ISSN:0975-9646

Comparison of Frequent Item Set Mining Algorithms

J.R.Jeba
Department of Computer Applications
Noorul Islam Centre For Higher Education
Noorul Islam University, Kumaracoil

Dr.S.P.Victor
Department of Computer Science
St.Xavier's College (Autonomous)
PalayamKottai.

***Abstract*-Frequent item sets mining plays an important role in association rules mining. Over the years, a variety of algorithms for finding frequent item sets in very large transaction databases have been developed. The main focus of this paper is to analyze the implementations of the Frequent item set Mining algorithms such as SMine and Apriori Algorithms.**

***General Terms*-Data Mining, Frequent Item sets, Association Rule Mining.**

***Keywords*-SMine, item_count, frequent_items.**

1. INTRODUCTION

Association rule mining is a focused area in today's data mining research. It usually consists of two phases viz., discovery of frequent itemsets and generation of rules from the discovered frequent itemsets. Finding frequent itemsets has gained popularity because it has more number of applications. Many research have been done and algorithms developed. [7, 8, 9]

A number of algorithms for mining frequent item sets have been proposed after Agarwal first introducing the problem of deriving categorical association rule from transactional databases in [8]. These existing algorithms uses the candidate generate-and-test approach and the pattern growth approach.

In Apriori[1, 4] as well as many subsequent studies[5, 6], each iteration of the candidate generate-and-test approach, pairs of frequent k-item sets are joined to form candidate (k+1)-item sets, then scanned the database to verify their supports. The Apriori algorithm achieves good reduction on the size of candidate sets, however, it takes many scans of the database to check the candidate item supports as much as the most long length of patterns. This algorithm is also compared with DIC Alogrithm [2].

In SMine[3] algorithm for mining the set of all frequent itemsets in the database by reducing the number of scans. During the first database scan the number of occurrences of each item is determined and the infrequent ones are discarded. Then the frequent items are counted in each transaction. The transactions are sorted based on the number of frequent items in descending order. Then graph based approach is used to find the frequent item sets.

In this paper, we analyze the implementations of frequent item set algorithms SMine and Apriori algorithm.

The organization of this paper is as follows: In Section 2, we give an insight into the detailed problem description. Section 3 explains the Apriori algorithm. In section 4, we give a detailed explanation of SMine algorithm used for generating frequent itemsets. In Section 5, the implementation and performance comparision of Apriori and SMine algorithm are given. We end with our conclusion in Section 6.

2. PROBLEM DESCRIPTION

Let $I = \{I_1, I_2, \ldots I_n\}$ be a set of items. Let D, the task-relevant data, be a set of transactions in a supermarket, where each transaction T is a set of items, such that $T \subseteq I$. Each transaction is assigned an identifier called TID. Let A be a set of items, a transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \varnothing$. The rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., both A and B). This is taken to be the probability $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B|A). That is, *Support* $(A \Rightarrow B) = P(A \cup B) = s$, *Confidence* $(A \Rightarrow B) = P(B|A) =$Support $(A \Rightarrow B)$/Support $(A)=c$. Thus association rules is composed of the following two steps: 1) Find the large item sets that have transaction support above a minimum support and 2) From the discovered large item sets generate the desired association rules.

2.1 Two Phases of Association Rule Mining

There are two phases to deal with association rule mining. First one is about the algorithm efficiency. The research on developing out an algorithm with less computation complexity is one of the most interesting topics related to association rule mining. The mining efficiency is so important because association rule mining always works on large database. The number of rules grows exponentially with the number of items. The related work mainly focus on efficient pruning on large data sets and reducing the times of scanning data.
Second phase is to find the effective ways needed to select interesting rules from discovered rules.

3. APRIORI ALGORITHM

3.1 Procedure

Apriori is an influential algorithm for mining frequent itemsets for Boolean association rules. This algorithm uses prior knowledge of frequent itemset properties. This algorithm iteratively finds all possible itemsets that have support greater or equal to a given minimum support value. First,the set of frequent frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. This resulting set is denoted as L_1. Next L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database. The output of the Apriori algorithm consists of a set of k-itemsets (k=1,2,……..) that have support greater or equal to a given minimum support value.

Apriori Works as follows :

1. First, the 1-itemset C_1 is obtained. Based on C_1, we count the frequency of occurrence of each itemset in all the transactions, prune the itemsets that do not exceed the support threshold and get the frequent 1-itemset L_1.
2. For all k, k=2,…, use L_k to generate C_{k+1} by L_k join L_k. For each of these (k+1)-itemsets, check whether all its k subset is in the frequent k-itemset L_k. If the answer is yes, keep it in C_{k+1}; otherwise, prune it. After getting C_{k+1}, scan the database once again to count the support for all the itemsets in C_{k+1}, prune those do not exceed the support threshold and get L_{k+1}.
3. Repeat the above step 2 until no itemset could be generated for C_{k+1}.

