An Improved Frequent Itemset Generation Algorithm Based On Correspondence

Ajay R Y[1], Sharath Kumar A[2], Preetham Kumar[3], Radhika M. Pai[4]

Department of Information and Communication Technology
Manipal Institute of Technology, Manipal University, Manipal-576104, India
ajaycse3@gmail.com, sharathkumara@yahoo.co.in

Abstract

Association rules play a very vital role in the present day market that especially involves generation of maximal frequent itemsets in an efficient way. The efficiency of association rule is determined by the number of database scans required to generate the frequent itemsets. This in turn is proportional to the time, which will lead to the faster computation of the frequent itemsets. In this paper, a single scan algorithm which makes use of the mapping of the item numbers and array indexing to achieve the generation of the frequent item sets dynamically and faster. The proposed algorithm is an incremental algorithm in that it generates frequent itemsets as and when the data is entered into the database.

Keywords

Maximal Frequent Itemset, Support, Data Mining, Mapping.

1. INTRODUCTION

The frequent itemsets involve the generation of the most frequent itemsets from the given set of transactions with the given support value [1]. The frequent itemsets are the items which occur frequently in multiple transactions. The number of frequent itemsets need to be generated varies depending on the application. The frequent itemsets are used to make decisions regarding the production of the sets of items that are bought more frequently, by benefitting the end users-retailers. In some of the business applications, the number of transactions may be large, hence there should be faster way of computing the frequent itemsets.

2. EXISTING ALGORITHM AND DRAWBACKS

The basic algorithm that is being widely used today in association rule for generating the frequent itemsets is the Apriori algorithm [1] [4]. Algorithm's basic idea is to identify all the frequent itemsets which exceeds the predefined threshold support. In other words frequent items generates strong association rule, which must satisfy minimum support and minimum confidence [2]. Even this algorithm is simple and clear, it has some limitations. It is costly to handle a huge number of candidate sets. Initially the frequent one itemsets are generated based on the given candidate one itemsets in the transaction. The candidate 2-itemsets generation is based on the frequent 1-itemsets. Now again the generation of the frequent 2-itemsets requires the entire scan of the transactions to count for the required support in the candidate 2-itemsets [3] [7]. This process goes on until the required number or all the possible number of frequent itemsets are generated for the given set of transactions. Hence Apriori algorithm needs ‘n’ number of passes for the generation of frequent n-itemsets. Hence the time required to obtain the maximal frequent itemsets would be more. Apart from this, the major drawback is to wait until the last transaction. Hence there is a need for proposing a new algorithm to generate the frequent itemsets as and when each transaction is entered into the database and also to reduce the number of passes possible.

To effectively extract information from a huge amount of data in databases, the knowledge discovery algorithms must be efficient and scalable in large databases [5]. Most of the algorithms apply only to static databases. That is, when more transactions are added, the process of generation of the frequent itemsets must start again from the beginning [6]. In the proposed algorithm, there is no restriction on the number of transactions.

3. PROPOSED WORK

The proposed algorithm involves the generation of the frequent itemsets as and when the transaction is entered into the database and also reducing the number of scans. In this work, a single scan algorithm is proposed to generate the frequent itemsets which makes use of mapping of the item numbers into the array index during computation. In this approach, since algorithm is incremental, the frequent itemsets are mapped for the particular index in an array during a single scan.

3.1 Algorithm

The main idea of this algorithm is to keep track of the frequent itemsets as and when a particular transaction is entered into the database. When a new transaction is added into the database, the counters of the corresponding itemsets are incremented. After incrementing the counters of the itemsets obtained are matched with the support value to obtain the frequent itemset.
Pseudocode for the proposed algorithm in generic.

```
C is an array which counts the itemset frequency
f-set collects all frequent itemsets
Begin
        Initialize the elements of array C to zero
        Read each transaction
        for each transaction  read
        begin
             Consider all items present in the transaction as a set
             Generate all the subsets of the above set
             Increment the array C element considering the subset as the index
        end
        for all element of array C
            if element is exceeding support value
                store the array index in f-set
End
```

