Middle-East Journal of Scientific Research 23 (7): 1475-1481, 2015
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2015.23.07.22260

Transaction Reduction in Apriori Algorithm

[1]*Sakshi Aggarwal and* [2]*Ritu Sindhu*

[1]SGT Institute of Engineering & Technology, Gurgaon, Haryana
[2]SGT Institute of Engineering & Technology, India Gurgaon, Haryana, India

Abstract: Association rule mining is used to find frequent item sets 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 Apriority algorithm efficiency by reducing the database size as well as reducing the time wasted on scanning the transactions.

Key words: Apriori algorithm · Support · Frequent Itemset · Association rules · Candidate Item Sets

INTRODUCTION

Mining association rules, as one of the several data mining tasks, have a big share in the data mining research. This is attributed to its wide area of applications. Applications of association rule mining span a wide area of business from market basket analysis, to analysis of promotions and catalog design and from designing store layout to customer segmentation based on buying patterns. Other applications include health insurance, fraudulent discovery and loss-leader analysis [5, 7].

Mining association rules has the same challenges facing data mining in general. Several algorithms are proposed to mine the association rules from the data and of these algorithms, Apriori algorithm is used the most. Apriori algorithm has been successful in finding 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.

So, the main challenge in Apriori algorithm lies to improve the efficiency of the algorithm by controlling number of database scans and thus control the I/O cost.

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. [4]. 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 [5]. Association rule mining problem is defined as follows.

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

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

Corresponding Author: Sakshi Aggarwal, SGT Institute of Engineering & Technology, Gurgaon, Haryana, India.

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.

$$\text{Support (Keyboard \_ Mouse)} = \frac{\text{No. of transactions containing both Keyboard and Mouse}}{\text{No. of total transactions}}$$

Confidence: For all transactions, it defines the ratio of data items which contains Y in the items that contains X.

$$\text{Confidence (Keyboard \_ Mouse)} = \frac{\text{No. of transactions containing Keyboard and Mouse}}{\text{No. of transactions (containing Keyboard)}}$$

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

Frequent Item set: 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.

Classical Apriori Algorithm: Using an iterative approach, in each iteration Apriori algorithm generates candidate item-sets by using large itemsets of a previous iteration. [6]. Basic concept of this iterative approach is as follows:

Algorithm Apriori_Algo(L_k):

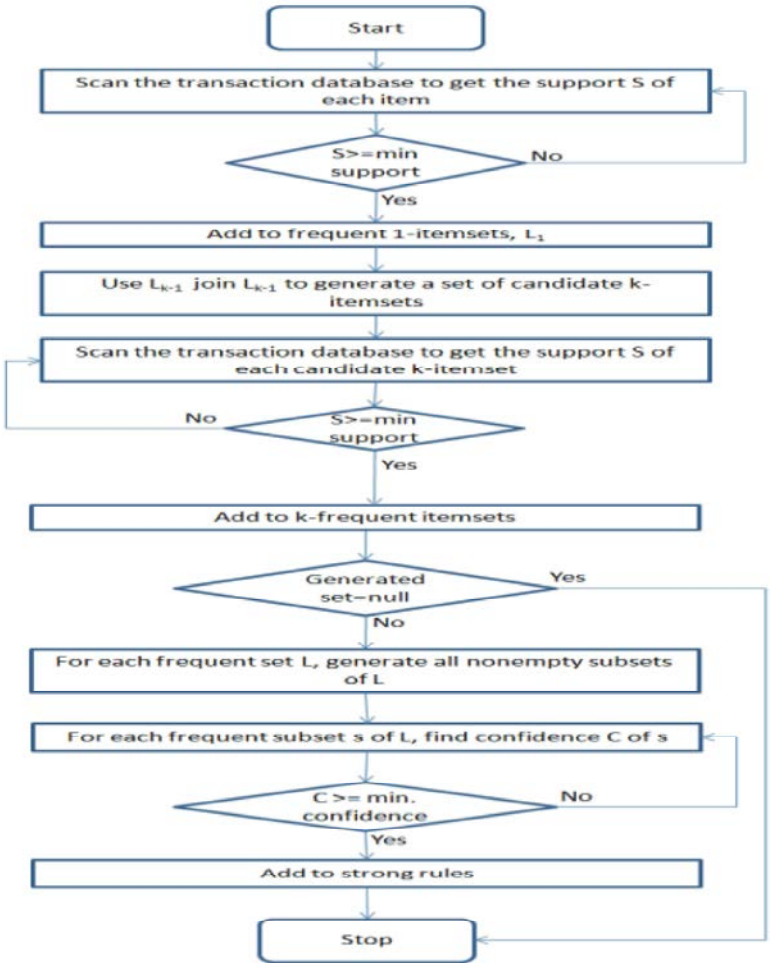
- L_1= {frequent-1 item-sets};
- for (k=2; Lk-1 ≠ Φ; k++) {
- C_k= generate_Apriori(L_{k-1}); //New candidates
- forall transactions t ∈ D do begin
- C_t=subset(C_k,t); //Candidates contained in t
- forall candidates c ∈ C_t do
- c.count++;
- }
- L_k={c ∈ C_k | c.count=minsup}
- end for
- Answer = $U_k L_k$

