Xudong LIU

7lxd@tongji.edu.cn

School of Software Engineering

Tongji University

Dec 22, 2015

ASSOCIATION RULE &APPLICATION WITH AMAZON

ABSTRACT

Association rule learning is a method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness. The association rules is widely used in many E-business company. Amazon is one of the most successful E-business company and the pioneer of recommendation system. Therefore, this paper is trying to find the performance of association rules now and if we can improve the association rule algorithm and what will an ideal association rules should be according to Amazon's association rules algorithm.

Keywords: Data Mining,Association rules, Amazon recommendation system, A priori Algorithm, DHP Algorithm, PHP Algorithm.

INTRODUCTION

Association Rules:

Association rule learning is one of most popular tools in data mining sector to discover the internal connection between the large transaction data. Association rule learning is wildly used in recommendation system, we can see the association rule learning's application in many electronic commerce company's website. The common algorithm for association rules are A priori algorithm,DHP and PHP. Amazon is one of the biggest electronic commerce company and have the most successful recommendation system.

Amazon:

Amazon.com, Inc. is an American electronic commerce and cloud computing company with headquarters in Seattle, Washington. It is one of the most largest electronic commerce in the world and the pioneer of recommendation system and have the most mature association rule learning technology.

ALGORITHM IMPROVEMENT

2.1 A priori Algorithm:

The A priori algorithm is probably the best known algorithm in association rule learning. The algorithm include two steps: firstly, find all item sets that have minimum support; secondly, use the frequent item sets to generate rules. The core idea of a priori algorithm is that any subsets of a frequent item set are also frequent item sets.

