International Journal of Soft Computing 3 (3): 205-211, 2008
ISSN: 1816-9503
© Medwell Journals, 2008

A Novel Scheme for Candidate Generation for Mining Frequent Patterns

[1]P.C. Saxena, [2]Asok De and [2]Rajni Jindal
[1]School of Computer and System Sciences, Jawaharlal Nehru University, Delhi, India
[2]Delhi College of Engineering, Delhi, India

Abstract: With the explosive growth of data, mining information and knowledge from large databases has become one of the major challenges for data management and mining community. Data mining is the extraction of hidden unpredictive information from large databases. It is concerned with the analysis of data and finding patterns that exist in large databases but are hidden among the vast amount of data. Association rules are one of the most popular data mining techniques. The first step in mining association rules is mining frequent patterns. They are particularly useful for discovering relationships among data in huge databases. This study proposes a novel scheme for candidate generation that generates all the candidate item sets in three iterations. A new algorithm called AR-mine for association rule mining is also presented that uses the proposed scheme for candidate generation. A distinct feature of this algorithm is that a candidate item set is generated only when it actually encounters an occurrence of that set in the database. Another important feature is that it requires only three scans of the database. A simple hash table is used to store the candidate item sets, which speeds up the searching process. Our experiments with synthetic data sets and real life data sets show that AR-mine performs better than apriori, a well known and widely used algorithm for association rule mining.

Key words: Data mining, association rules, candidate item set, frequent patterns

INTRODUCTION

Databases have been used in business management, government administration, scientific and engineering data management and many other important applications. In order to extract potentially useful information from large databases, a process known as data mining is used. Data mining allows nontrivial extraction of implicit, previously unknown information in terms of patterns or rules from vast amount of data. This newly extracted information or knowledge may be applied to information management, query processing, process control, decision making and many other useful applications. In addition to the existing areas, the widely used new areas like world wide web also need to use various data mining techniques to understand user behavior and his data requirements in a better way in order to provide better service and to increase business opportunities (Chen *et al*., 1996).

The typical data mining techniques are classified as association, sequencing and classificatory. Classificatory algorithms partition output data into disjoint groups. The results of this classification might be represented as a decision tree. The sequencing algorithms find items or events which are related in time such as events A and B are usually being followed by an event C. Association algorithms find sets of items that appear together in sufficient frequency to merit attention. The problem of mining associations from data has received a great deal of attention.

The problem of mining for association rules was first introduced in Aggarwal *et al*. (1993). An association rule is a rule that implies certain association relationships among a set of objects in a database. Given a set of items and a large collection of transactions, the task is to discover the important associations or relationships among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction.

ASSOCIATION RULES

Given a set of transactions, an association rule is an expression $X \Rightarrow Y$, where X and Y are subsets of A. Here $A = \{I_1, I_2, \ldots, I_m\}$ is a set of items. Each transaction T is also a set of items. Such a rule means that the transactions in the database which contain the items in X also tend to contain the items in Y. Each association rule has a left hand side known as antecedent and a right hand side called consequent. Both antecedent and consequent can contain multiple items. The rule $X \Rightarrow Y$ holds with confidence c, if c % transactions in the database that support X also support Y. The rule $X \Rightarrow Y$ has support s

Corresponding Author: P.C. Saxena, School of Computer and System Sciences, Jawaharlal Nehru University, Delhi, India in the transaction set T if s% of transactions in T support X U Y. Support measures how often X and Y occur together as a percentage of the total transactions and confidence measures how much a particular item is dependent on other.

PROBLEM STATEMENT

The problem of mining association rules is, essentially, to discover all rules, from a given set of transactions of database D that have support and confidence greater than or equal to the user specified minimum support and minimum confidence. The problem of association rule mining is usually broken down into two sub problems.

- To find all the sets of items whose support is greater than or equal to the user specified minimum threshold $\propto$. Such item sets are also called frequent item sets or frequent patterns in the literature. An item set x is said be a frequent item set in T with respect to α, if $s(X)_T \geq \alpha$.

In example 1, {E, rice} is a frequent item set, if we assume $\alpha = 50\%$, as it is supported by 4 out of 8 transactions. We may note that the set of frequent sets for a given T, with respect to a given α, shows following properties.

