Analysis of Traditional and Enhanced Apriori Algorithms in Association Rule Mining

Logeswari T, Ph.D
Assistant Professor
Department of Computer Applications
Dr.N.G.P.Institute of Technology
Coimbatore, India

Valarmathi N
Department of Computer Applications
Dr.N.G.P.Institute of Technology
Coimbatore, India

Sangeetha A
Department of Computer Applications
Dr.N.G.P.Institute of Technology
Coimbatore, India

Masilamani M
Department of Computer Applications
Dr.N.G.P.Institute of Technology
Coimbatore, India

ABSTRACT
In this paper, Enhanced Apriori Algorithm is proposed which takes less scanning time. It is achieved by eliminating the redundant generation of sub-items during pruning the candidate item sets. Both Traditional and Enhanced Apriori algorithms are compared and analysed in this paper.

Keywords
Candidate generation; frequent itemsets; transaction_size; support count; threshold.

1. INTRODUCTION
Data mining is known as discovering knowledge from the database. From the abstracted patterns, decision-making can be done easily. Many of the problems will be predicted by taking some decisions using data mining. [1]

Association rule is mainly based on discovering frequent item sets. Association rules are frequently used by retail stores to assist in advertising, marketing, inventory control, predicting faults in telecommunication network [2].

Traditional Apriori algorithm represents the candidate generation approach. It generates candidate (k+1) itemsets based on frequent k-itemsets[3].

Enhanced Apriori algorithm reduces the scanning time by cutting down unnecessary transaction records.

2. TRADITIONAL APRIORI ALGORITHM
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 L1. 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 important property called the Apriori property, presented is used to reduce the search space[4].

2.1 Apriori property
All nonempty subsets of a frequent itemsets must also be frequent. A two-step process is used to find the frequent itemsets: join and prune actions.

2.1.1 The join step
To find L_k a set of candidate k-itemsets is generated by joining L_k-1 with itself. This set of candidates is denoted C_k [5].

2.1.2 The prune step
The members of C_k may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k).[6] To reduce the size of C_k, 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 a candidate k-itemset is not in L_k-1, then the candidate cannot be frequent either and so can be removed from C_k[7].

2.2 Description of the Traditional Apriori algorithm
Input

D, Database of transactions, min_sup, min_support threshold.

Output*:*

L, frequent itemsets in D

Method:

(1) L1=find_frequent_1-itemsets(D);

(2) for(k=2; L_k-1≠Φ; k++){

(3) C_k=apriori_gen(L_k-1, min_sup);

(4) for each transaction t∈D{

(5) Ct=subset(C_k,t);

(6) for each candidate c∈Ct

(7) c.count++;

(8) }

(9) L_k={ c∈C_k |c.count≥min_sup }

(10) }

(11) return L=U_kL_k ;

Procedure apriori_gen (L_k-1:frequent(k-1)-itemsets)
(1) for each itemset l1∈ L_k-1{

(2) for each itemset l2∈ L_k-1{

(3) if(l1 [1]= l2 [1])∧ (l1 [2]= l2 [2]) ∧…∧(l1 [k-2] = l2 [k-2]) ∧(l1 [k-1]< l2 [k-1]) then {

(4) c=l1∞l2;

(5) if has_infrequent_subset(c, L_k-1) then

(6) delete c;

(7) else add c to C_k ;

(8) }}}

(9) return C_k;

Procedure has_infrequent_subset

(c: candidate k-itemset; Lk-1:frequent(k-1)-itemsets)

(1) for each(k-1)-subset s of c {

(2) if s ∉ L_k-1 then

(3) return true; }

(4) return false;

2.3 Example

Let's look at a concrete example, based on the textile transaction database, D, of Table 1.1. There are nine transactions in this database, that is, |D| = 9. We illustrate the Traditional Apriori Algorithm using following steps.

Table I. Experimental data

TID	List of item_IDs
T_{100}	a,b,e
T_{200}	b,d
T_{300}	b,c
T_{400}	a,b,d
T_{500}	a,c
T_{600}	b,c
T_{700}	a,b,c,d
T_{800}	a,b,c,e
T_{900}	a,b,c

2.4 Generation of Frequent Itemsets Using Traditional Apriori Algorithm

Following steps explain the generation of candidate item set and frequent item set for the above transaction table 1.1.where minimum support count is 2.

