# A vertical format algorithm for mining frequent item sets

GUO Yi-ming
Institute of Graduate
Liaoning Technical University
Huludao, China
e-mail: lengxin855@163.com

WANG Zhi-jun
College of Electronic and Information Engineering
Liaoning Technical University
Huludao, China
e-mail: lntuwzj@163.com

***Abstract*—Apriori is a classical algorithm for association rules. In order to get the support degree of candidate sets, Apriori needs to scan the database for many times. This paper presents a new algorithm, which mine frequent item sets with vertical format. The new algorithm only needs to scan database one time. And in the follow-up data mining process, it can get new frequent item sets through 'and operation' between item sets. The new algorithm needs less storage space, and can improve the efficiency of data mining.**

***Keywords- Apriori; vertical format; data mining***

## I. INTRODUCTION

Data mining technology is mainly used to process massive amounts of data and information. It can help us find the potential and meaningful knowledge. Association rule mining is one of the most active research methods. It was proposed by Agrawal etc. to analysis market basket question. The purpose is to discover the association rules among different commodities included in trading database.

## II. APRIORI ALGORITHM

### A. Brief overview

Apriori[1] algorithm is a classical algorithm to find association rules which was presented by Agrawal and Srikant in 1993. It is the boolean association rules algorithm to mining frequent item sets. The basic idea of the algorithm is to recursively generate frequent item sets. The basic idea of the algorithm is that it starts with the only *1*-item sets, recursively generates a frequent *2*-item set, and then produces a frequent *3*-item set, and continues like this, until all frequent item sets are produced. The algorithm will stop till generate all the frequent item sets.

### B. Properties

The properties of Apriori algorithm is that an item set is frequent and its all non-empty subsets are frequent. In other words, if an item set is not frequent, all of its super-sets will not be frequent.

### C. Pseudocode

```
Input：D，min_sup
Output：Lk
  L1=find_frequent_1-itemsets (D);
  for(k=2;Lk-1≠Φ;k++){
      Ck=Apriori_gen(Lk-1);
      For each item.Tid∈D{
          Ct=subset(Ck,t);
          for each candidate c∈Ct
              c.count++;
      }
      Lk={c∈Ck | c.count>=min_sup}
}
return L=∪kLk
procedure Apriori_gen(Lk-1: frequent (k-1)-item sets)
    for each item set p1∈Lk-1
        for each item set p2∈Lk-1
            if(p1[1]=p2[1])∧(p1[2]=p2[2])∧...∧(p1[k-
1]=p2[k-1]) then{
              c=p1 connect p2;
              if has_infrequent_subset(c,Lk-1) then
                  delete c;
              else add c to Ck ;
            }
  return Ck;
procedure has_infrequent_subset(c:candidate k-item set;
    Lk-1:frequent (k-1)-item sets)
        for each (k-1)-item set s of c
            If s not in Lk-1 then
                return TURE;
            return FALSE;
```

### D. Algorithm weaknesses

*a)* This algorithm may generate many candidate item sets in the calculation process. When the number of frequent *1*-item set is very big or the frequent patterns are very long, the number of the generated candidate item sets will increase sharply. So the algorithm's efficiency will fall dramatically. For example, if the number of frequent *1*-item set is $10^4$, the number of candidate *2*-item set we need to generate, will be $10^7$. If the length of frequent mode is 100, we will need to generate $2^{100}$ candidate sets. If we search so many candidate sets, the efficiency of the algorithm should be fairly low.

*b)* This algorithm needs to scan the database many times and check candidate item sets by matching pattern. If the database is very large and the patterns needed to be matched are also very long, the efficiency of the algorithm will be greatly reduced.

**978-1-4244-5848-6/10/$26.00 ©2010 IEEE**

## III. IMPROVED ALGORITHM

### A. Brief overview

Apriori is a horizontal format algorithm. It is for mining frequent item sets. It needs to scan the database repeatedly to get support degree of the candidate sets. The time consumption in this process is the key of the algorithm. A variety of improved algorithms proposed to reduce the comparision times between candidate sets and the transaction records. Such as the DHP[2], FP-growth[3] and so on. They all have played a certain role in this regard. Here, we think that if there is a new algorithm can eliminate this comparison process, it will make the performance improved greatly.

This paper presents a new algorithm, which mine frequent item sets with vertical format. The new algorithm only needs to scan database one time to get frequent *1*-item set. In the follow-up of the mining process we statistic support degree of candidate sets which is used to generate table. We needn't to scan database again.

### B. Algorithm Benefits

Benefits of vertical algorithm for mining frequent item sets: It will be able to judge whether the non-frequent item sets before generating candidate item sets. This can save time. Since each TID set of *k*-item set to carry the complete information that can calculate the support degree, so we needn't to scan the database to calculate the support degree of (*k+1*)-item set.

### C. Description of the algorithm

First, scan the database to generate frequent *1*-item set. Second, transform horizontal format of frequent *1*-item set into vertical format. Then do 'and operation' among each element of frequent item set $L_k$ and record the result. If the result is more than the min_sup, we'll obtain a candidate set $C_{k+1}$, else we will do the next 'and operation'. We will stop doing the 'and operation' till the following situations come: there is a frequent item set left and we have no way to do the 'and operation' or all the results of 'and operation' is less than min_sup.

### D. Pseudocode

```
Input：D，min_sup
Output：Lk
  L1=find_frequent_1-itemsets (D);
     For (k=2; | Lk | >1;k++)
     Lk =P(Lk-1,min_sup)
     RETURN ∪kLk
Procedure P(D，min_sup) {
     Lk=null;
     For(i=0；i<Lk-1.count；i++)
        For(j=i+1；j<Lk-1.count；j++){
          If(Lk-1[i].string(k-2)=Lk-1[j].string(k-2))
            Then item.Tid=Lk-1[i].Tid∩Lk-1[j].Tid
          If(item.Tid.length>=min_sup)
            Then Lk.add(item)
        }
  }
```

## IV. ALGORITHM EXAMPLES

### A. Example Process

*a)* Define min_sup=2, scan the database and generate item sets which are vertical data format. They are listed in Table 1.

