ISSN (Online): 2230-7893
www.IJCEM.org

ENHANCED FP-GROWTH ALGORITHM

Kuldeep Malik[1], Neeraj Raheja[2] and Puneet Garg[3]

[1] CSE Department, Royal Institute of Management & Technology
Panipat, Haryana, India
malik.4084@gmail.com

[2] CSE Department, Maharishi Markandeshwar University, M.M.E.C
Ambala, Haryana, India
neeraj_raheja2033@yahoo.co.in

[3] CSE Department, Haryana Institute Of Technology
Jhajjar, Haryana, India
Puneetgarg.er@gmail.com

Abstract

Based on analyzing an association rule mining algorithm called FP tree. a new association rule mining algorithm called Enhanced FP was presented. As the main disadvantage of FP-Growth is that it is very difficult to implement because of its complex data structure. In this FP-tree takes a lot of time to build and also needs more memory for storing the transactions. To overcome these disadvantages, I introduce Enhanced-FP, which does its work without any prefix tree and any other complex data structure. It processes the transactions directly, so its main strength is its simplicity.

Keywords: *Association rule mining; Frequent Pattern(FP); FP growth algorithm; Enhanced FP algorithm.*

1. Introduction

Data mining is an essential step in the process of knowledge discovery in databases, in which intelligent methods are applied in order to extract patterns. Thus data mining refers to extracting or mining knowledge from large amounts of data. Association rule mining is one of the most important and well researched techniques of data mining. Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database.

Let I= {i1, i2, i3…….im} be a set of items. Let D be a database of transactions. Each transaction t is represented as a binary vector with t[k] =1 if t brought the item ik and t[k] =0 otherwise. A set X= {i1…….ik} i.e. subset of I is called item set or a K-item set if it contains K items. A transaction over I is a couple T= (tid, I) where tid is the transaction identifier and I is an item set. An Association rule is an implication of form A⇒B, where A, B is subset of I, A∩B =Ø.A is called antecedent where B is called consequent.

There are two important basic measure for association rules, support(s) and confidence(c).

Support(s) of an association rule is defined as the percentage/fraction of records that contain X ⇒Y to the total number of records in the database. Suppose the support of an item is 0.2%, it means only 0.2 percent of the transaction contain purchasing of this item.

Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain X ⇒ Y to the total number of records that contain X. Confidence is a measure of strength of the association rules, suppose the confidence of the association rule X⇒Y is 90%, it means that 90% of the transactions that contain X also contain Y together

2. Fp-Growth Algorithm

FP-Growth approach is based on divide and conquers strategy for producing the frequent item sets. FP-growth is mainly used for mining frequent item sets without candidate generation. Major steps in FP-growth is-

Step1- It firstly compresses the database showing frequent item set in to FP-tree. FP-tree is built using 2 passes over the dataset.

Step2: It divides the FP-tree in to a set of conditional database and mines each database separately, thus extract frequent item sets from FP-tree directly.

It consist of one root labeled as null, a set of item prefix sub trees as the children of the root, and a frequent .item header table. Each node in the item prefix sub tree consists of three fields: item-name, count and node link where---item-name registers which item the node represents; count registers the number of transactions represented by the

portion of path reaching this node, node link links to the next node in the FP- tree. Each item in the header table consists of two fields---item name and head of node link, which points to the first node in the FP-tree carrying the item name. The pseudo code of mining on FP-tree is depicted in Figure 1.

```
Input: constructed FP-tree
Output: complete set of frequent patterns
Method: Call FP-growth (FP-tree, null).
procedure FP-growth (Tree, α)
{
    1) if Tree contains a single path P then
    2) for each combination do generate pattern β ∪ α with support = minimum support
       of nodes in β.
    3) Else For each header ai in the header of Tree do {
    4) Generate pattern β = ai∪α with support = ai.support;
    5) Construct β.s conditional pattern base and then β.s conditional FP-tree Tree β
    6) If Tree β = null
    7) Then call FP-growth (Tree β, β)}
}
```

Figure 1. Pseudo code of FP-Growth algorithm

3. Procedure of Enhanced Fp-Growth Algorithm

Enhanced-FP, which does its work without any prefix tree and any other complex data structure. It processes the transactions directly, so its main strength is its simplicity. The pseudo code of Enhanced FP-Growth is depicted in Figure 2.
The step by step processing of Enhanced-FP as follows:

Step 1: In the initial scan, the supports of the items are calculated. The items whose support count is less than minimum support are discarded and specified as infrequent items. Then the items in the database are sorted in ascending order with respect to their support. Let us consider a transactional database D. suppose the minimum support count is 3.
Transaction database (left), item support (middle), reduced transaction database with items in transactions sorted ascending w.r.t. their support.
F and G are the items which are discarded because their support count is less than the minimum support count

