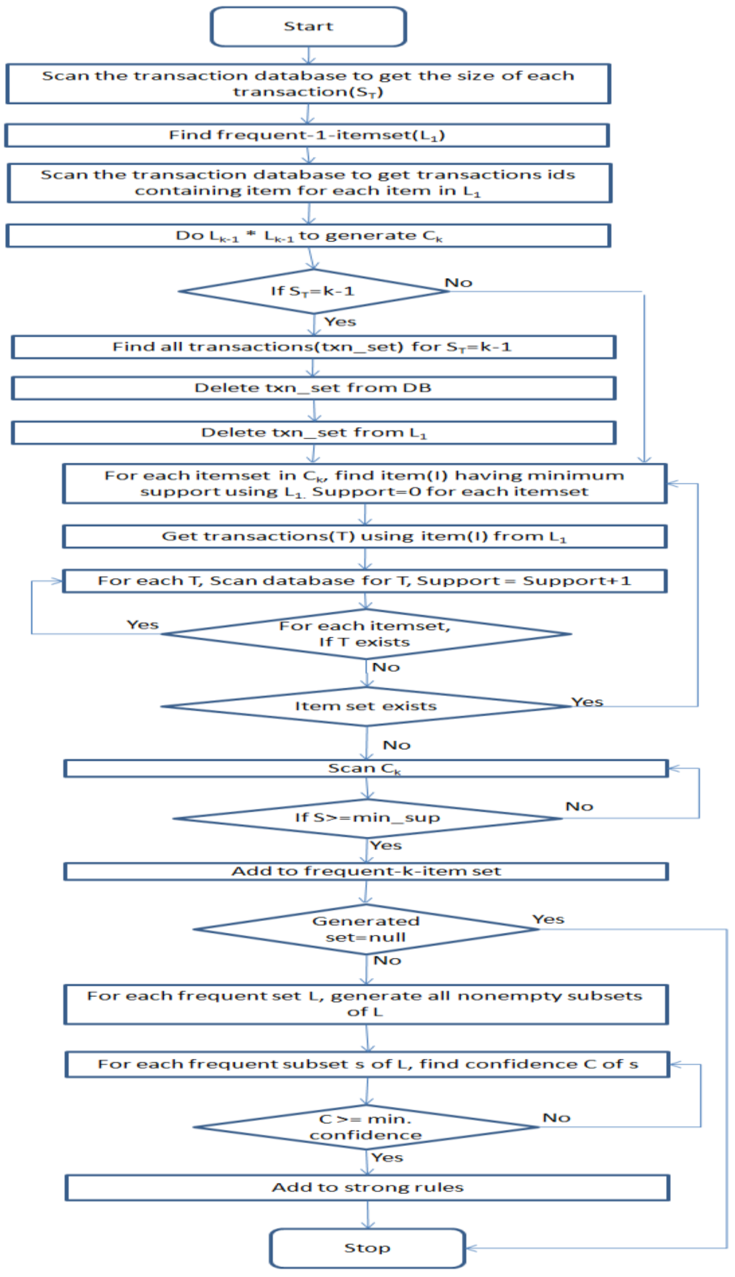
Start
Scan the transaction database to get the size of each transaction(S_T)
Find frequent-1-itemset(L_1)
Scan the transaction database to get transactions ids containing item for each item in L_1
Do L_{k-1} * L_{k-1} to generate C_k
If S_T=k-1
No
Yes
Find all transactions(txn_set) for S_T=k-1
Delete txn_set from DB
Delete txn_set from L_1
For each itemset in C_k, find item(I) having minimum support using L_1. Support=0 for each itemset
Get transactions(T) using item(I) from L_1
For each T, Scan database for T, Support = Support+1
Yes
For each itemset, If T exists
No
Item set exists
Yes
No
Scan C_k
If S>=min_sup
No
Yes
Add to frequent-k-item set
Generated set=null
Yes
No
For each frequent set L, generate all nonempty subsets of L
For each frequent subset s of L, find confidence C of s
C >= min. confidence
No
Yes
Add to strong rules
Stop

Fig.2. Proposed Apriori Algorithm Steps

5. EXPERIMENTAL EXAMPLE

Below is the transaction database (D) having 10 transactions and min_sup=3. Size of transaction (S_T) is calculated for each transaction. (Refer figure 3).

