Comparative Evaluation of Association Rule Mining Algorithms with Frequent Item Sets

Vimal Ghorecha
(Department of MCA, ATMIYA Institute of Technology & Science, India)

Abstract: This paper represents comparative evaluation of different type of algorithms for association rule mining that works on frequent item sets. Association rule mining between different items in large-scale database is an important data mining problem. Now a day there is lots of algorithms available for association rule mining. To perform comparative study of different algorithms various factor considered in this paper like number of transaction, minimum support and execution time. Comparisons of algorithms are generated based on experimental data which gives final conclusion.

Keywords – Apriori, Association Rules, Data Mining, Frequent Pattern

I. INTRODUCTION

Many change and many of the developments in association rule mining for the last decade are due to new algorithms introduced. On the one hand where the main aim of association rule mining is to provide the better rule and frequent item set to the predict and make decision. This section explains core concept of frequent itemset and association rule mining. Frequent itemset are those items which occur in transaction frequently. Any transaction T has element X if and only if it the element X is with that transaction T. An transaction is associated with an identifier, called TID. A formal definition is [4] Let X is a set which contains different items, D is a database, is a set of database transactions (variable length) where each transaction T contains a set of items i1, i2, i3, i4………. ik ∈ I. Each transaction is associated with an identifier, called TID. A formal definition is [4] Let X is a set which contains different items. Any transaction T has element X if and only if it the element X is with that transaction T. An association rule is an implication of the form X → Y, where X, Y ⊆ I and X ∩ Y = Ø. Here X is called the antecedent and Y is called the consequent of the rule. The rule X → Y satisfies in the dataset with confidence c if within all transactions that contain X also contains Y with ratio of c%. The rule X → Y has support s in dataset D, if s% of transaction in dataset D either contain X or Y (X ∪ Y). The selection of association rules is based on these two values (Other constraints also matters). These are two important measures of rule importaneness. They can be described by the following equations:

\[
\begin{align*}
\text{support}(X \rightarrow Y) & = \frac{\text{Freq}(X \cup Y)}{\text{Total Trns}} \\
\text{confidence}(X \rightarrow Y) & = \frac{\text{Freq}(X \cup Y)}{\text{Freq}(X)}
\end{align*}
\] (1) (2)

The first rule is \{i1, i2, …, ik-1\} → \{ik\}, by checking the confidence. We can create rules from large item sets which satisfies minimum support. First we check that rule can be satisfied by our requirements. If rule is not as per our requirement new rule are generated by removing items from antecedent and inserting this items to consequent. This new rule again checked for confidence and further rule can be satisfied by our requirements.
The above processes continue until the antecedent does not have any item of say it becomes empty. Our second problem is very easy so every researcher concentrates on first problem.

The first sub-problem can be further divided into two sub-problems: candidate large item sets generation process and frequent item sets generation process. We call those item sets whose support exceeds the support threshold as large or frequent item set.

This requires multiple scan of dataset. Each time for checking support and creating rule based on frequent itemset we need to scan whole dataset and we can say all transactions.

We already discuss the process of creating rules in this section which divided into two parts [5]. First one is to identify all frequent item sets. Second generate strong rules which satisfy minimum support and minimum confidence [6].

III. **APRIORI ALGORITHM**

Apriori is an algorithm [1] is very popular and previous first algorithm for mining frequent item sets. It generates candidate itemset and frequent itemset, like at each level are utilized in next level to generate candidate itemset. As we said earlier it requires multiple database scans, as many as longest as frequent item sets. Apriori employs an iterative approach known as level-wise search, where k item set are used to explore (k+1) item sets. There are two steps in each iteration.

The basic Apriori algorithm is plinth of association rule mining classic Apriori algorithm inspires different researcher how to mine association rules and the market basket analysis example is most popular in 1990s.

L1 = {large 1-itemsets};
for ( k = 2; Lk-1 ≠ Ø; k++) do begin
Ck = apriori-gen(Lk-1); //New candidates
forall transactions t ∈ D do begin
Ct = subset(Ck, t);
//Candidates contained in t
forall candidates c ∈ Ct do
c.count++;
end
Lk = { c ∈ Ck | c.count ≥ minsup }
end
Answer = ∪ k Lk;

The first weakness of this algorithm is the generation of a large number of candidate item sets. The second problem is the number of database passes which is equal to the max length of frequent item set.