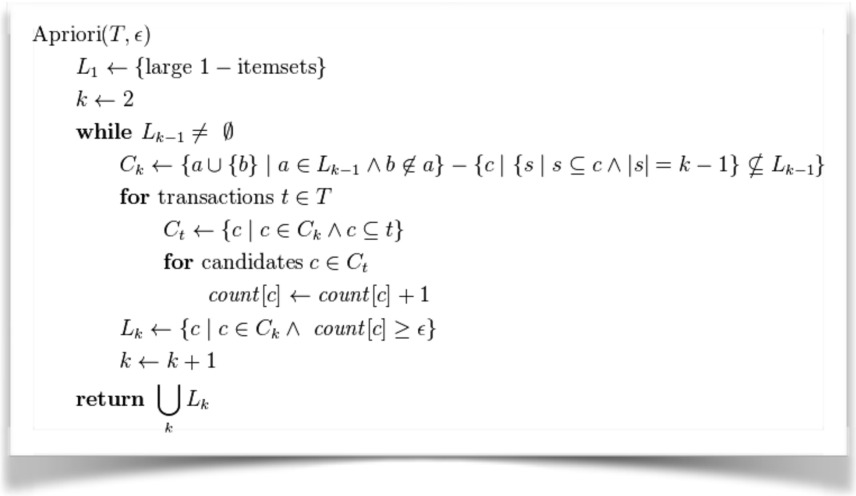


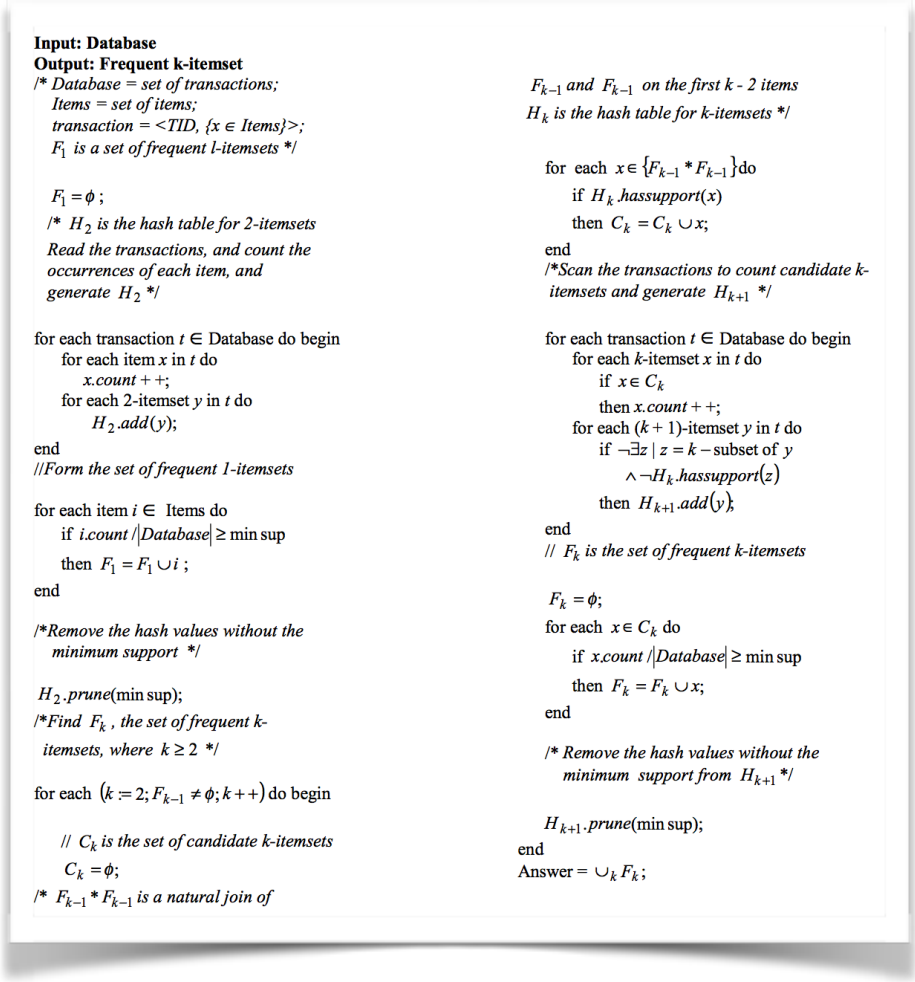
Figure 2.1

The most limitations of A priori algorithm is that the algorithm lack of efficiencies, the bottom-up subset exploration finds any maximal subset S only after all 2^|S|-1 of its proper subsets. However, just because of the limitations of Apriori algorithm, it has spawned other algorithms.

2.2 DHP Algorithm(Direct Hashing and Pruning Algorithm):

DHP varies from the Apriori algorithm in the sense that it uses hashing technique to remove the unwanted itemsets for the generation of next set of candidate itemsets. It accumulates information about (K+1) itemsets in advance in such a way so that all possible (K +1) itemsets of each transaction after some pruning are hashed into a hash table. Each bucket in the hash table consists of the count of itemsets that have been hashed into the bucket so far.

Limitations of DHP is that with DHP, it generates a number false positives. Therefore, PHP is another algorithm which is suggested as an improvement over DHP.



Input: Database
Output: Frequent k-itemset
/* *Database = set of transactions;*
Items = set of items;
transaction = <TID, {x ∈ Items}>;
F_1 is a set of frequent l-itemsets */

$F_1 = \phi$;
/* *H_2 is the hash table for 2-itemsets*
Read the transactions, and count the occurrences of each item, and generate *H_2* */

for each transaction $t \in$ Database do begin
  for each item x in t do
    $x.count++$;
  for each 2-itemset y in t do
    $H_2.add(y)$;
end
//*Form the set of frequent 1-itemsets*

for each item $i \in$ Items do
  if $i.count / |Database| \geq$ min sup
  then $F_1 = F_1 \cup i$;
end

/**Remove the hash values without the minimum support* */

$H_2.prune$(min sup);
/**Find F_k, the set of frequent k-itemsets, where $k \geq 2$* */

for each $(k := 2; F_{k-1} \neq \phi; k++)$ do begin

  // *C_k is the set of candidate k-itemsets*
  $C_k = \phi$;
/* *$F_{k-1} * F_{k-1}$ is a natural join of F_{k-1} and F_{k-1} on the first k - 2 items*
H_k is the hash table for k-itemsets */

  for each $x \in \{F_{k-1} * F_{k-1}\}$ do
    if $H_k.hassupport(x)$
    then $C_k = C_k \cup x$;
  end
  /**Scan the transactions to count candidate k-itemsets and generate* *H_{k+1}* */

  for each transaction $t \in$ Database do begin
    for each k-itemset x in t do
      if $x \in C_k$
      then $x.count++$;
    for each $(k+1)$-itemset y in t do
      if $\neg\exists z \mid z = k-$subset of y
        $\wedge \neg H_k.hassupport(z)$
      then $H_{k+1}.add(y)$;
  end
  // *F_k is the set of frequent k-itemsets*