Step 1: Scan D for count of each candidate

Itemset	Sup.count
[a]	6
[b]	8
[c]	6
[d]	3
[e]	2

Step 2: Compare candidate support count with minimum support count

Itemset	Sup.count
[a]	6
[b]	8
[c]	6
[d]	3
[e]	2

Step 3: Generate C2 candidate from L1

Itemset
[a , b]
[a , c]
[a , d]
[a , e]
[b , c]
[b , d]
[b , e]
[c , d]
[c , e]
[d , e]

Step 4: Scan D for count of each candidate

Itemset	Sup.count
[a,b]	5
[a,c]	4
[a,d]	2
[a,e]	2
[b,c]	4
[b,d]	3
[b,e]	2
[c,d]	0
[c,e]	1
[d,e]	0

Step 5: Compare candidate support count with minimum support count

Itemset	Sup.count
[a,b]	5
[a,c]	4
[a,d]	2
[a,e]	2
[b,c]	4
[b,d]	2
[b,e]	2

Step 6: Generate C3 candidate from L2

Itemset
[a,b,c]
[a,b,e]
[a,b,d]

Step 7: Scan D for count of each candidate

Itemset	Sup.count
[a,b,c]	3
[a,b,e]	2
[a,b,d]	2

Step 8: Compare candidate support count with minimum support count

Itemset	Sup.count
[a,b,c]	3

3. ENHANCED APRIORI ALGORITHM

Traditional Apriori algorithm generates large number of candidate sets for large database. So it consumes more cost[8]. To avoid this, Enhanced Apriori Algorithm is introduced which reduces the size of the database. In this proposed system variable Transaction_Size(TS) is introduced which contains the number of items in individual transaction. If the Transaction_Size is less than threshold value, that transaction alone will be deleted from the database[9].

3.1 Description of the algorithm

Input

D: Database of transactions; min_sup: minimum support threshold

Output

L: frequent itemsets in D

Method

1) L1=find_frequent_1-itemsets(D);

2) For(k=2;Lk-1≠∅; k++){

3) Ck=apriori_gen(Lk-1, min_sup);

4) for each transaction t∈D{

5) Ct=subset(Ck,t);

6) for each candidate c∈Ct

7) c.count++;

8) }

9) Lk={ c∈Ck |c.count≥min_sup };

10) if(k>=2){

11) delete_datavalue(D, Lk, Lk-1);

12) delete_datarow (D, Lk); }

13) }

14) return L=UkLk ;

Procedure apriori_gen(Lk-1:frequent(k-1)-itemsets)

1) for each itemset l1∈ Lk-1 {

2) for each itemset l2∈ Lk-1 {

3) if(l1 [1]= l2 [1])∧ (l1 [2]= l2 [2]) ∧…∧(l1 [k-2]= l [k-2]) ∧(l1 [k-1]< l2 [k-1]) then {

4) c=l1 ∞l2;

5) for each itemset l1∈Lk-1 {

6) for each candidate c ∈Ck {

7) if l1 is the subset of c then

8) c.num++; }}}}}

9) C'k={ c∈Ck |c.num=k};

10) return C'k;

Procedure delete_datavalue (D:Database; Lk: frequent (k)-itemsets; Lk-1: frequent(k-1) - itemsets)

1) for each itemset i ∈Lk-1 and i ∉ Lk{

2) for each transaction t∈D{

3) for each datavalue∈t{

4) if (datavalue=i)

5) update datavalue=null;

6) }}

Procedure delete_datarow
(D: Database; Lk:frequent(k) - itemsets)

1) for each transaction t∈D{
2) for each datavalue∈t{
3) if(datavalue!=null and datavalue!=0){
4) datarow.count++; }}
5) if(datarow.count<k){
6) delete datarow;}
7)**}**

3.2 Example of Algorithm

Following steps show the generation of frequent itemset for table 1.1 using Enhanced Apriori Algorithm.

3.2.1 Generation Of Frequent Itemset Using Enhanced Apriori Algorithm

Step 1

D1

TID	List of item_IDS	TS
T_{100}	a,b,e	3
T_{200}	b,d	2
T_{300}	b,c	2
T_{400}	a,b,d	3
T_{500}	a,c	2
T_{600}	b,c	2
T_{700}	a,b,c,d	2
T_{800}	a,b,c,e	4
T_{900}	a,b,c	3

C1

Itemset	Sup.count
[a]	6
[b]	8
[c]	6
[d]	3
[e]	2

L1

Itemset	Sup.count
[a]	6
[b]	7
[c]	6

Step 2

D2

TID	List of item_IDS	TS
T_{100}	a,b	2
T_{200}	B	1
T_{300}	b,c	2
T_{400}	a,b	2
T_{500}	a,c	2
T_{600}	b,c	2
T_{700}	a,b,c	3
T_{800}	a,b,c	3
T_{900}	a,b,c	3