Downward closure property: Any subset of a frequent set is a frequent set.

Upward closure property: Any superset of an infrequent set is an infrequent set.

- To generate the association rules desired from the frequent item sets. Let A U B and A are frequent item sets, then we can say that the rule A ⇨ B holds if the ratio of support (A U B) to support (A) is greater than or equal to the minimum confidence threshold β. i.e.,

$$\frac{s(A \text{ U } B)}{s(A)} \geq \beta$$

Here s(X) is the support of X in T. We may note that the rule will have minimum support because A U B is frequent.

The database D is considered as a Boolean relational table. Each row in the table corresponds to a tuple and each column corresponds to an attribute. The Ith entry in a row contains T or F depending on whether attribute I is present in the corresponding tuple or not. This table can be physically organized in a horizontal manner in which each tuple is represented as a pair (tno, itemset) where tno is the tuple number and the item set contains a sequence of attributes. An attribute is included in this sequence if its corresponding value for this tuple in the database is T. The pair (tno, itemset) is also referred to as a transaction.

tno	a	b	c	d	e	tno	itemset
1	T	T	F	T	T	1	abde
2	F	T	T	F	T	2	bce
3	T	T	F	T	T	3	abde
4	T	T	T	F	T	4	abce

Database relational table   Physical representation

Given a set of distinct attributes, we can represent any subset of A as a sequence which is sorted according to the lexicographic order of attribute names. For example, {1,2} and {2,1} represent the same subset of {1,2,3}, which is identified by the sequence 12. The transactions in database D and the item sets in the candidate sets and the frequent sets are all in the lexicographic order of the attribute names.

Most of the research (Aggarwal *et al*., 1993; Houstsma and Swami, 1995; Aggarwal and Srikant, 1994; Park *et al*., 1995; Han *et al*., 2000; Sergery *et al*., 1997; Berzal *et al*., 2001) has been focussed on the first subproblem because the overall performance of mining association rules is determined by the first step which accesses the database. Discovering all frequent patterns and their supports is a non-trivial problem if the database is large.

As the underlying databases are large and the algorithms for mining association rules may require multiple passes over the database, to find different association rules, the algorithms are required to reduce I/O operations and the run time to enhance the efficiency.

This problem can be solved by constructing a candidate set of potentially frequent patterns and identifying the frequent patterns within these candidate sets.

PRIOR WORK

Several algorithms have been proposed in the literature to solve this problem. Some of them are AIS (Aggarwal *et al*., 1993), SETM (Houstsma and Swami, 1995), Apriori algorithm, AprioriTid, Apriorihybrid (Aggarwal and Srikant, 1994), Direct Hashing and pruning (Park *et al*., 1995), Partition algorithm, Pincer-Search algorithm, FP-tree Growth algorithm (Han *et al*., 2000), dynamic itemset counting algorithm (Sergery *et al*., 1997) and TBAR (Berzal *et al*., 2001). These algorithms, in general except the FP tree growth algorithm, first construct a candidate set of frequent itemsets based on some heuristic and then discover the subset that actually contains frequent itemsets. The frequent itemsets found in one iteration are then used as the basis to find the candidate set for the next iteration.

Heuristic to construct the candidate set of frequent itemsets is crucial to performance because the cost of processing to discover the frequent itemsets increases as the size of the candidate set increases. The heuristic should only generate candidates that are likely to be frequent itemsets because for each candidate, we need to count its appearances in all transactions. Another performance related issue is the amount of data that has to be scanned during frequent itemset discovery. Reducing the number of transactions to be scanned and trimming the number of items in each transaction can improve the data mining efficiency.

Apriori algorithm (Aggarwal and Srikant, 1994) is a landmark in association rule mining. This is the most popular algorithm to find all the frequent sets. It makes multiple passes over the data. To reduce the combinatorial search space, this algorithm makes use of the downward closure property and at each pass prunes many of the sets which are unlikely to be frequent sets.

We may note that the apriori algorithm operates in a bottom up, breadth first search method. The process starts from the smallest set of frequent item sets and proceeds upward till it reaches the largest frequent item set. The number of times the database is scanned is same as the size of the largest frequent item set.