  $F_k = \phi$;
  for each $x \in C_k$ do
    if $x.count / |Database| \geq$ min sup
    then $F_k = F_k \cup x$;
  end

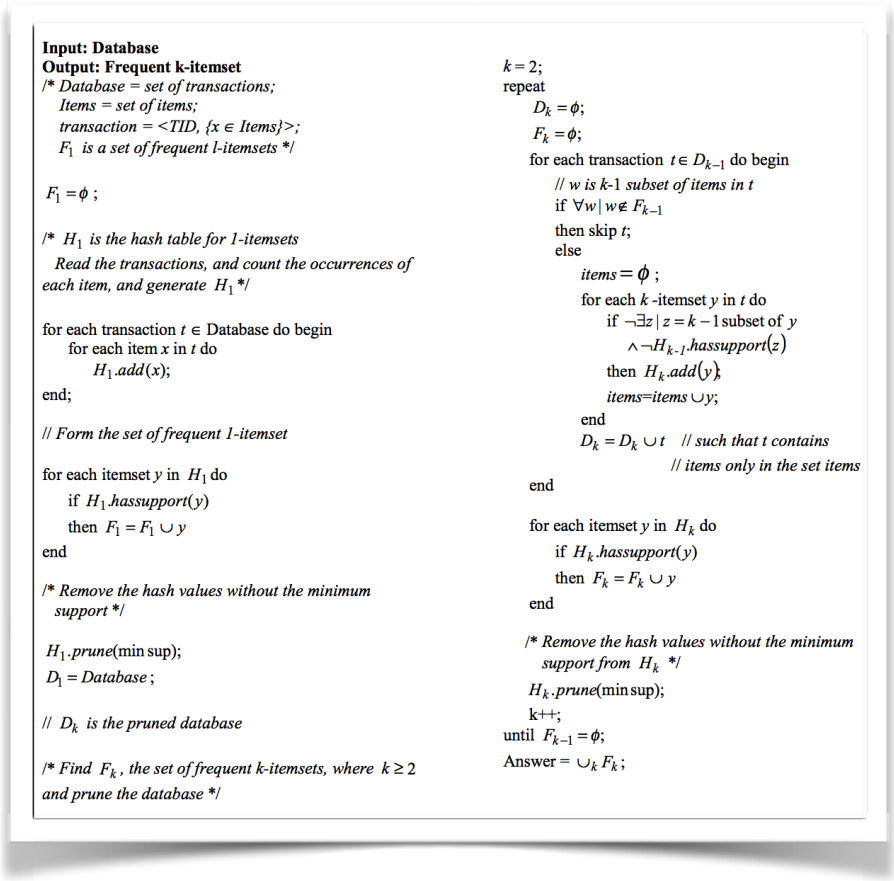
  /* *Remove the hash values without the minimum support from* *H_{k+1}* */

  $H_{k+1}.prune$(min sup);
end
Answer = $\cup_k F_k$;