IV. **THE FP-GROWTH ALGORITHM**

The FP-growth [2] algorithm for mining frequent patterns with FP-tree by pattern fragment growth is:

A FP-tree constructed with the above mentioned algorithm; where D – transaction database; s – minimum support.

Output of this algorithm is complete set of frequent itemset without generating candidate set and without multiple database scans.

Method:
call FP-growth(FP-tree, null).

Procedure FP-growth(Tree, A)
{
  if Tree contains a single path P
  then for each combination (denoted as B) of the nodes in the path P do
    generate pattern B ∪ A with support=minimum support of nodes in B
  else for each ai in the header of the Tree do
  {
    generate pattern B = ai ∪ A with support = ai.support;
    construct B’s conditional pattern base and B’s conditional FP-tree
    TreeB;
    if TreeB ≠ Ø
    then call FP-growth(TreeB, B)
  }
}
V. THE CLOSURE ALGORITHM

Candidate = all 1-item sets; KFrequent = empty; Maximal = empty
while (true) do
    initialize the elements of matrix M with 0;
    count the support for all elements of Candidate
    and update their corresponding rows in M;
    for all item sets C in Candidate do
        if (C is not frequent)
            continue to next itemset;
        add C to KFrequent;
        compute closure of C in cl(C);
        compute all extended closures of C and add them to Maximal;
        if no extended closures were found
            add cl(C) to Maximal;
        od
    empty Candidate;
    generate new candidate item sets in Candidate based on KFrequent;
    if Candidate is empty
        break;
    empty KFrequent;
    for all item sets L in Maximal
        mark as frequent all item sets from Candidate
        those are contained in L;
    for all item sets C in Candidate
        if C is marked as frequent
            add C to KFrequent;
        empty Candidate;
        generate new candidate item sets in Candidate based on KFrequent;
        if Candidate is empty
            break;
        empty KFrequent;
    od
return Maximal

Note that the Closure algorithm requires \((n/2 + 1)\) scans of the database where \(n\) are the dimension of the longest frequent itemset.

VI. THE MAXCLOSURE ALGORITHM

Candidate = all 1-itemsets; Frequent = empty; Maximal = empty;
while (true) do
    initialize the elements of matrix M with 0;
    count the support for all elements of Candidate
    and update their corresponding rows in M;
    for all item sets C in Candidate do
        if C is not frequent
            continue to next itemset;
        compute closure of C in cl(C);
        for all extended closures xcl(C) of C
            if xcl(C) has not been already added to Frequent
                add xcl(C) to Frequent;
        if no extended closures were found
            if cl(C) has not been already added to Maximal
                add cl(C) to Maximal;
        od
        if Frequent is empty
            break;
    empty Candidate;
    move into Candidate all elements from Frequent;
    od
return Maximal
VII. COMPARATIVE EVALUATION BETWEEN A PRIORI, FP-GROWTH, CLOSURE AND MAXCLOSURE ALGORITHM

Four different association rule mining algorithm implemented in Java and tested based on different criteria. The platform specification for this test was:

Intel Core-i3-2330M 2.20GHz processor, with 4 GB RAM, Windows 7 64bit. To study the performance and scalability of the algorithms generated data sets with 10K to 500K transactions, and support factors of 25% and 33% were used. Any transaction may contain more than one frequent itemset. The number of items in a transaction may vary, as well as the dimension of a frequent itemset. Also, the number of items in an itemset is variable. Taking into account these considerations, the generated data sets depend on the number of items in a transaction, number of items in a frequent itemset, etc.

VII.1. Comparative evaluation between Apriori and FP-Growth

From TABLE-1 and Fig.1 we can tell that FP-Growth algorithm is more efficient than classic Apriori algorithm. In analysis process we have taken records from 10K to 500K with support of 33% and 100 numbers of items.

<table>
<thead>
<tr>
<th>Records</th>
<th>Apriori (Time in Seconds)</th>
<th>Fp-Growth (Time in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1.358</td>
<td>1.352</td>
</tr>
<tr>
<td>50000</td>
<td>6.822</td>
<td>6.651</td>
</tr>
<tr>
<td>100000</td>
<td>13.273</td>
<td>13.133</td>
</tr>
<tr>
<td>300000</td>
<td>40.906</td>
<td>39.657</td>
</tr>
<tr>
<td>500000</td>
<td>99.317</td>
<td>89.314</td>
</tr>
</tbody>
</table>

Fig. 1. Apriori vs. Fp-Growth

When we increase the size of record we can see in the table the results are very efficient related to time.