Pseudocode for the generation of the frequent itemsets for the number of items being less than 100.

```
// Computation for generation of frequent 1 itemset.
      for i =1 to n
              if a[i] = 1
                    c[i]++;
// Computation for 2 itemsets
      for i = 1 to n
```

```
            for j = i+1 to n
                  if a[i] = 1 and a[j] = 1
                        c[i*100+j]++;
// Computation for 3 itemsets
        for i =1 to n
              for j = i+1 to n
                    for k = j+1 n
                          if a[i] =1 and a[j ]= 1 and a[k] = 1
                                c[i*10000+j*100+k]++;
                                . . . . . . . . . . .
```

As an example, consider a transaction containing 3 items, 011(the second and the third items are added in this transaction). The counter c2 and c3 gets incremented. In correspondence with this, c23 would be incremented as well. Thus after all the transactions are complete, the counters are checked for the support value to obtain the frequent itemsets. The counters indicate the indexes to an array corresponding to the item numbers. Hence the mapping can be done easily. The support of the itemset 'i' can be checked by looking at the content of the array at index i, as each time the item i occurs the content at that index is incremented.

3.2 Demonstration of the Mapping

As an example for demonstration of the pseudocode for mapping items into array index, consider the set of transactions as shown in Table 1, Based on the algorithm and the pseudocode specified above, the mapping of the itemsets into array index takes place as follows.

If 'a' is an array keeping track of the count of the itemsets and minimum support (threshold value) is 3, then the transaction proceeds as follows.

Table1. Transaction set.

	Item 1	Item 2	Item 3	Item 4	Item 5
Trans1	1	1	1	0	0
Trans2	0	1	1	0	1
Trans3	0	0	0	1	1
Trans4	1	0	1	1	1
Trans5	0	1	1	0	0

In the scan of the first transaction, a[1], a[2], a[3], a[12], a[13], a[23] and a[123] gets incremented. Now the array "a" is updated as follows: a[1]=1, a[2]=1, a[3]=1, a[12]=1, a[13]=1, a[23]=1, a[123]=1.

In the scan of the second transaction, a[2], a[3], a[5], a[23], a[25], a[35] and a[235] gets incremented. Now array "a" would be updated as -> a[1]=1, a[2]=2, a[3]=2, a[5]=1, a[12]=1, a[13]=1, a[23]=2, a[25]=1, a[35]=1, a[123]=1 and a[235]=1.

In the third transaction, a[4], a[5], a[45] gets incremented. Now array gets updated as follows, a[1]=1, a[2]=2, a[3]=2, a[4]=1, a[5]=2, a[12]=1, a[13]=1, a[23]=2, a[25]=1, a[35]=1, a[45]=1, a[123]=1 and a[235]=1.

In the scan of fourth transaction, a[1], a[3], a[4], a[5], a[13]. a[14], a[15], a[34], a[35], a [45], a[134], a[145], a[345] gets incremented. Now the array gets updated as follows, a[1]=2, a[2]=2, a[3]=3, a[4]=2, a[5]=3, a[12]=1, a[13]=2, a[14]=1, a[15]=1, a[23]=2, a[25]=1, a[34]=1, a[35]=2, a[45]=2, a[123]=, 1[134]=1, a[145]=1, a[235]=1 and a[345]=1.

In the scan of fifth transaction, a[2], a[3] and a[23] gets incremented, the array would updated as follows, a[1]=2, a[2]=3, a[3]=4, a[4]=2, a[5]=3, a[12]=1, a[13]=2, a[14]=1, a[15]=1, a[23]=3, a[25]=1, a[34]=1, a[35]=2, a[45]=2, a[123]=1, a[134]=1, a[145]=1, a[235]=1 and a[345]=1. So frequent itemsets are {2}, {3}, {5} and {2, 3}.

4. ADVANTAGES

Since the algorithm is based on the array index mapping, the algorithm is best suitable when used for the incremental approach, i.e. as and when the data is entered into the database, the value of the particular array index is incremented corresponding to the items. Hence it is not required to explicitly generate the frequent itemsets. In this approach, the frequent itemsets are available at any point of time. Generation of the n-frequent itemsets is independent of the candidate itemsets and also on (n-1) frequent itemsets. The algorithm is reliable even if there are millions of transactions.