This algorithm uses an efficient candidate generation method. It uses only the frequent item sets at a level to construct the candidates at the next level. But it requires one pass over the database of all transactions for each iteration. Therefore, the number of passes required is as many as the longest item set.The hash based technique DHP tries to reduce the size of the candidate k-item sets by collecting approximate counts in the previous level. Another version of this algorithm tries to reduce the number of transactions to be scanned at higher values of k. but the number of passes over the database remains as in apriori algorithm. Since these algorithms mine sequential patterns by scanning the database multiple times, the CPU and I/O times increases with the increase in the size of the database.

In AprioriTid, a modification of apriori algorithm, reading the complete database after the first iteration is avoided. The transaction id's and candidate frequent k-item sets present in each transaction are generated in each iteration which is used to determine the large (k+1) - item sets present in each transaction during the next iteration.

Research (Berzal *et al.*, 2001; Zaki, 2000) has shown that in the initial stages, apriori is more efficient than aprioriTid as there are too many candidate k-itemsets to be tracked during the early stages of the process. However, for the later stages aprioriTid is better than apriori. Another algorithm apriorihybrid, a hybrid of these two algorithms shows better performance, in general. It has the option of switching from apriori to aprioriTid after early passes for better performance but it is difficult to determine the switch over point to operate the hybrid algorithm.

The partition algorithm partitions the database into small parts such that each part can be managed in the memory. It generates the set of all potential and local item sets in the first pass and counts their global support in the second pass. However, it is quite possible, that item sets which are locally frequent in some partition may not be globally frequent. Therefore, partition may enumerate so many false candidates in the first pass. Also, if the local frequent set does not fit into memory, additional database scans will be required.

The DIC algorithm is able to reduce the number of scans over apriori by dynamically counting the candidates of various lengths as the database progresses. But it gives good performance only if the data is distributed fairly uniformly and stop points are chosen reasonably close.

The DLG algorithm uses a bit vector per transaction. It writes the Tids where the item set has occurred. Then, it generates the frequent item sets by performing logical AND operations on the bit vectors. This algorithm assumes that the bit vectors fit in the memory. But databases are generally very large containing millions of transactions. Therefore, scalability could be a problem.

TBAR algorithm uses an efficient data structure to represent the sets of candidate and frequent item sets. It requires less memory and produces less memory fragmentation. It is faster than apriori. But the number of passes over the database remains same as in apriori algorithm.

PROPOSED SCHEME FOR CANDIDATE GENERATION

The proposed scheme generates all the candidate item sets in 3 iterations irrespective of the length of the longest frequent item set. In the first iteration, it generates candidate item sets and frequent item sets of length 1. In the second iteration, it finds candidate item sets and frequent item sets of length 2 and in the third iteration, it finds all the remaining candidate and frequent item sets of all lengths upto the longest frequent item set. Now we present our new algorithm AR-mine (Association Rule mining) which uses the proposed scheme of candidate generation for mining association rules.

AR-mine: AR-mine requires only 3 scans of the database at the most. In the first scan , it counts the support of all the length 1 itemsets and finds all length 1 itemsets which are frequent. In the second step it finds all the combinations of length 2 for each transaction in the database D and inserts in the C2. A combination is included in C2 only if it follows the upward closure property i.e. the support of both the items constituting the length 2 itemset are above threshold. To insert a combination in C2, it checks whether that particular combination is already there in C2 or not. If it is present, then it increases the support count of that item set otherwise, it inserts it in C2 and sets its support count to one. The pseudo code is given in Fig. 1.

AR-mine generates the candidate set C_2 and counts its support simultaneously in the second scan. This algorithm generates candidate set from the transactions of the database D, unlike *apriori* which generates C_i from L_{i-1}, thus reducing the size of the candidate sets drastically. This step ensures that an item set is included in the candidate set only if it is actually present in a transaction T of the database D.