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 ∈ 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):

- insert into C_k
- p =L_{k-1}, q= L_{k-1}
- select $p.I_1$,$p.I_2$,.....$p.I_{k-1}$,$q.I_{k-1}$ from p, q where $p.I_1$ = $q.I_1$...$p.I_{k-2}$= $q.I_{k-2}$,$p.I_{k-1}$<$q.I_{k-1}$;
- forall itemsets c ∈ C_k do
- forall {s ⊃ (k-1) of c) do
- if (s ∉ L_{k-1}) then
- from C_k, delete c



Apriori Algorithm Steps

Algorithm 2: Apriori-Gen Algorithm [6].

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. 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.

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

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.

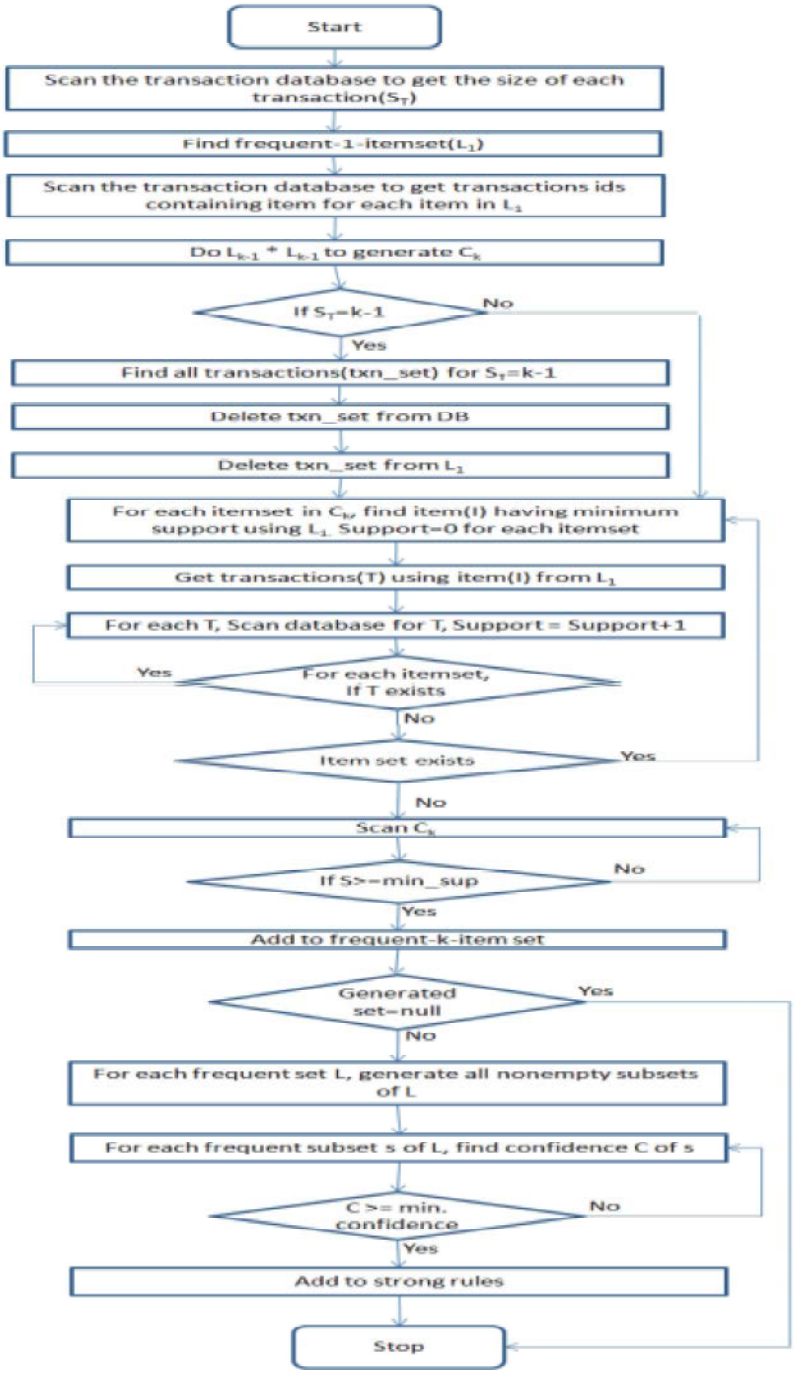
Find minimum support from x and y and get transaction ids of minimum support count item from L_1.