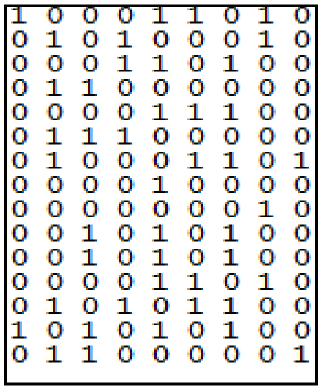
5. DISADVANTAGES

Memory is not used efficiently as only some of the array indexes are mapped to the items and the remaining part of the array would not be utilized. There is a limit on the number of items in the transactions depending on the availability of memory and also the maximal frequent itemsets to be generated.

6. RESULT ANALYSIS

Table 2 shows the set of transactions t hat are being entered, while Figure 1 shows the snapshot of the output generated for given input transactions. For the given example, we considered minimum support as 3. From Figure 1, we can observe that the frequent itemsets are generated

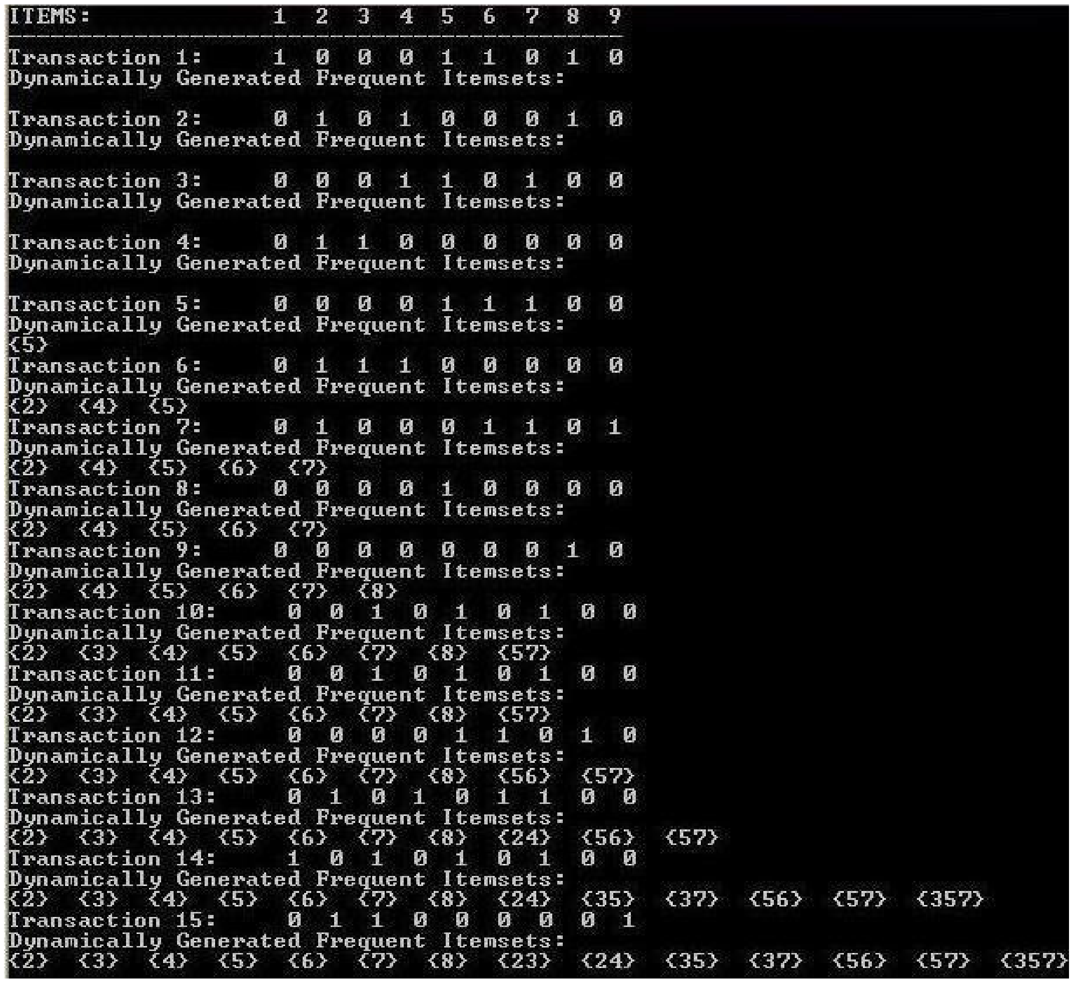
Table 2. Transactions Sets.