Once L_2 is generated, our algorithm generates C_3 form L_2 . At this stage, to avoid multiple database scans, AR-mine generates C_k from C_{k-1} instead of L_{k-1} for all k > 3 using gen function (Fig. 2). Function gen takes as input a 2 dimensional array C_k which is a candidate set of item sets of length k and generates the set C_{k+1} which is the candidate set of item sets of length k+1. Ck though is a 2D array is addressed as a linear array. Any element C[i][j] is addressed as C[i*k+j] where k is the number of columns in 2D array. Our algorithm prunes the candidate item sets by checking for upward closure property i.e. an item set is included in C_k if all its subsets are included in C_{k-1}. Therefore, once C_{k+1} is formed, the algorithm checks whether all subsets of the candidate item set are in C_k or not. The pseudo code for the pruning step is given in Fig. 3. At the end of this step, we have candidate sets C_3, C_4, C_5, Now, our algorithm scans the database for the third and last time and counts the support for all candidate sets C_k for k >= 3.

After the third scan, we have support count of all candidate sets C_k for k >= 3. The items of C_k whose support count is greater then the threshold are included in L_k. At this stage, we have found all the frequent item sets L_j for j >= 1 from which, we can generate the association rules.

```
for each transaction t in D do
for each item t[i]  in t do
    if  t[i] is frequent length 1 item then
          for(k=i+1;k<t_len;k++)
                  if t[k] is frequent length 1 item
                     combination =  ( t[i], t[k] )
                  if combination already in  C2
                          C2[j].support ++
                  else C2[j].item = combination
                        C2[j].support = 1;
 L2 = null
for each item set c in C2 do
if c.support > threshold
  L2 = L2 U c
```

Fig. 1: Pseudo code for second scan

```
 for (k = 3; Ck = Φ; k++) do
    Ck+1=NULL;
    for each item  i of  Ck do
        for each item  j = i + 1 of Ck do
    flag=1;
    for(int z=0;z<k-1;z++)
       if( Ck[ i*k+z ] != Ck[ j*k+z ] )
          flag=0;
       else
          Ck+1[z]=Ck[i*k+z];
     end
     if(flag)
       Ck+1[k-1]=Ck[i*k+(k-1)];
          Ck+1[k]=CK[j*k+(k-1)];
     end
```

Fig. 2: Pseudo code for gen function

```
for each itemset c of Ck do
               for each subset s of c do
                    if not (s in Ck-1) then
                         Ck = Ck – c
               end
end
```

Fig. 3: Pseudo code for pruning

In order to do an efficient search, AR-mine uses a hash based dynamic array to store the candidate item sets. The hash function used is

$a_1^2 + a_2^2 +\ldots + a_n^2$ % number of candidate item sets

Here, { $a_1, a_2,\ldots,a_n$} is an item in the candidate item set whose support is being counted. In case of a collision, this algorithm puts a pointer at the hashed address. This pointer points to a link list which contains the items that are hashed to this address. We will use the example 3 to illustrate our algorithm.

Example 1: Suppose that after some preprocessing, we obtain the simple horizontally organized database as shown in Table 1. This database contains 10 transactions and 10 items per transaction. Further, let us set ∝ = 30% which means 3 tuples out of 10 must support the frequent item sets.
After the first scan we get

L_1 = { I1, I2, I3, I5, I6, I7, I8, I9, I10}

I4 is not included in L_1 as its support count is less than 3. Now instead of taking the natural join of L_1 with

Table 1: Sample data

tno	Transaction
1	3 5 7 9 10
2	1 2 3 4 5 6
3	1 4 6
4	1 3 5 6 8
5	3 9 10
6	1 2 3 5 6
7	3 5 6 8
8	1 3 7 9 10
9	1 7 9 10
10	1 2 6 8

itself to get C_2, AR-mine scans the database for the second time to form the length 2 item sets from the tuples in the database itself. So after the second scan we get

L_2= { (I1, I2),(I1, I3), (I1, I5), (I1, I6), (I2, I6), (I3, I5), (I3, I6), (I3, I9), (I3, I10), (I5, I6), (I6, I8), (I7, I9), (I7, I10), (I9, I10) }

If we follow apriori technique, C_2 includes item sets like (I6, I7), (I6, I9), (I6, I10), (I8, I9), (I8, I10) etc. which do not occur even once in the database. Our algorithm prunes all these item sets as we include an item set only when it actually occurs in the database.
Now AR-mine finds C_3

C_3 = { (I1, I2, I6), (I1, I3, I5), (I1, I3, I6), (I1, I5, I6), (I3, I5, I6), (I3, I9, I10), (I7, I9, I10) }

We may note that (I1, I2, I3) is not included in C_3 as its subset (I2, I3) is not a frequent item set. Now from C_3, using upward closure property, we find C_4 and then C_5.