Apriori algorithm needs to scan the database multiple times. When mining a huge database, multiple database scans are costly. One feasible strategy to improve the efficiency of Apriori algorithm is to reduce the number of database scans.

4. SMINE ALGORITHM

SMine algorithm is an efficient algorithm to find frequent itemsets. The number of database scans is reduced when compared with Apriori algorithm. During the first database scan the number of occurrences of each item is determined and the infrequent ones are discarded. Then the frequent items are counted in each transaction. The transactions are sorted based on the number of frequent items in descending order. Then graph based approach is used to find the frequent item sets.

4.1 Procedure

Step 1: The algorithm scans the database in order to count the number of occurrences of each item to find the candidate 1-itemset with their support count.

Step 2: The set of frequent 1-itemset L_1 can then be determined by removing the items having less than the minimum support count. It consists of the candidate 1-itemsets satisfying minimum support. Let the number of frequent 1-itemset be ‘n’.

Step 3: Removes the infrequent items from each transaction and count the number of items in each transaction (item_count).

Step 4: The transactions are sorted in descending order based on the item count.

Step 5: Create a table called ’M’ with two columns namely ‘no.of.items’ and ‘no.of. transactions’**.** Let no=n;

Step 6: Add a row with no.of.items = no and no.of.transactions is equal to the number of transactions having item_count >= no. If no >2, then decrement ‘no’ value by 1 and repeat step 6.

Step 7: Select the maximum ‘no.of.items’ from the table M having the ‘no.of.transactions‘ equal to or greater than the minimum support count. Let it be m. Create a directed graph starting from all possible items in m-itemset as the header nodes in the first level, all the possible (m-1) itemset in the second level, all the possible (m-2) itemset in the next level and so on, until 2-itemset.

Step 8 : Get the ‘no.of transactions’ for m-itemset from the table M. Let it be ‘R’. If it is greater than or equal to the minimum support count, then find the support count of each unvisited node of the m-itemset by scanning first ‘R’ transactions. If the set is frequent, mark this node and all its sub nodes as frequent items. If a set is frequent, all its subsets must also be frequent.

Step 9 : Go to the next level .Let m=m-1. Repeat step 8 until m =2.

Step 10: All the marked nodes are frequent itemsets.

4.2 Algorithm :

Input: Database, D, of transactions; minimum support threshold, min_sup.

Output: L, frequent itemsets in D.

Method:

```
Begin
    L1= find_frequent_1-itemsets(D);
    n=number of items in L1;
    L2, L3,...., Ln are initially set to null;
    for each transaction t in D
    {
        //Removing infrequent items from
        transactions
         Delete item not in L1 from t .
          t. item_count = Number of frequent items.
    }
    Sort the transactions in D, descending order based on
    their item_count.
    call DB_details(D)
    call creategraph(L1, n)
End
procedure DB_details(D)
for( k=n; k>=2; k--)
      Mk=Number of transactions having item_count >=k;
// Arrange these values in a table called ‘M’
 end procedure
```

procedure creategraph(L_1, n)

// L_1 is frequent 1-itemset, n is the number of
// items in L_1
while(M_n satisfies min.support count)
n=n-1;
Add all possible n- itemsets as the header nodes
C_n = all possible 'n' itemsets.
for (k=n; k >= 3; k--)
Call subgraph(C_k)
for (k=n; k >= 2; k--)
Call frequent_items (k)
end procedure

procedure subgraph(C_k)

C_{k-1} = {}
for each item in C_k
{
C_{k-1} = C_{k-1} U (all the possible subset (k-1) itemsets)
for each subset (k-1) itemset
{
If the node is already created then
make a link to the parent node
else
create a new node for the item and make a link to
the parent node
}
}
end procedure

procedure frequent_items(k)

for each itemset in C_k
{
If (item is not visited || not marked) then
{
Find the support count by scanning first M_k transactions .
If it satisfies the min. support then
{
Mark this node and all its subnodes up to the level
2 itemsets are frequent.
Add these marked nodes to L_k, L_{k-1}, L_{k-2}, …., L_2 in
their respective array.
}
else
mark this node as infrequent.
}
}
end procedure

SMine Algorithm reduces the number of scans when compared to the Apriori Algorithm. It finds the candidate 1-itemsets using the same approach as Apriori. The remaining steps are different from Apriori.

5. IMPLEMENTATION AND ANALYSIS

This section deals with the implementation of Apriori and SMine Algorithm. We compare the frequent item sets obtained and the factors that affect the efficiency of the algorithms.

5.1 Result analysis:

For the experiment we have used datasets of Hepatitis and Heart dataset. These datasets was obtained from the UCI repository of machine learning databases [CC98].