Step 2: The initial transaction database is converted in to a set of transaction list, with one list for each item. These lists are stored in array, each of which contains a pointer to the head of the list. The list elements consists a successor pointer and a pointer to the transaction. The transactions are inserted one by one in this by using their leading item as an index.

```
Function Enhanced-FP (a: array of transaction lists,
                      p: set of items,
                      s min: int) : int
Var i; k: item;
s: int;
n: int;
b: array of transaction lists;
t; u: transaction list element;
begin
n := 0;
While a is not empty do
i := last item of a; s := a[i].wgt;
If s >= s_min then
p := p ∪{i};
Report p with support s;
p: = p – {i};
end;
If s >= s_min then
b: = array of transaction lists;
t: = a[i].head;
While t = nil do
u: = copy of t; t: = t.succ;
k: = u.items[0];
remove k from u.items;
if u.items is not empty
then u.succ = b[k].head; b[k].head = u; end;
b[k].wgt := b[k].wgt +u.wgt;
end;
n := n + 1 + Enhanced-FP(b; p; s_min);
end;
t := a[i].head;
while t=nil do
u := t; t := t.succ;
k := u.items[0]
remove k from u.items;
if u.items is not empty
then u.succ = a[k].head; a[k].head = u; end;
a[k].wgt := a[k].wgt +u.wgt;
end; remove a[i] from a;
end;
return n;
end;
```

Figure 2. Pseudo code of Enhanced FP-Growth algorithm

Step3: the first list corresponding to the item e contains the second, seventh .and eight transactions, with the item e, removed. The counter in the array elements states the number of transactions contains the corresponding item. Now the transaction lists are traversed from left to right for finding all the frequent item set that contain the item the list corresponds to. Before a transaction list is processed, its support count is checked, if it exceeds than minimum support count

than there must be a frequent item set. Thus processing of transactions lists is as follows: For each list element the leading item of its transaction is retrieved and used as an index in to the list array, and then the element is added at the head of the corresponding list. In such type of reassignment, the leading item is also removed from the transaction. In addition a copy of the list elements is inserted in the same way in to an initially empty second array of transaction lists. Now the second array collects the subset of transactions that contains a specific item and represents them as a set of transaction list. This set of transaction lists is then processed recur lively, noting the item associated with the list it was generated from as a common prefix of all frequent item sets found in the recursion. After the recursion the next transaction list is reassigned, copied, and processed in a recursive call and so on. In the first step all item sets containing the item E are found by processing the leftmost list. The elements of this list are reassigned to the lists to the right and copies are inserted into a second list array. This second list array is then processed recursively, before proceeding to the next list, i.e., the one for item A. The list elements that contain only one item is neither reassigned nor copied, because the transaction would be empty after the removal of the leading item. In that case only the counter in the lists array element is incremented.

3.1 Implementation and Performance Evaluation

In this section, we compare the performance between Enhanced FP-Growth, FP-Growth and Apriori Algorithms. Graphs show the execution time of implementations over the various supports. The blue line refers to apriori algorithm. The red line refers to FP-Growth algorithm. The green line refers to Enhanced-FP.

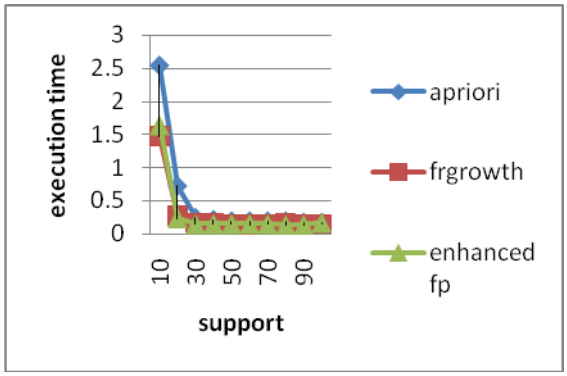

Figure 3. Results of Apriori, FP-Growth and Enhanced-FP

4. Conclusions

In this we introduced Enhanced-FP,which does its work without any complex data structure or prefix tree.Its main strength is its simplicity.There is no need to re-repersentation of transactions.I run C based implementations of these three algorithms.By comparing these frequent itemset mining algorithms apriori and fp-growth and Enhanced-FP, the strength of Enhanced-FP is analyzed. As the Experimental results show,Enhanced-FP clearly outperforms apriori and FP-Growth.It is faster than apriori and FP-Growth and is not expensive like FP-tree. Its Transactional database is memory resident

References

[1] Agrawal R, Imielinski T, Swami A. "Mining Association Rules between Sets of Items in Large Databases." Proc. Conf. on Management of Data, 207- 216.ACM press, New York, NY, USA. June 1993.

[2] Agrawal R, Srikant R. "Fast Algorithms for Mining Association Rules", in Proc.20th Intl. Conf. on very Large Databases (*VLDB'94),*Santiago de Chile,pp.487-99,1994.

[3] Agrawal R.C., Aggarwal C.C., and Prasad V.V., "A Tree Projection Algorithm for Generation of Frequent Item Sets," Parallel and Distributed Computing, pp. 350-371,(2000).

[4] Bodon F. "A fast apriori implementation". In Goethals B. and Zaki M.J.,editors, Proceedings of the IEEE ICDM Workshop on Frequent Item set Mining Implementation (FIMI'03), CEUR Workshop Proceedings Melbourne, Florida, USA, Vol. 90,(2003).

[5] Grahne G. and Zhu J., "Efficiently Using Prefix-Trees in mining Frequent Item sets," Proc. ICDM 2003 Workshop Frequent Item set Mining Implementations, (2003).