C_4 = { (I1, I3, I5, I6) }
$C_5 = \Phi$

We now scan the database for the last time to get the support of the candidate item sets in C_3 and C_4. At the end of the third scan we get

L_3 = { (I1, I2, I6), (I1, I3, I5), (I1, I3, I6), (I1, I5, I6), (I3, I5, I6), (I3, I9, I10), (I7, I9, I10) }
L_4 = { (I1, I3, I5, I6) }
$L_5 = \Phi$

So, in the three scans, our algorithm could find all the frequent item sets. We can find all the frequent items in two scans also, but in that case the number of candidate sets generated will be very large and there will be large memory required to store all those candidate sets.

RESULTS

To evaluate the efficiency of AR-mine, we have performed experiments on a 1.13 Ghz, P3 machine with 128 MB RAM and 40 GB hard disk running Microsoft Windows. AR-mine and apriori are implemented by us using Visual C++ 6.0 package. To compare the results in the same runtime environment, we have implemented apriori algorithm on the same machine to the best of our knowledge based on the published work, as different machine architectures may differ greatly on the absolute runtime for the same algorithm.

We have performed experiments on the real datasets as well as synthesized datasets. Both AR-mine and apriori have been applied to various datasets. The real datasets used are Mushroom dataset and chess dataset. The results of the experiments are presented in Table 2.

Mushroom dataset: This is a dataset taken from the UCI Machine learning Database Repository at http://www.ics.uci.edu/~mlearn/MLRepository.html. It includes descriptions of 8124 hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous or of unknown edibility and not recommended. The last class was combined with the poisonous one. The samples have 22 predictor attributes and 2480 missing values.

Chess dataset: This dataset is taken from www.cs.rpi.edu/~zaki. This dataset includes description of 3196 hypothetical transactions. The samples have 74 predictor attributes.

We have used the synthetic databases that are used as benchmark databases for many association rule algorithms (Aggarwal and Srikant, 1994; Park *et al*., 1995; Han *et al*., 2000; Sergery *et al*., 1997; Zaik, 2000). We have taken them from www.cs.rpi.edu/~zaki. These databases mimic the transactions in the retailing environment in which customers tend to buy sets of items together. Each such set is potentially a maximal frequent itemset. A transaction may contain more than one frequent item set. Transaction sizes are typically clustered around a mean and a few transactions have number of items. Typical sizes of large item sets are also clustered around a mean with a few frequent item sets having a large number of items.

In this synthetic data D denotes the number of transactions, T denotes the average transaction size, I denotes the maximal potentially frequent itemset and N denotes the number of items. We have done experiments on data sets T10I4D100K and T40I10D100K. Both data sets contains 1,00,000 transactions. Each transaction contains 1000 items in the data sets. In the data set T40I10D100K, the average longest frequent item set is 10 items and the average transaction size is up to 40 items. It is a relatively dense data set. The results of the experiments are presented in Table 3.

Table 2: Experimental results for real data sets

Database	No. of transactions	Threshold support ∝	No. of frequent item sets	Timing (in sec)
Mushroom database	8124	1000	123277	615.01
		1500	56693	243.37
		2000	6623	20.53
		2500	2365	7.01
Chess database	3196	2000	166580	1530.12
		2250	45862	224.15
		2500	11493	41.96

Table 3: Experimental results for synthetic data sets

Database	No. of transactions	Threshold support ∝	No. of frequent item sets	Timing (in sec)
T10I4D100K	1,00,000	250	7703	27.41
		500	1073	4.22
		750	561	3.08
		1000	385	2.36
		1500	237	1.82
		2000	155	1.08
T40I10D100K	1,00,000	1000	65236	2134.00
		1500	6539	950.00
		2000	2293	190.00
		2500	1221	55.69

