International Journal of Computer Applications (0975 – 8887)
Volume 137 – No.9, March 2016

A Novel Association Rule Algorithm to Discover Maximal Frequent Item Set

Hartej Singh
MTech
NIT Jalandhar
Punjab,India

Vinay Dwivedi
MTech
NIT Jalandhar
Punjab,India

ABSTRACT

Association Rule mining is a sub-discipline of data mining. Apriori algorithm is one of the most popular association rule mining technique. Apriori technique has a disadvantage that before generating a maximal frequent set it generates all possible proper subsets of maximal set. Therefore it is very slow as it requires many database scans before generating a maximal frequent itemset In the method proposed in this paper entire database is scanned only once. Frequency count of all distinct transactions is stored in a hash map. Algorithm maintains an array of tables such that each table in the array contain frequency count of all potential k-itemsets..Binary search and the concept of longest common subsequence are used to efficiently extract maximal frequent itemset. Experimental results show that proposed algorithm performs better than apriori algorithm.

Keywords

Association rule mining ,Apriori algorithm, Frequent itemset, Hashing, Longest common subsequence

1. INTRODUCTION

DATA mining has appeared as a significant research field focusing mainly on knowledge discovery from the databases. Data sets from the databases are mined so as to generate information that can be used effectively in various domains. The objective of data mining is prediction and description [1]. Association Rule mining is a sub-discipline of data mining. Generic procedure in any associative rule mining procedure is to first identify the frequent item-set in the database on the basis of minimum support and then build the association rules from the identified frequent item-set with specific minimum confidence. The association rule relates the occurrence of item A with the occurrence of item B. This type of mining is more applicable and very useful in the market basket analysis [2].

2. RELATED WORKS

Apriori algorithm was proposed by R.Agrawal and R.Srikant in 1994[2]. It uses an iterative procedure to evaluate the specific length of item collection in the given database to generate frequent item sets. It reduces the count of candidate item sets using the principle that all subsets of frequent item sets are frequent too.

Han et al. [3] proposed FP – Tree algorithm which scans the database only two times without any iteration for candidate sets generation. In the first scan FP Tree is constructed in the next scan using a procedure known as FP growth frequent patterns are extracted from FP tree.

Christian Hidber in [4] proposed CARMA - *continuous Association rule mining algorithm.* CARMA performs the scan of transaction database twice. In phase I it generates a lattice of large itemsets and in phase II it prunes all itemsets with support lesser than threshold.

Another associative rule mining algorithm *Rapid association rule mining algorithm RARM* [5] uses a tree like data structure to represent a database. Preprocessing in RARM is done using trie Item set TrieIT. RARM generates 1-itemset and 2-itemset faster using Support Oriented Trie Itemset (SOTrieIT) structure.

DHP algorithm [6] uses Hash technique in order to avoid the scanning of entire transactional database every time when we need to scan for a subset of item-sets and Reduction technique of event database in order to improve the efficiency of the Apriori algorithm.

3. THE PROPOSED METHOD

Classical apriori algorithm begins with item set containing a single element and gradually increases size of the item set to obtain maximal frequent item set.

In the proposed approach the idea of set table is introduced. A set table BT_k contains all possible itemsets of length *k* where 1<=*k*<=n. Set tables { $BT_1.......BT_n$ } are kept in an array.

For n items, there can exist at most 2^n distinct transactions in the transaction database. Frequencies of all distinct transaction are stored in a hash map. To figure out whether an itemset in the set table is frequent or not we used the concept of longest common subsequence [8].

A longest common subsequence (LCS) is the the longest subsequence common to all sequences in a set of sequences [8]. The complexity of finding a longest common subsequence between two sequences of lengths m and n is O(m*n). Using LCS technique improves the performance of proposed methodology over classical Apriori, as we do not need to generate all possible subsets of each transaction in the database and moreover instead of scanning the whole transaction database the proposed algorithm only scans 2^n records stored in a Hash table. Thus this approach is very effective in cases, where the value 2^n is less than number of transactions. The detailed strategy proposed is discussed in this section:

Step1: Create set tables

If there are *n* distinct items in the database, an array of size *n* is constructed and every element in this array contains a table. A set table B_k contains all item sets of length k. An entry in a set table contains two attributes. The first attribute holds an itemset and the second attibute holds an integer value representing frequency count of that itemset. Frequency count of all itemsets in set table is set to zero initially . Array of set tables is created in following three steps.

1) An array of size *n* is created to hold *n* set tables. Where *n* is count of distinct items present in database. Map Data structure is used for this purpose as every entry in set table has two attributes.
2) From the given list of items in the database all possible itemsets are generated and an itemset of length *i* is assigned to an array element B[i].

Frequency of all item sets is initialized to zero.

For example all itemsets of length one is allocated to set table contained at array index one i.e. BT[1], and their frequency count is initialized to zero. For item list having items *{p,q,r,s}* our set tables are shown below.