Figure 2.2

2.3 PHP Algorithm(Perfect Hashing and Pruning Algorithm):

In PHP algorithm, during the first pass the hash table with size equal to the number of individual items in the database is created. Then each individual item is mapped to different item in the hash table. After the first pass the hash table consists of exact number of occurrences of each item in the database. Then each item in the database. Then the algorithm removes all the transactions which have no items that are frequent and also trims the items the items that are not frequent. Simultaneously, candidate k-itemsets are generated and the occurrences of each pass, the algorithm gives a pruned database, occurrences of candidate k-itemsets and set of frequent k-itemsets and the algorithm terminates when no new frequent k-items are found. Obviously, PHP's performance is better than DHP with no false positives and extra time required for counting frequency of each item set is eliminated.



```
Input: Database
Output: Frequent k-itemset
/* Database = set of transactions;
   Items = set of items;
   transaction = <TID, {x ∈ Items}>;
   F1 is a set of frequent l-itemsets */

F1 = ϕ ;

/* H1 is the hash table for 1-itemsets
   Read the transactions, and count the occurrences of
each item, and generate H1 */

for each transaction t ∈ Database do begin
    for each item x in t do
        H1.add(x);
end;

// Form the set of frequent 1-itemset

for each itemset y in H1 do
    if H1.hassupport(y)
    then F1 = F1 ∪ y
end

/* Remove the hash values without the minimum
   support */

H1.prune(min sup);
D1 = Database ;

// Dk is the pruned database

/* Find Fk , the set of frequent k-itemsets, where k ≥ 2
and prune the database */

k = 2;
repeat
    Dk = ϕ;
    Fk = ϕ;
    for each transaction t ∈ Dk−1 do begin
        // w is k-1 subset of items in t
        if ∀w | w ∉ Fk−1
        then skip t;
        else
            items = ϕ ;
            for each k -itemset y in t do
                if ¬∃z | z = k − 1 subset of y
                    ∧ ¬Hk-1.hassupport(z)
                then Hk.add(y);
                items=items ∪ y;
            end
            Dk = Dk ∪ t    // such that t contains
                           // items only in the set items
    end

    for each itemset y in Hk do
        if Hk.hassupport(y)
        then Fk = Fk ∪ y
    end

    /* Remove the hash values without the minimum
       support from Hk */
    Hk.prune(min sup);
    k++;
until Fk−1 = ϕ;
Answer = ∪k Fk ;
```

Figure 2.3

2.4 Conclusion:

Comparing the three algorithm above, I can safely come to the conclusion that the PHP algorithm perform better in both efficiency and accuracy. However, we have to admit that only with the keeping exploration in Apriori and DHP algorithm can we create the PHP algorithm.

I will recommend the Amazon Inc adopt the PHP algorithm instead of the Apriori and DHP algorithm.

RARE ITEM PROBLEM & SOLUTION

According to the Amazon and other big electronic commerce, we can find the problem that: some items just can't become the frequent item forever. With analysis of the data and the fact, it's not difficult to find the reason why the problem happen is that different items have different accessing time, using one min sup is clearly unfair for different items. Therefore, I think the system can use multiple minimum support to solve the problem above.

To use multiple minimum support, we have to give every item one specific minimum support, what's more, each iteration has to every item with their unique minimum support to justify if the item is frequent. In this way, we can get more real frequent itemsets with our

association rule algorithm because we can find the real frequent item in each kinds of items.

CONCLUSION

We can come to the conclusion as follows:

- To improve the amazon current association rule algorithm performance, we recommend PHP(perfect hashing and pruning) algorithm, according to the reasons mentioned above, we can easily come to the conclusion that the PHP algorithm perform better in efficiency and accuracy. Therefore, we recommend Amazon to use PHP algorithm instead of DHP & Apriori algorithm to improve the algorithm's performance.
- I have to admit that Amazon is the pioneer of the recommendation system and may have the most advanced association rules technology. However, I still think the system has the rare item problem. I tried to give the solution about the rare item problem by using multiple minimum support.
- I think the ideal association rule algorithm is that the algorithm not only can recommend the "similar" items but also can analysis the person who are browse the item and give the most needed items the specify person want to buy.
- The difficulties of the algorithm are as follows: fist of all, because of the algorithm has to be done in the limit time, therefore, we have to consider the efficiency of the algorithm. Secondly, the cold launch problem and the accuracy improvement should be considered in the enterprise level application.
- What's more, the association rules is just one of many recommendation technologies, therefore, I believe we can use other technologies such like common friends algorithm based on the items to improve the whole recommendation system's performance.

REFERENCE:

[1] Wikipedia,Association rule: https://en.wikipedia.org/wiki/Association_rule_learning
[2] S.Ayse Ozel, H.Altay Guvenir, "An Algorithm for Mining Association Rules Using Perfect Hashing and Database Pruning" : http://www.cs.bilkent.edu.tr/~guvenir/publications/TAIN-N01-AOAG.pdf
[3] Bing Liu, Wynne Hsu, Yiming Ma, "Mining Association Rules with Multiple Minimum Supports": http://sse.tongji.edu.cn/wrao/mining/paper-ar.pdf
[4] The blog about PHP algorithm: http://associationrule.blogspot.jp/2008/09/perfect-hashing-and-pruning-algorithm.html