ITEM-BASED PARTITIONING APPROACH OF SOYBEAN DATA FOR ASSOCIATION RULE MINING

G. Pradeepini¹ and S. Jyothi²

Abstract: The advance in computing and information storage has provided vast amount of data. Association rules are mostly used in mining transaction data to find interesting relationship between attribute values and it is a core topic of data mining. It is difficult task as size of data increases. There is a greatest challenge in candidate generation for large data with low support threshold. In this paper we proposed an approach which allows data to be partitioned based on items. It organizes data into tree structure. Experimental results show that the method works well for increasing dimension of data. Performance of our approach is significantly better than other approaches, especially with the dense data and low support threshold.

Keywords: Association Rules, Apriori, Partition Algorithm, Item-based Partitioning

1. INTRODUCTION

Association rule mining discovers correlations among data items in a transactional database. Each transaction in D is a set of data items. Association rules are usually represented in the form AB, where A and B are item sets, i.e., sets of data items. Association rule mining involves the discovery of rules that satisfy defined threshold from tabular database. For this the most fundamental concern is frequency, i.e. how often the rule occurs in the database. The support of rule is how often A and B occurs together as a percentage of the total transactions. The frequent sets are those for which the support exceeds some threshold value. Association Rule Mining is the process of finding frequent sets with minimum support and confidence.

In the process of Mining Association Rules, Frequent sets generation from candidate itemsets is the most time consuming aspect. It is called support counting phase. It is not possible to determine the candidate itemsets in advance so it is necessary to consider many itemsets that are not in frequent. In finding frequent sets many algorithms need one or more passes over the database. The performance of these algorithms depends both on sizes of original database and on the number of candidate itemsets being considered. The number of possible candidates increases with increasing density of data and with decreasing support thresholds.

Performance will be affected, if the magnitudes involved make it impossible for the algorithm to process entirely within primary memory. In these cases, some techniques for partitioning the data may be required to enable the algorithm to reside original data on primary memory. Effective partitioning will reduce the number of accesses to secondary memory. There are two types of partitions i.e. Record-based partitioning (RBP) and Item-based partitioning (IBP), RBP divides the source data into sets of records, and IBP partitions records into sets of items. In this paper we examine a novel method of IBP Experimental results shows this method offers significantly better performance than others.

The remainder of this paper organized as follows. Section 2 surveys related work. Section 3 introduces the Item-based Partitioning Approach. The experiments, which evaluate the effectiveness of the proposed Item-based partitioning approach, are presented in Section 4. Section 5 draws conclusions and future developments of the proposed approach.

2. PREVIOUS WORK

2.1 Apriori Algorithm
The Association Rule Mining was first introduced in the AIS algorithm. It was again modified in [1]. Many
algorithms generate frequent sets based on Apriori algorithm. Apriori performs repeated passes over the database for support counting of one itemset, two itemset and three itemset and so on. The weakness of Apriori is that it requires multiple scans over the source data; it generates huge number of candidates, tedious workload of support counting for candidates. In practice, the number passes required is one greater than the size of the largest frequent set. It is especially a problem, if the source data cannot be contained in primary memory. These problems may be eliminated by the Partition Algorithm.

2.2 Partition Algorithm

The Partition algorithm is basically based on the Apriori algorithm. In Partition algorithm data logically partitioned into a number of equal-sized partitions, such that each partition can be accommodated in the main memory. The algorithm differs from the Apriori and other mining algorithms in terms of the number of passes it makes over the database. The algorithm makes just two passes over the input database to generate the rules. In the first pass it generates set of locally frequent itemsets. In the second pass it generates globally frequent itemsets. The drawback of partition algorithm is that it may have many candidates during second scan. Therefore the drawback of both these methods is that the number of candidates whose support is to be counted may become very large, especially when the data is such that the frequent sets may contain many items.

Partition algorithm data logically partitioned into a number of equal-sized partitions, such that each partition can be accommodated in the main memory. The algorithm differs from the Apriori and other mining algorithms in terms of the number of passes it makes over the database.

The algorithm makes just two passes over the input database to generate the rules. In the first pass it generates set of locally frequent itemsets. In the second pass it generates globally frequent itemsets. The drawback of partition algorithm is that it may have many candidates during second scan.

The drawback of Apriori and Partition algorithms is that the number of candidates whose support is to be counted may become very large, especially when the data is such that the frequent sets may contain many items. It is also require all candidates to be retained in primary memory during final database pass.

2.3 Record-based Partitioning

This method begins by performing a single pass of the database to perform a base summation of the support totals. These base counts are stored in a tree structure that we call the B-tree, which enumerates itemsets counted in lexicographic order. The B-tree contains all the sets of items present as distinct records in the database, and some additional sets that are leading subsets of these.

Consider a database with 20 records and itemset is \{a, b, c, d, e\}. Records are \{abcde, abce, abd, abde, a, acde, ace, ade, b, bcde, bce, bd, bde, be, cd, cde, ce, d, de, e\}. Figure 1 shows the B-tree that would be constructed. The counts stored and its succeeding supersets in the tree. We then apply to this structure an algorithm, Apriori-TFP, which completes the summation of the final support counts, storing the results in a second set-enumeration tree, it is ordered in opposite way to the B-tree. The F-tree (Final support count tree) finally contains all frequent sets with their complete support-counts.

![Fig. 1: Example of a B-tree](image)
memory in each pass. The second problem is that the entire F-tree, which finally contains all the frequent sets, must be contained in primary memory while counting proceeds. This tree may itself become very large, especially when long frequent patterns are encountered. When the tree becomes too large to be contained in primary memory, a large set of candidates leads to slower counting.