Table 1: Set Tables

p	0	pq	0	pqr	0	pqrs	0
q	0	pr	0	pqs	0		
r	0	ps	0	prs	0		
s	0	qr	0	qrs	0		
		qs	0				
		rs	0				

BT[1] BT[2] BT[3] BT[4]

Step2: Build transaction table

The transaction table as shown in table 2 contains the frequency count of various transactions.
Building transaction table can be described as follows.

1) A textual database is used for building transaction table. In this textual database, items in a itemset are kept in a lexicographical order. i.e. if a transaction contains items "yzx" then the transaction is kept in the textual database as "xyz".
2) Every entry in transactional database is traversed in order to create a table holding the frequency count of each transaction.

Suppose the transactional database contains following transactions {pq, prs, qs, pq, rs, pqs, prs, qs}. Transaction table built using given transactions is shown in Table2.

Table 2: Transaction Table

pq	2
qs	2
rs	1
prs	2
pqs	1

Step 3: Binary search

Apriori builds maximal frequent set by generating frequent sets in a bottom-up fashion. In the proposed algorithm however the frequent set formed at any stage is determined by a binary search procedure which is used on the array containing set tables. Lower limit *low* in binary search is initialized to 1 while upper limit *upper* is initialized to *n* count of items in itemlist. A variable *mid* is used to hold middle value between lower and upper limit. i.e. *mid*=(*low*+*upper*)/2. The variable *max* in the end will contain length of maximal frequent itemset

```
Low=1, upper=n
While low < upper
Mid= (upper+lower)/2
if (check(mid))
     Low=mid+1;
    If(mid>max)
   Max=mid;
 else
     Upper=mid-1;
```

Maximal frequent itemset can be found at table *BT[max]* .

To check whether a table at *B[mid]* has a frequent itemset following procedure is adopted.

BT[mid,i] denotes i^{th} itemset in set table at *BT[mid]*

BT[mid,i,f] denotes frequency count of i^{th} itemset in table *BT[mid]*.

TT[t] is the t^{th} transaction in transaction table

TT[t,f] is frequency count t^{th} transaction in transaction table.

|*s*| denotes length of a string. For e.g.|abcd|=4.

T is size transaction table TT

```
Boolean CHECK (mid )
{
k=mid;
For i=1 to C(n,k)
{
For t=1 to T
 {
   If (|LCS(BT[mid,i],TT[t])| = = |BT[mid,i]|)
      BT[mid,i,f] += TT[t,f]
      If(BT[mid,i,f] >=support )

  Return true
}
 Return false
}
```

Procedure *CHECK* returns a *true* if frequency count of any of the itemsets at *BT[mid]* is found greater than or equal to support value. To calculate the frequency count of ith itemset *BT[mid,i]*, the length of Longest common subsequence (LCS) between it and every transaction in TT is evaluated one by one and if the length of longest common subsequence is equal to the length of itemset itself frequency count is incremented by the frequency count of transaction. The procedure is explained with the help of an example.

Binary search is implemented as follows:-

1) Suppose the item list contains four elements {a,b,c,d} .

2) Therefore initially *low*=1, *upper* =4 and *mid* = 2, so procedure *check ( )* with *mid*=2 is called to check if BT [2] has any frequent itemset.

Table 3: BT[2]

Itemset	Frequency
pq	0
pr	0
ps	0
qr	0
qs	0
cs	0

3) Now Procedure *CHECK* computes length Longest common subsequence (LCS) between itemsets in table BT[2] and every element in transaction table TT. The first element of BT [2] is BT[2,1] and for our example it is "ab". Hence here |BT[2,1]| is 2 . First iteration in the procedure make following calculations.

|LCS(BT[2,1]),TT[1])|=2 (1)
|LCS(BT[2,1]),TT[2])|=1 (2)
|LCS(BT[2,1]),TT[3])|=0 (3)
|LCS(BT[2,1]),TT[4])|=1 (4)
|LCS(BT[2,1]),TT[5])|=2 (5)

The condition *if(|LCS(BT[mid,i],TT[j])| = = |BT[mid,i]|)* is true for (1) and (5)

Therefore BT[mid,i,f] + = TT[j,f] is executed for t=1 and t=5.

Frequencies of transaction at TT[1] and TT[5] are added to the frequency count of BT[2,1] in set table. Hence we have BT[2,1,f]=BT[2,1,f]+TT[1,f]+TT[5,f]

BT[2,1,f]=0+2+1=3

Which is equal to the value of support hence BT[2,1] is a frequent itemset and the procedure *CHECK* terminates by returning *true*.

For our example BT[2] gets modified as shown below :-

Table 4: Modified Set Table

pq	3
pr	0
ps	0
qr	0
qs	0
rs	0

As frequency count of "pq" and "pqs" in transaction table is 2 and 1 respectively. These frequency counts are added to frequency count of "pq" in set table.

4) The procedure above is repeated for all elements in the set table at BT [*mid*] till an element in set table whose frequency count is more than specified support is not found. As an element in set table whose frequency is more than specified support is found, next iteration of binary search is executed, and the set table is scanned at array index given by new value of *mid*.