D

Transaction	Items	S_T
T_1	I_1,I_3,I_7	3
T_2	I_2,I_3,I_7	3
T_3	I_1,I_2,I_3	3
T_4	I_2,I_3	3
T_5	I_2,I_3,I_4,I_5	4
T_6	I_2,I_3	2
T_7	I_1,I_2,I_3,I_4,I_6	5
T_8	I_2,I_3,I_4,I_6	4
T_9	I_1	1
T_{10}	I_1,I_3	2

Fig.3. Transaction Database

All the transactions are scanned to get frequent-1-itemset, L_1. It contains items, respective support count and transactions from D which contain the items. Infrequent candidates' i.e. itemsets whose support < min_sup are eliminated or deleted. (Refer Figure 4 and Figure 5)

C_1

Item	Support	
I_1	5	
I_2	7	
I_3	9	
I_4	3	
~~I_5~~	~~1~~	X
~~I_6~~	~~2~~	X
~~I_7~~	~~2~~	X

Fig.4. Candidate-1-itemset

L_1

Item	Support	Transactions	
I_1	5	$T_1, T_3, T_7, T_9, T_{10}$	
I_2	7	$T_2, T_3, T_4, T_5, T_6, T_7, T_8$	
I_3	9	$T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_{10}$	
I_4	3	T_5, T_7, T_8	
~~I_5~~	~~1~~	~~T_5~~	X
~~I_6~~	~~2~~	~~T_7, T_8~~	X
~~I_7~~	~~2~~	~~T_1, T_2~~	X

Fig.5. Frequent-1-itemset

From L_1, frequent-2-itemset (L_2) is generated as follows. Example: consider itemset $\{I_1, I_2\}$. In classical apriori, all transactions are scanned to find $\{I_1, I_2\}$ in D. But in our proposed idea, firstly, transaction T_9 is deleted from D as well as from L_1 as S_T for T_9 is less than k (k=2). New D and L_1 are shown in figure 6 and figure 7 respectively. Secondly, $\{I_1, I_2\}$ is split into $\{I_1\}$ and $\{I_2\}$ and item with minimum support i.e. $\{I_1\}$ is selected using L_1 and its transactions will be used in L_2. So, $\{I_1, I_2\}$ will be searched only in transactions which contain $\{I_1\}$ i.e. T_1, T_3, T_7, T_{10}.

So, searching time is reduced twice:

- By reducing database size
- By cutting down the number of transactions to be scanned.

L_2 is shown in Figure 8.

D

Transaction	Items	S_T
T_1	I_1, I_3, I_7	3
T_2	I_2, I_3, I_7	3
T_3	I_1, I_2, I_3	3
T_4	I_2, I_3	3
T_5	I_2, I_3, I_4, I_5	4
T_6	I_2, I_3	2
T_7	I_1, I_2, I_3, I_4, I_6	5
T_8	I_2, I_3, I_4, I_6	4
~~T_9~~	~~I_1~~	~~1~~
T_{10}	I_1, I_3	2

Fig.6. Transaction Database (updated)

L_1

Item	Support	Transactions
I_1	5	$T_1,T_3,T_7,$~~T_9~~$,T_{10}$
I_2	7	$T_2,T_3,T_4,T_5,T_6,T_7,T_8$
I_3	9	$T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_{10}$
I_4	3	T_5,T_7,T_8

Fig.7. Frequent-1-itemset (updated)

L_2

Item	Support	Min	Transactions	
~~I_1,I_2~~	~~2~~	~~I_1~~	~~T_1,T_3,T_7,T_{10}~~	**X**
I_1,I_3	4	I_1	T_1,T_3,T_7,T_{10}	
~~I_1,I_4~~	~~1~~	~~I_4~~	~~T_5,T_7,T_8~~	**X**
I_2,I_3	7	I_2	$T_2,T_3,T_4,T_5,T_6,T_7,T_8$	
I_2,I_4	3	I_4	T_5,T_7,T_8	
I_3,I_4	3	I_4	T_5,T_7,T_8	

Fig.8. Frequent-2-itemset

To generate frequent-3-itemset (L_3), D is updated by deleting transactions T_6 and T_{10} as S_T for these transactions is less than k (k=3). L_1 is also updated by deleting transactions T_6 and T_{10}. Then, repeating above process, L_3 is generated and infrequent itemsets are deleted. Refer figure 9, figure 10 and figure 11 for updated database, L_1 and L_3 respectively.