An alternative strategy for partitioning the data is to divide the itemset into subsets; it is called as item based partitioning. The problem with this is, the sets for which support is to be counted contain items from several partitions. The B-tree structure offers another form of item partitioning, into sub trees that represent equivalence classes of the items represented. In this strategy, it is not possible to compute the support for a set by considering only the sub tree in which it is located.

The advantage for partitioning Base tree is that PB-trees become more compact and more equal in size.

The process of total support using PB-trees is:
1. Get an order of the frequency of items.
2. Choose an appropriate partitioning of the items into \( n \) sequences 1, 2, 3, … etc by using above order.
3. Divide the source data into \( m \) parts based on records.
4. For each part, construct \( n \) PB-trees in primary memory, storing finally to disk. It involves just one pass of the source data.
5. For partition 1, read the PB1 trees for all segments into memory, and apply the Apriori-TFP algorithm to build a F-tree that finds the final frequent sets in the partition. This stage requires the PB1 trees for each segment of data to be read once only. The F-tree remains in memory throughout, finally being stored to disk.
6. Repeat step 5 for partitions 2, 3,… \( n \).

This method offers two advantages over partition algorithm. First advantage, we have now effectively reduced the number of disk passes to 2: one to construct the PB-trees, and a second pass to complete the counting. The second advantage is that at each stage, dealing with smaller tree structures, leading to faster traversal and counting.

There are many common properties in common with the FP-tree structure. There are two main differences. First, the nodes of FP-tree correspond to individual items, whereas in the \( B \)-tree a sequence of items which is partially closed will be stored as a single tree node. The second difference is that, in order to implement the FP-growth algorithm, the FP-tree must store pointers at each node to link all nodes representing the same item and child nodes. \( B \)-tree is a set of nodes which can be processed in any order. Once tree has been constructed, no pointers are required to store data in a tabular form. The absence of pointers allows us easily to use record based partitioning in order to build a succession of \( B \)-trees, \( B \)-tree again partitioned into FB-tree.

4. EXPERIMENTAL RESULTS
We now discuss the implementation of Item-based partitioning approach to soybean data set. We implemented the Item-based Partitioning algorithm by using Python. The experiments were conducted on 3.0 GHz Pentium IV (Core to Dual) workstations running Windows XP professional operating system configured with 2GB main memory and 300 GB SATA 7200 rpm disk. Our developed program takes the input file as .arff type or .txt (tab separated) type. Soybean data file is .arff type and it contains 683 records. Our program first generates all frequent itemsets and then generated association rules.

4.1 Data Set and Description
The dataset includes the description of soybean. There are 19 classes, only the first 15 of which have been used in prior work. The folklore seems to be that the last four classes are unjustified by the data since they have so few examples. There are 36 categorical attributes, some nominal and some ordered. The value “dna” means does not apply. The values for attributes are encoded numerically, with the first value encoded as “0”, the second as “1”, and so forth. An unknown value is encoded as “?”.

It consists of 683 records and its source is agridataset.jar [8]. Here the task is to diagnosis soybean disease. Attributes are Date, Plant-stand, Precip, Temp, Hail, Crop-hist, Area-damage, Severity, Seed-tmt, Germination, Plant-growth, Leafruit-pods, Fruit-spots, Seed, Mold-growth, Seed-discolor, Seed-size, Shriveling, roots, class.

4.2 Assumptions
The implementation of the Item-based Partitioning Algorithm was done to help in discovering and presenting the features of the algorithm, to accomplish this task, the following steps were made.
1. Initial soybean database was gathered
2. Support and Confidence for the rules were modified many times and the influence of their change on
the rules was recorded on all experiments.

3. All experiments were implemented on the same hardware (same computer with the same processor and memory).

4. All experiments were implemented on the same operating system.

5. No additional processes were running in the background.

6. No scheduled programs were running.

7. No Screen saver was chosen.

Figure 2 shows the overall time to generate association rules with support of 10% for records of 100, 200, 300, 400 and 500. It shows IBP takes less time than RBP as we increase the size of dataset.

Figure 3 shows the execution times for the IBP method and RBP method with a support threshold of 0.01, an Item-based partitioning of 15 items per partition, and record based partitioning of 3,000 records per segment. Here the time taken by IBP is less when compared with RBP that is record-base partitioning has lower performance than Item-based partitioning.

Figure 4 shows the execution time taken by IBP and RBP for different values of support factor. It is observed that as the support threshold decreasing from 90% to 10% IBP performs very well than RBP. For low support value RBP takes much time than IBP.

Figure 5 presents the execution time of RBP and IBP for different values of the confidence factor with minimum support of 30%. At 90% confidence, the time taken by IBP and RBP to generate association rules is 0.30 and 0.50 respectively. At the confidence of 10%, the time taken by IBP and RBP is 0.4 and 0.58 respectively. It is observed that RBP has lower performance than IBP.

Figure 6 illustrates that the performance of IBP for different values of support factor on different sized soybean dataset. It is observed that as the size of the database increases the performance of the IBP decreases dramatically. The performance depends only on the size of database but not support factor.
Figure 7 illustrates that the performance of RBP for different values of support factor on different sized soybean dataset. It is observed that as the size of the database increases the performance of the RBP decreases dramatically. The performance depends only on the size of database but not support factor. From Figure 6 and 7, we observed that IBP has best performance than RBP.

REFERENCES


5. CONCLUSIONS

Knowledge Discovery and Data Mining (KDD) is an interdisciplinary area focusing upon methodologies for extracting useful knowledge from data. Association rule mining is an important and useful technique to discover interesting patterns which is hidden in the database. In this paper we have examined ways of partitioning data for Association Rule Mining from the soybean data set. Compared the performance of Association rule generation using Item-based partitioning and Record-based partitioning approaches. In future, we also plan to extend this work to the multiple databases.