TABLE I. TO SIMPLIFY THE DATABASE

| Item set | TID set |
|---|---|
| I1 | 1,4,5,7,8,9 (b1) |
| I2 | 1,2,3,4,6,8,9 (b2) |
| I3 | 3,5,6,7,8,9 (b3) |
| I4 | 2,4 (b4) |
| I5 | 1,8 (b5) |

*b)*
b1∩b2={1,4,8,9} FLAG=4
b1∩b3={5,7,8,9} FLAG=4;
b1∩b4={4} FLAG=1<2 delete
b1∩b5={1,8} FLAG=2;
b2∩b3={3,6,8,9} FLAG=4
b2∩b4={2,4} FLAG=2;
b2∩b5={1,8} FLAG=2
b3∩b4=Φ delete;
b3∩b5={8} FLAG=1<2 delete
b4∩b5=Φ delete

*c)* Arrange the above datas to generate *2*-frequent set shown in Table 2.

TABLE II. FREQUENT 2-ITEM SET

| Item set | TID set |
|---|---|
| I1I2 | 1,4,8,9 (b1) |
| I1I3 | 5,7,8,9 (b2) |
| I1I5 | 1,8 (b3) |
| I2I3 | 3,6,8,9 (b4) |
| I2I4 | 2,4 (b5) |
| I2I5 | 1,8 (b6) |

*d)* If the first item in the item set is the same, connect and intersect them.
b1∩b2={8,9} FLAG=2
b1∩b3={1,8} FLAG=2
b2∩b3={8} FLAG=1<2 delete
b4∩b5=Φ delete
b4∩b6={8} FLAG=1<2 delete
b5∩b6=Φ delete

*e)* Arrange the above datas to generate frequent *3*-item set shown in Table 3.

TABLE III. FREQUENT3-ITEM SET

| Item set | TID set |
|---|---|
| I1I2I3 | 8,9 (b1) |
| I1I2I5 | 1,8 (b2) |

*f)* b1∩b2=8 FLAG=1<2 delete

So the maximal frequent item set is{I1，I2，I3}and{I1，I2，I5}.

### B. *Summary*

When frequent *k*-item set to generate frequent (*k+1*)-item set, the mode of intersection of any two sets is used in [4]. Pruning is not used in the paper, thus there are some operations which are not necessary to be done. So when the number of the frequent pattern needed to be found is very big or the amounts of data is very large, the efficiency of the algorithm will be greatly reduced. Pruning is first done in this paper, namely count the number of TID sets. If the number is less than min_sup, we'll delete the corresponding item set. When we do 'and operation' on the candidate sets, we can use apriori_gen algorithm to filter to reduce the counts of 'and operation' and improve efficiency.

## V. EXPERIMENT

We use the classical data set, Mushroom Database as test data. This data set includes 8124 instances, 23 attributes and 127 kinds of values. When the support degree is large, the Apriori algorithm takes little running time. However, the improved algorithm takes less time than it. With the reduction of the support degree, the performance of the improved algorithm will be better.

We use a data set including 8124 instances to do simulation experiment. Figure 1 shows that the time needed by the two algorithms under different support degrees. Figure 2 shows that the performance of two algorithms under different records. The degree of support is 25%.

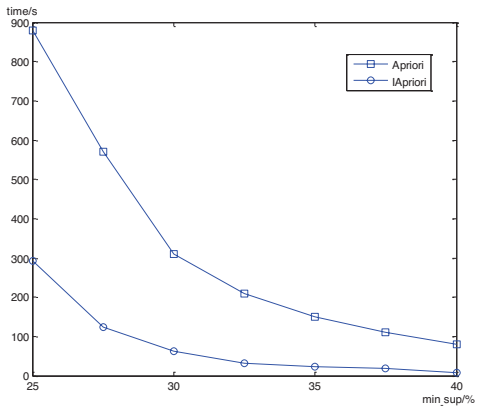


Figure 1. The comparison result of the two kinds of algorithms under different degrees of support

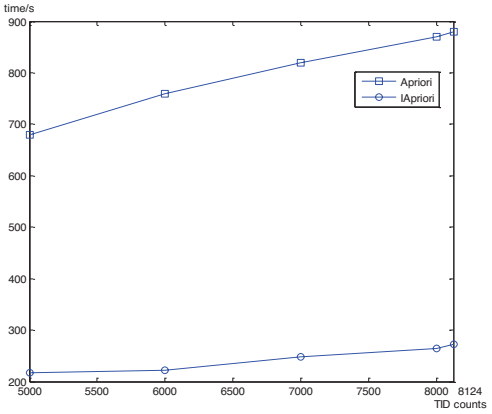


Figure 2. The comparison result of the two kinds of algorithms under different TID counts

Experimental results show that the efficiency of the improved algorithm is much better than Apriori algorithm.

## REFERENCES


[1] Margaret H.Dunham, Data Mining Itroductory and Advanced Topics,Tsinghua University Press,2005 pp.145–155.

[2] Jiawei Han, Micheline Kamber, Data Mining : Concepts and Techniques, 2nd ed. China Machine Press, 2006, pp.155–160.

[3] Song Jingjing , Mining Maximal Frequent Patterns in a Unidirectional FP-tree, Henan University,Henan Zhengzhou, May 2007.

[4] Wang Cuiru, Wang Shaohua, An Improved Apriori Algorithm for Association Rules. Computer Technology and Applications, February 2008.