D

Transaction	Items	S_T
T_1	I_1,I_3,I_7	3
T_2	I_2,I_3,I_7	3
T_3	I_1,I_2,I_3	3
T_4	I_2,I_3	3
T_5	I_2,I_3,I_4,I_5	4
~~T_6~~	~~I_2,I_3~~	~~2~~
T_7	I_1,I_2,I_3,I_4,I_6	5
T_8	I_2,I_3,I_4,I_6	4
~~T_9~~	~~I_1~~	~~1~~
~~T_{10}~~	~~I_1,I_3~~	~~2~~

Fig.9. Transaction Database (updated)

L_1

Item	Support	Transactions
I_1	5	T_1,T_3,T_7,~~T_9~~,~~T_{10}~~
I_2	7	T_2,T_3,T_4,T_5,~~T_6~~,T_7,T_8
I_3	9	T_1,T_2,T_3,T_4,T_5,~~T_6~~,T_7,T_8,~~T_{10}~~
I_4	3	T_5,T_7,T_8

Fig.10. Frequent-1-itemset (updated)

L_3

Item	Support	Min	Transactions	
~~I_1,I_2,I_3~~	~~2~~	~~I_1~~	~~T_1,T_3,T_7~~	X
~~I_1,I_3,I_4~~	~~1~~	~~I_4~~	~~T_5,T_7,T_8~~	X
I_2,I_3,I_4	3	I_4	T_5,T_7,T_8	

Fig.11. Frequent-3-itemset

So, above process is followed to find frequent-k-itemset for a given transaction database. Using frequent-k-itemset, association rules are generated from non-empty subsets which satisfy minimum confidence value.

6. COMPARATIVE ANALYSIS

We have counted the number of transactions that are scanned to find L_1, L_2 and L_3 for our given example and below figure shows the difference in count of transactions scanned by using original apriori algorithm and our proposed idea.

k	Classical Apriori	Proposed approach
1	70	70
2	60	24
3	30	9

Fig.12. Comparative Results

For k=1, number of transactions scanned is same for both classical apriori and our proposed idea but with the increase in k, count of transactions decrease. Refer below figure.

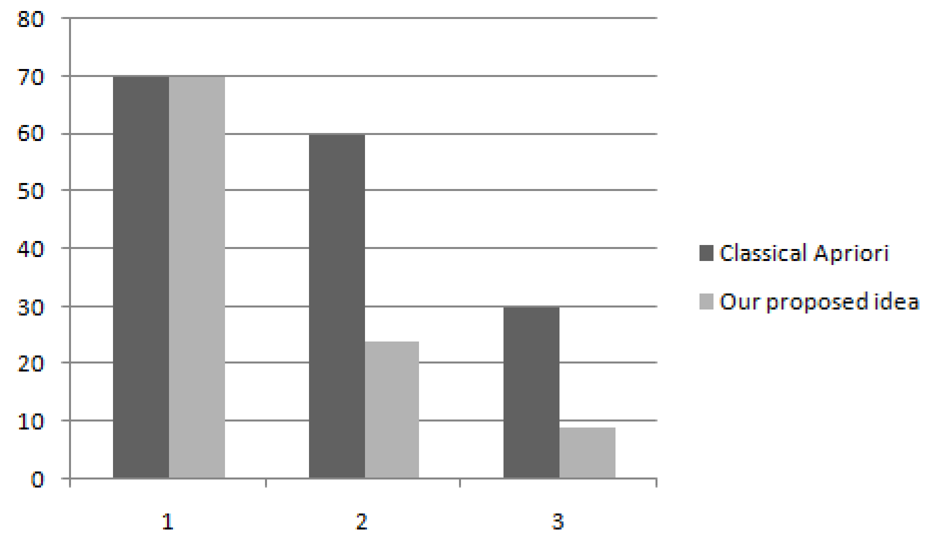

Fig.13. Comparative Analysis

7. CONCLUSION

We have proposed an idea to improve the efficiency of apriori algorithm by reducing the time taken to scan database transactions. We find that with increase in value of k, number of transactions scanned decreases and thus, time consumed also decreases in comparison to classical apriori algorithm. Because of this, time taken to generate candidate item sets in our idea also decreases in comparison to classical apriori.

REFERENCES

1. J. Han and M. Kamber, Conception and Technology of Data Mining, Beijing: China Machine Press, 2007.
2. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From data mining to knowledge discovery in databases,” vol. 17, no. 3, AI magazine, 1996, pp. 37.
3. S. Rao, R. Gupta, “Implementing Improved Algorithm over APRIORI Data Mining Association Rule Algorithm”, International Journal of Computer Science And Technology, pp. 489-493, Mar. 2012
4. H. H. O. Nasereddin, “Stream data mining,” International Journal of Web Applications, vol. 1, no. 4,pp. 183–190, 2009.

5. M. Halkidi, “Quality assessment and uncertainty handling in data mining process,” in Proc, EDBT Conference, Konstanz, Germany, 2000.
6. Rakesh Agarwal, Ramakrishna Srikant, “Fast Algorithm for mining association rules” VLDB Conference Santiago, Chile, 1994, pp 487-499.