5) *Apriori property* states that any subset of frequent itemset is frequent. Hence if any frequent itemset is not found at BT[*mid*] then a maximal frequent itemset would definitely be of length lesser than *mid.* Binary search is hence resumed to search the lower half of the array i.e. BT[*low…..mid-1*] . And conversely if a frequent itemset is found at BT[*mid*] then maximal frequent itemset has length greater than or equal to *mid*. Binary search now searches maximal frequent set in upper half of the array BT i.e. BT[*mid+1…..upper*]. For example if our item set contains ten elements then our binary search operation has *low*=1 and *upper*=10. Now initial value of *mid* is 5 if we find any frequent itemset at set table BT[5] we resume binary search with value of mid is 8(*low*=6,*upper*=10) but if we do not find any such itemset in table then we resume binary search with value of *mid* equal to 2(*low*=1,*upper*=4).

6) Using Binary search reduces run time complexity of apriori .

4. EXPERIMENTAL RESULTS

To study the performance of our algorithm we have implemented it using c++ programming language. The programme was executed on an intel core i5 machine. We used a large data set consisting of 1 million transactions and item set consists of 15 items. The experimental study compares the time consumed to find maximal frequent itemset in Apriori and proposed algorithm for various values of minimum support. The results are shown in Fig.1 and Fig 2. The results show that proposed algorithm takes lesser time to discover maximal frequent itemset as compare to Apriori algorithm.

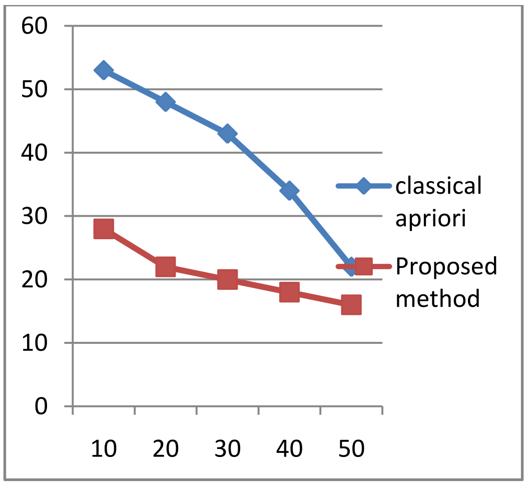

Fig : 1 Time (Y-axis) (secs) vs Support (X-axis)

Table 5: Comparison of time taken (in secs) and time reduction rate on various values of minimum support

Support	Classical Apriori	Proposed Method	Reduction Rate
10	53	28	47.16%
20	48	22	54.16%
30	43	20	53.48%
40	34	18	47.05%
50	22	15	31.81%

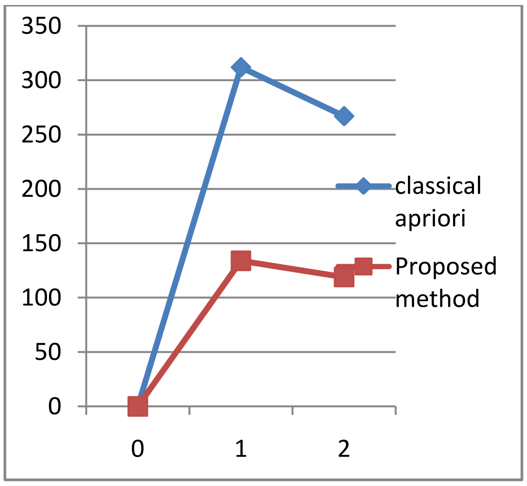

Fig : 2 Time (Y-axis) (secs) vs Support (X-axis)

Table 6: Comparison of time taken (in secs) and time reduction rate on various values of minimum support

Support	Classical apriori	Proposed method	Reduction rate
0	0	0	0%
1	312	134	57.05%
2	267	119	55.43%

5. CONCLUSIONS

This paper presented an associative rule mining algorithm for generating maximal frequent itemset. Experimental study shows that proposed algorithm performs better than Apriori algorithm by taking lesser time to compute maximal frequent itemset. Apriori algorithm applies iterative procedures to search frequent item-sets therefore it is very slow as it requires many database scans before generating a maximal frequent itemset. In the proposed algorithm database is scanned only once to create a transaction table having 2^n entries (where n is number of items). The transaction table contains frequency count of every possible transaction on

6. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Database,” *Proceedings of the 1993 ACM SIGMOD* International Conference on Management of Data, Vol. 22, Issue 2, 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487-499.

[3] J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation,” *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 1-12.

[4] Hidber, C. (1999). Online association rule mining (Vol. 28, No. 2, pp. 145-156). ACM.

[5] J Das, A., Ng, W. K., & Woon, Y. K. (2001, October). “Rapid association rule mining” In *Proceedings of the tenth international conference on Information and knowledge management* (pp. 474-481). ACM.

[6] J. S. Park, M. S. Chen, and P. S. Yu, "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," IEEE Trans. on Knowledge and Data Engineering, Vol. 9, No.5, Sep/Oct 1997, pp. 813-825.

[7] NCSA Computational Resources, Retrieved May 14,2006 from *http:// www.ncsa.uiuc.edu/UserInfo/*

[8] Longest Common Subsequence problem . Available at : wikipedia.org.