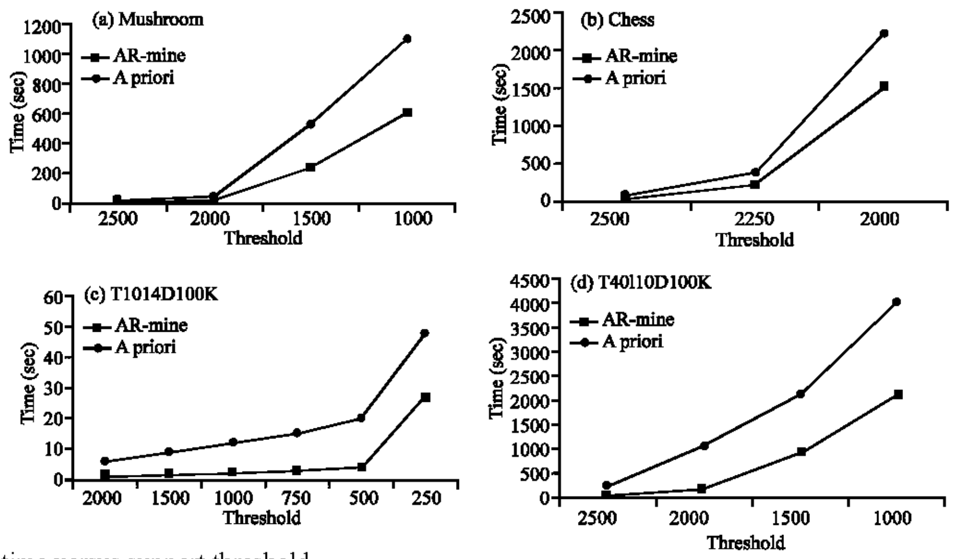

Fig. 4: Run time versus support threshold

We have compared AR-mine with apriori for increasing values of ∝ on different data sets. As ∝ increases, the number of frequent item sets decreases and the time for finding the frequent item sets also decreases. Experiments show that AR-mine performs better than apriori in all cases (Fig. 4).

CONCLUSION

In this study, we have proposed a new scheme for candidate generation that generates all the candidate item sets in three iterations. We have also presented an algorithm AR-mine which uses this new scheme for candidate generation for mining frequent item sets. A simple hash table is used to store the frequent item sets which speeds up the searching process. A distinct feature of this algorithm is that a candidate set is generated only when we actually encounter an occurrence of that set in the database. It is shown by the experiments that AR-mine has better performance than apriori on real datasets as well as synthetic datasets with very limited space overhead. This algorithm finds all the frequent item sets of any length in 3 database scans. AR-mine can be scaled upto very large databases due to limited run time memory overhead.

REFERENCES

Aggarwal, R., T. Lmielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. ACM SIGMOD.

Agrawal, R. and R. Srikant, 1994. Fast algorithms for mining association rules in large databases. Proc. 20th Int. Conf. Very Large Databases, 478-499.

Berzal, F., J. Gubero, N. Morin and J. Serrano, 2001. TBAR: An efficient method for rule mining in relational databases. IEEE Transactions on Knowledge and Data Engineering, pp: 47-64.

Chen, M.S., J. Han and P.S. Yu, 1996. Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering, 8(6): 866-883.

Han, J., J. Pei and Y. Yin, 2000. Mining frequent patterns without candidate generation. Proceedings of ACM-SIGMOD, International Conference on Management of Data, pp: 1-12.

Houstsma, M. and A. Swami, 1995. Set oriented mining of assosiation rules in relational databases. International Conference on Data Engineering, IEEE.

Park, J.S., M.S. Chen and P.S. Yu, 1995. An effective hash based algorithm for mining association rules. Proceedings of ACM-SIGMOD, International Conference on Management of Data, pp: 175-186.

Sergey, B., R. Motwani, T. Dick and J. Ullman, 1997. Dynamic item set counting and implication rules for market basket data. ACM SIGMOD.

Zaki, M., 2000. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12(3): 372-390.