Table 1 shows the characteristics of the datasets selected for the experiment.

To study the performance of the algorithms, support threshold of 30% to 70% were used. The Table 2 shows that the execution time of the algorithms decreases with varying support thresholds for a hepatitis data set. Table 3 shows that the execution time of the algorithms with varying support thresholds for a heart data set. In these two cases, the execution time of SMine is less when compared to Apriori algorithm.

Table 1: Datasets used in comparison

Dataset	No.of transactions	No. of Attributes	Attribute Types
Hepatitis.D56. N155.C2. num	155	20	Categorical, Integer, Real
heart.D52.N30 3.C5.num	303	75	Categorical, Integer, Real

Table 2 : Execution Time for Apriori and SMine – Hepatitis Data set

Support (%)	Execution Time (Seconds)	
	Apriori	SMine
30	0.86	0.45
40	0.22	0.16
50	0.03	0.02
60	0.02	0.00
70	0.00	0.00

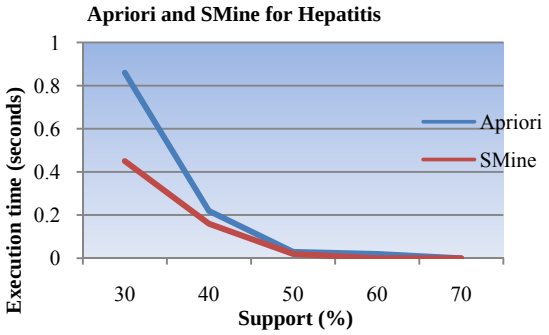

Fig. 1

Table 3 : Execution Time for Apriori and SMine- Heart Data set

Support (%)	Execution Time (Seconds)	
	Apriori	SMine
30	0.06	0.04
40	0.04	0.03
50	0.03	0.02
60	0.01	0.01
70	0.01	0.00

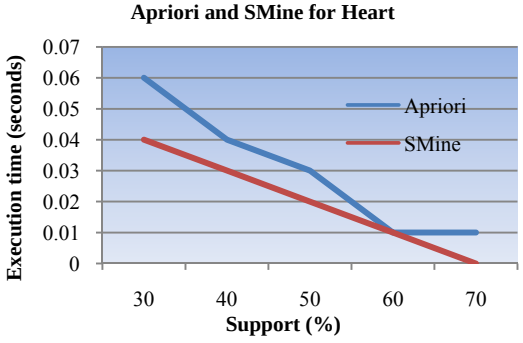

Fig. 2

Here, we present a performance comparison frequent Item set Mining algorithms Apriori and SMine. We find that, in Apriori requires multiple passes of the data to calculate the support count.

6. CONCLUSION

In this paper, Apriori algorithm is compared with SMine Algorithm using two different datasets namely Hepatitis and Heart datasets. As per the result, SMine Algorithm is superior to the Apriori algorithm.

7. REFERENCES

[1] Ashok Savasere, E. Omiecinski and S. Navathe, "An efficient algorithm for mining association rules in large databases", Proceedings of the 21st International Conference on Very large database, 1995, pp. 420-431.

[2] Deepti Chandra, Poonam Sao, Samta Gajbhiye , "Comparative Study of Apriori and DIC Algorithm", Proceedings of the Int. Conf. on Information Science and Applications ICISA 2010 6 February 2010.

[3] J.R.Jeba and S.P.Victor," A Novel Approach for finding Frequent item sets with Hybrid Strategies ", International Journal of Computer Applications, ,vol : 17, No: 5,March 2011.

[4] Jia Ling, Koh and Vi-Lang Tu, " A Tree-based Approach for Efficiently Mining Approximate Frequent Itemsets", IEEE International Conference on Research Challenges in Information Science, 2010, pp. 25-36.

[5] Jian Pei ,J. Han, J. Lu, H. Nishio.S.and Tang, "H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases", ICDM International Conference on Data Mining, ICDM, 2001, pp. 441-448.

[6] Jiawei Han, Jian Pei, and Yiwen Yin, "Mining Frequent Patterns without Candidate Generation", Proceedings of ACM SIGMOD Conference, Dallas, TX, 2000, pp.53-87.

[7] Jong Soo Park, M.S. Chen, and P.S. Yu, "An effective hash based algorithm for mining association rules", Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995, pp. 175-188.

[8] Ramesh Agrawal, Tomasz Imielinski, and A. Swami, "Mining association rules between sets of items in large databases", ACM-SIGMOD Int. Conf. Management of Data, Washington, D.C., May 1993, pp 207–216.

[9] Senthil Kumar A.V and R.S.D. Wahidabanu, "A Frequent Item Graph Approach for Discovering Frequent Itemsets", Proceedings of 2008 IEEE International Conference on Advanced Computer Theory, 2008, pp.952-956.