Transaction Reduction Approach to Improve Efficiency of Apriori Algorithm

Sakshi Aggarwal
SGT Institute of Engineering & Technology
Gurgaon, Haryana
sakshii.26@gmail.com

Dr. Ritu Sindhu
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 an algorithm to cut down the unnecessary transactions and redundant generation of items during pruning which helps to reduce the scanning time.

Index Terms—Apriori algorithm, Support, Frequent Itemset, Association rules, Candidate Item

I. 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.

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 items from the database. But as the dimensionality of the database increase with the number of items, then:

a) More search space is needed and I/O cost will increase.

b) 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.

II. 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, \( \text{AGI}, \text{BGI, A} \cap \text{B}=\emptyset \). The whole point of an algorithm is to extract the useful information from these transactions.

III. 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. \( L_1=\{\text{frequent-1 item-sets}\} \);
2. for (k=2; \( L_{k-1} \neq \emptyset \); k++) {
3. \( C_k=\text{generate\_Apriori (L}_{k-1})\); //New candidates
4. for all transactions \( t \in \text{D} \) do begin
5. \( C_k=\text{subset (} C_k, t) \); //Candidates contained in \( t \)
6. for all candidates \( c \in C, \) do
7. \( \text{c.count}++ ; \)
8. }
9. \( L_k=\{c \in C, \text{c.count} \geq \text{minsup}\} \)
10. end for
11. **Answer=U_L_k**

Algorithm 1: Apriori Algorithm [6]

The basic steps to mine the frequent elements are as follows:

**Generate and test:** In this first find the 1-itemset frequent elements \( L \) by scanning the database and removing all those
elements from C which cannot satisfy the minimum support criteria.

Join step: This step is used to find next level element C\(_k\). Self join of L\(_k\) is done with L\(_{k-1}\). This step generates new C\(_k\) using this self join approach.

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\(_k\)). 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\(_k\).

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\(_k\) 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. for all itemsets c \(\epsilon\) C\(_k\) do
5.   for all \{s \supset (k-1) of c\} do
6.     if (s \notin L\(_{k-1}\)) then
7.       from C\(_k\), delete c

Algorithm 2: Apriori-Gen Function [6]

A. 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.

IV. TRANSACTION REDUCTION APPROACH

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

A. Improvement of Apriori

In Apriori algorithm, when database size is large, large numbers of candidate item sets are generated. Input output cost increases during generation of C\(_k\) due to large number of records in database. In our approach of transaction reduction, size of transaction (SOT) concept is used. SOT refers to the number of items in a transaction of a database. During C\(_k\) generation, after each process, transactions will be deleted from database according to value of k. When k=SOT, delete transactions from database whose SOT is equal to k.

Description of proposed Algorithm is as follows:
Next step is used to generate C₂ and L₂ i.e. k=2. Transaction T₉ is deleted from database as its SOT=k-1. Resultant DB is D₁. L₁ is self joined with L₁ to generate itemsets in C₂. Transactions in D₁ are scanned and support count for C₂ is determined. L₂ consists of those itemsets which satisfy minimum support value. C₂ and L₂ are shown in figure 5.

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 Fig. 13.

VI. COMPARATIVE ANALYSIS

We have counted the number of transactions that are scanned to find L₁, L₂ and L₃ for our given example and below table shows the difference in count of transactions scanned by using original Apriori algorithm and our proposed idea.

<table>
<thead>
<tr>
<th>k</th>
<th>Classical Apriori</th>
<th>Proposed approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>9</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

In this paper, cutting down the database size was the key point to improve the Apriori algorithm efficiency. This method not only optimizes the algorithm by reducing the size of Cₖ, but it also improves the input output cost by cutting down the transaction from the database. Performance of the algorithm is overall improved. Although our approach has improved and optimized the efficiency, but the overhead to maintain new database has to be managed. Future work focuses on to find a solution to reduce the overhead to maintain new database. One of the solutions is to reduce the database among multiple processors.

REFERENCES