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



Proposed Apriori Algorithm Steps

- find S_T //for each transaction in DB
- L_1=find frequent_1_itemset (D)
- L_1= find frequent_1_itemset (D)
- L_1+=get_txn_ids(D)
- for (k=2;$L_{k-1} \neq \Phi$; k++){
- C_k=generate_candidate (L_{k-1})
- x= item_min_sup(C_k, L_1) //find item from C_k(a,b) which has minimum support using L_1
- target =get_txn_ids(x) //get transactions for each item
- foreach (txn t in tgt) do{
- C_k.count++
- L_k=(items in C_k>=min_sup)
- } //end foreach
- foreach(txn in D){
- if(S_T=(k-1))
- txn_set+=txn
- //end foreach
- delete_txn_DB(txn_set) //reduce DB size
- delete_txn_L_1(txn_set,L_1) //reduce transaction size in L_1
- } //end for

Algorithm 3: Proposed Apriori Algorithm

Experimental Example: Proposed algorithm example is given below. A database of 10 transactions is considered and size of each transaction is calculated and stored as ST. In first iteration, C_1 is generated by simple scanning the database to count the number of transactions of each itemset. Min_sup for this example is set as 4. So to generate L_1, itemsets having transaction count equal to or greater than 3 are considered as frequent and included in L_1.

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.

Next step is used to generate C_2 and L_2 i.e k=2. Transaction T_9 is deleted from database as its SOT=k-1. Resultant DB is D_1. L_1 is self joined with L_1 to generate itemsets in C_2. Transactions in D_1 are scanned and support count for C_2 is determined. L_2 consists of those itemsets which satisfy minimum support value. C_2 and L_2 are shown in figure 5.

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.

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

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

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

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

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

Fig. 12: Comparative Results

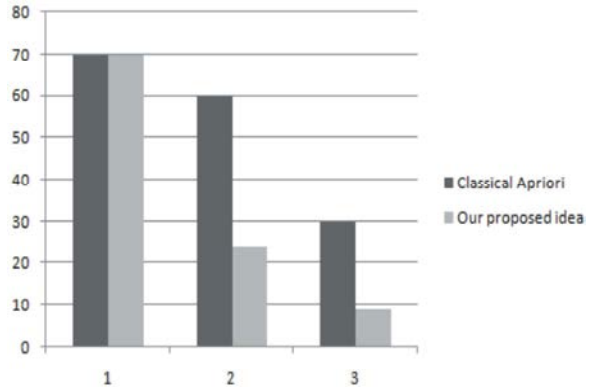


Fig. 13: Comparative Analysis

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.

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 Figure 12, 13.

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. Han, J. and M. Kamber, 2007. Conception and Technology of Data Mining, Beijing: China Machine Press.
2. Fayyad, U., G. Piatetsky-Shapiro and P. Smyth, 1996. From data mining to knowledge discovery in databases, AI Magazine, 17(3): 37.
3. Rao, S. and R. Gupta, 2012. Implementing Improved Algorithm over APRIORI Data Mining Association Rule Algorithm, International Journal of Computer Science And Technology, pp: 489-493.
4. Nasereddin, H.H.O., 2009. Stream data mining, International Journal of Web Applications, 1(4): 183-190.
5. Halkidi, M., 2000. Quality assessment and uncertainty handling in data mining process, in Proc, EDBT Conference, Konstanz, Germany.
6. Agarwal Rakesh and Ramakrishna Srikant, 1994. Fast Algorithm for mining association rules,VLDB Conference Santiago, Chile, pp: 487-499.