VII.2. Comparative evaluation between Apriori and Closure

From TABLE-2 and Fig.2 we can tell that Closure algorithm is more efficient than classic Apriori algorithm. In analysis process we have taken records from 10K to 500K with support of 33% and 100 numbers of items.

<table>
<thead>
<tr>
<th>Records</th>
<th>Apriori (Time in Seconds)</th>
<th>Closure (Time in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1.358</td>
<td>0.722</td>
</tr>
<tr>
<td>50000</td>
<td>6.822</td>
<td>3.444</td>
</tr>
<tr>
<td>100000</td>
<td>13.273</td>
<td>6.671</td>
</tr>
<tr>
<td>300000</td>
<td>40.906</td>
<td>20.907</td>
</tr>
<tr>
<td>500000</td>
<td>99.317</td>
<td>68.678</td>
</tr>
</tbody>
</table>

Fig. 2. Apriori vs. Closure
When we increase the size of record we can see in the table the results are very efficient related to time.

VII.3. Comparative evaluation between Apriori and MaxClosure

From Table-3 and Fig.3 we can tell that MaxClosure algorithm is more efficient than classic Apriori algorithm. In analysis process we have taken records from 10K to 500K with support of 33% and 100 numbers of items.

TABLE 3

<table>
<thead>
<tr>
<th>Records</th>
<th>Apriori (Time in Seconds)</th>
<th>MaxClosure (Time in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1.358</td>
<td>0.677</td>
</tr>
<tr>
<td>50000</td>
<td>6.822</td>
<td>3.340</td>
</tr>
<tr>
<td>100000</td>
<td>13.273</td>
<td>6.563</td>
</tr>
<tr>
<td>300000</td>
<td>40.906</td>
<td>20.141</td>
</tr>
<tr>
<td>500000</td>
<td>99.317</td>
<td>65.288</td>
</tr>
</tbody>
</table>

When we increase the size of record we can see in the table the results are very efficient related to time.
VIII. COMPARATIVE EVALUATION BETWEEN ALL ALGORITHMS TOGETHER.

<table>
<thead>
<tr>
<th>Records</th>
<th>Apriori (Time in Seconds)</th>
<th>Fp-Growth (Time in Seconds)</th>
<th>Closure (Time in Seconds)</th>
<th>MaxClosure (Time in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1.358</td>
<td>1.352</td>
<td>0.722</td>
<td>0.677</td>
</tr>
<tr>
<td>300000</td>
<td>40.906</td>
<td>39.657</td>
<td>20.907</td>
<td>20.141</td>
</tr>
<tr>
<td>500000</td>
<td>99.317</td>
<td>89.314</td>
<td>68.678</td>
<td>65.288</td>
</tr>
</tbody>
</table>

For the comparative study of Classical Apriori, FP-Growth, Closure and MaxClosure Algorithm, we have taken a database of different transaction form 10K to 500K with different number of items 100 and 200.

In this analytical process we considered 500 transactions to generate the frequent pattern with the support count 25% and 33%. We have repeated the same process by increasing the transaction.

Here we first compare the result of all approaches with Classical Apriori Algorithm because all approaches are based on Classical Apriori. After that we compare all approaches to find out the best. After the experiment on all approaches, we have designed a graph and summarized a result in the following table.
Comparative Evaluation of Association Rule Mining Algorithms with Frequent Item Sets

IX. CONCLUSION

From above experimental data we can conclude that the MaxClosure algorithm is better than all algorithms. We can also conclude that FP-Growth is better than Apriori. Closure is better than FP-Growth and MaxClosure is better than Closure Algorithm. Here we just consider time as factor. If we consider other factor other than time the result may be vary from factor to factor. Performance of all algorithms is not affected by support factor as well as number of items.

If we closely monitor Table – 5 we can say that time taken by Apriori algorithm when records is 500K and Support is 25%, is less than time taken by Apriori when records is 500K and Support is 33%. So based on data we can conclude that classical Apriori algorithm affected by Support Factor.

If we closely monitor Figure – 5 we can say that time taken by Closure and MaxClosure when records are 500K is same as time taken by Apriori and FP-Growth when records is 300K. So based on data we can conclude that both Closure algorithms take almost half of the time taken by Apriori and FP-Growth.

REFERENCES

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