C2

Itemset	Sup.count
[a,b]	5
[a,c]	3
[b,c]	5

L2

Itemset	Sup.count
[a,b]	5
[b,c]	5

Step 3:

D3

TID	List of item_IDS	TS
T_{100}	a,b	2
T_{300}	b,c	2
T_{400}	a,b	2
T_{600}	b,c	2
T_{700}	a,b,c	3
T_{800}	a,b,c	3
T_{900}	a,b,c	3

C3

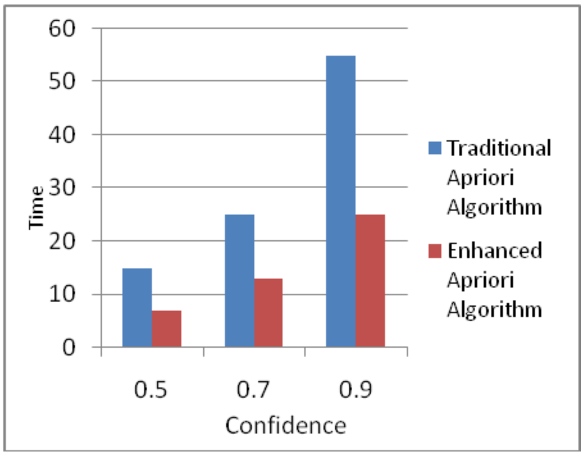
Itemset	Sup.count
[a,b]	4
[b,c]	4

L3

Itemset	Sup.count
[a,b,c]	2

4. COMPARISION BETWEEN APRIORI AND ENHANCED APRIORI ALGORITHM

We have performed the comparison of Apriori and Enhanced Apriori algorithms for different set of instances and confidence. This comparison is shown in the below graph.



The above Fig.1 shows that the time taken to execute the Enhanced Apriori Algorithm is less compared with Apriori for any Confidence level. Thus the performance of Enhanced Apriori Algorithm is an efficient and scalable method for mining the complete set of frequent patterns[10].

5. ACKNOWLEDGMENT

In this paper, Enhanced Apriori algorithm reduces the I / O spending time by cutting down unwanted transaction records in the database. The efficiency of Apriori algorithm is optimized so that mining of association information from massive data will be faster and better.

The performance analysis is done by varying number of instances and confidence level. The efficiency of both algorithms is evaluated based on time to generate the association rules. From the above analysis it can be concluded that the Enhanced Apriori Algorithm behaves better than the Traditional Apriori Algorithm.

6. REFERENCES

[1] Margret H. Dunham, S.Sridhar, "Data mining Introductory and advanced topics", Pearson Education,Second Edition,2007.

[2] Agrawal, R, Srikant, R, 1994, 'Fast algorithms for mining association rules in large databases', Proc. of 20th Int'l conf. on VLDB: 487-499.

[3] C. Gyorodi, R. Gyorodi. "Mining Association Rules in Large Databases". Proc. of Oradea EMES'02: 45-50, Oradea, Romania, 2002.

[4] M., Suraj Kumar Sudhanshu, Ayush Kumar and Ghose M.K., "Optimized association rule mining using genetic algorithm Anandhavalli Advances in Information Mining" , ISSN: 0975–3265, Volume 1, Issue 2, 2009, pp-01-04

[5] Han, J, Pei, J, Yin, Y 2000, 'Mining Frequent Patterns without Candidate Generation', Proc. of ACM-SIGMOD.

[6] Daniel Hunyadi, "Performance comparison of Apriori and FP-Growth algorithms in generating association rules", Proceedings of the European Computing Conference.

[7] Goswami D.N. et. al. "An Algorithm for Frequent Pattern Mining Based On Apriori", (IJCSE) International Journal on Computerm Science and Engineering Vol. 02, No. 04, 2010, 942-947

[8] Ms Shweta and Dr. Kanwal Garg "Mining Efficient Association Rules Through Apriori Algorithm Using Attributes and Comparative Analysis of Various Association Rule Algorithms", International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 6, June2013.

[9] Pulari.s.s et al, "Understanding Rule Behavior through Apriori Algorithm over Social Network Data", Global Journal of Computer Science and Technology Volume 12 Issue 10 Version 1.0 May 2012.

[10] "Fast Algorithms for Mining Association Rules", IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.