1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0
0	0	0	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0
1	0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0	1

implicitly as and when each transaction is updated into the data base. In the time analysis, the computation of 1-frequent itemset took an average of 40207 ns for a given set of 15 transactions and 4 items, while apriori took 116343 ns for the same set of transactions and items.

Fig. 1. Correspondence Result.



```
ITEMS:              1  2  3  4  5  6  7  8  9
-------------------------------------------------
Transaction 1:      1  0  0  0  1  1  0  1  0
Dynamically Generated Frequent Itemsets:

Transaction 2:      0  1  0  1  0  0  0  1  0
Dynamically Generated Frequent Itemsets:

Transaction 3:      0  0  0  1  1  0  1  0  0
Dynamically Generated Frequent Itemsets:

Transaction 4:      0  1  1  0  0  0  0  0  0
Dynamically Generated Frequent Itemsets:

Transaction 5:      0  0  0  0  1  1  1  0  0
Dynamically Generated Frequent Itemsets:
<5>
Transaction 6:      0  1  1  1  0  0  0  0  0
Dynamically Generated Frequent Itemsets:
<2>  <4>  <5>
Transaction 7:      0  1  0  0  0  1  1  0  1
Dynamically Generated Frequent Itemsets:
<2>  <4>  <5>  <6>  <7>
Transaction 8:      0  0  0  0  1  0  0  0  0
Dynamically Generated Frequent Itemsets:
<2>  <4>  <5>  <6>  <7>
Transaction 9:      0  0  0  0  0  0  0  1  0
Dynamically Generated Frequent Itemsets:
<2>  <4>  <5>  <6>  <7>  <8>
Transaction 10:      0  0  1  0  1  0  1  0  0
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <57>
Transaction 11:      0  0  1  0  1  0  1  0  0
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <57>
Transaction 12:      0  0  0  0  1  1  0  1  0
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <56>  <57>
Transaction 13:      0  1  0  1  0  1  1  0  0
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <24>  <56>  <57>
Transaction 14:      1  0  1  0  1  0  1  0  0
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <24>  <35>  <37>  <56>  <57>  <357>
Transaction 15:      0  1  1  0  0  0  0  0  1
Dynamically Generated Frequent Itemsets:
<2>  <3>  <4>  <5>  <6>  <7>  <8>  <23>  <24>  <35>  <37>  <56>  <57>  <357>
```

In the proposed algorithm, the generation of the maximal frequent itemsets is independent of the generation of previous maximal frequent itemsets. As an example, the generation of maximal frequent 4-itemsets is independent of the result of frequent 3-itemsets.

7. Conclusion And Future Work

The proposed algorithm generates the frequent item sets in a single pass in an efficient way which makes use of the mapping of the item numbers and array indexing. The algorithm also supports for the incremental approach. Generation of the frequent itemsets are independent of the candidate itemsets. The scope for the future work includes generation of the large number of frequent itemsets in an efficient way in a single pass, i.e. making use of the array efficiently, irrespective of the number of items in the memory.

References

[1] Pujari, A.K. 2001, *Data mining Techniques*, Universities Press (India) Private Limited, Hyderabad.

[2] Agrawal, R., and Srikant, R. 1994,'Fast Algorithms for Mining Association Rules in Large Databases', *In Proceedings of the 20th international conference on Very Large Data Bases,* pp 478-499.

[3] Agrawal, R., Imielinski, T., and Swami, A. 1993,'Mining Association Rules between Sets of Items in Large Databases', *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington ,DC,pp.207-216.

[4] Wei Yong-qing, Yang Ren-hua, and Liu Pei-yu.2009, "An improved Apriori algorithm for association rules of mining," *IT in Medicine & Education, 2009. ITIME '09. IEEE International Symposium on*, vol.1, pp.942-946.

[5] Chen, M., Han, J., and Yu, P.S., 1996,'Data Mining: An Overview from a Database Perspective', *IEEE Transactions on Knowledge and Data Engineering*, Vol.8, No.6, pp.866-883.

[6] Omiecinski, E., and Savasere, A. 1998,'Efficient Mining of Association Rules in Large Dynamic Databases', *Proceedings of the 16th British National Conference on Databases: Advances in Databases*, pp.49-63.

[7] Rao, S., and Guptha, P., 2012,' Implementing Improved Algorithm Over APRIORI Data Mining Association Rule Algorithm', *IJCST,* Vol. 3, Issue 1 Jan. – March.