D-006

Accumulative Mining of Frequent Patterns

Tran MINH QUANG (*) Shigeru OYANAGI (*) Katsuhiro YAMAZAKI (*)

1. Introduction

Association rule mining is to analyze relationships between data in a large database. It has been applied in various fields including market analysis, customer habit study, detection, classification, clustering and so on. However, it consumes a great amount of time. There are 2 steps in mining association rules: Finding all frequent patterns and then generating association rules from above frequent patterns. The second step is the easiest one of two and the overall performance is determined by the first step [1]. Various researches have been done under the Apriori-like approach [2] to improve performance of the frequent pattern mining task but the results were not as much as expected due to many scans on dataset [3]. Recently, tree based approach has been proposed to compact data in tree-like data structures minimizing search space hence improve the mining performance. The most prominent one is the frequent pattern tree (FP-tree) which scans database only twice and then applies FP-growth algorithm to mine frequent patterns from the tree [4]. This method substantially solves the stalemate of the Apriori-like approaches but is not flexible in mining with different support thresholds. The tree has to be rebuilt from the scratch whenever the support threshold is changed. This research aims to propose a flexible way of mining frequent patterns with different support thresholds in manner of accumulative mining by extending the idea of the FP-tree method.

The rest of the paper is organized as follows. Section 2 is the ideas of extending FP-tree method for "building once – mining anytime" and accumulative mining. The experimental evaluation is presented in section 3. Section 4 is discussions and future work.

2. Accumulative Mining Frequent Patterns

2.1. Building Once –Mining Anytime

In FP-tree method, data is compacted in a prefix frequent pattern tree (FP-tree). The FP-growth algorithm is then applied to mine frequent patterns from the tree. The process of this method is described as follows: (1) According to a given support threshold ϴ, dataset is scanned twice to build the FP-tree. The first scan is to recognize frequent items that will appear in the tree and the second scan is to build the tree [4]. (2) FP-growth algorithm is applied to mine frequent patterns. For each frequent item above, FP-growth algorithm traverses the tree to recognize its conditional pattern bases, considers them as a sub-dataset, and builds the conditional FP-tree for this sub-dataset. The FP-growth algorithm is then continually applied on this conditional FP-tree recursively until final frequent patterns are found.

As mentioned above an FP-tree is built based on a support threshold, hence whenever the support threshold is changed the tree has to be rebuilt from the scratch. Moreover, to recognize conditional pattern bases for each frequent item, FP-growth algorithm has to traverse the tree many times which costs a great amount of time. For solving this problem we propose to build a "global" FP-tree (a tree with the smallest support threshold among desire support thresholds, in a general case this support threshold is 1 - in support count number). Then information (in fact are conditional pattern bases of all frequent items) of the tree is saved into a file, called "conditional patterns" file. The conditional pattern bases are sorted by the descending order of frequent items' supports. Therefore the file is conveniently be read from the beginning until a place where the support of item becomes unsatisfied a given support threshold. This file can be read any time to mine frequent patterns with any different support threshold. This is the idea of "building once – mining anytime". Experimental results in the next section reveal that this approach significantly improves the performance compared to the original FP-tree method.

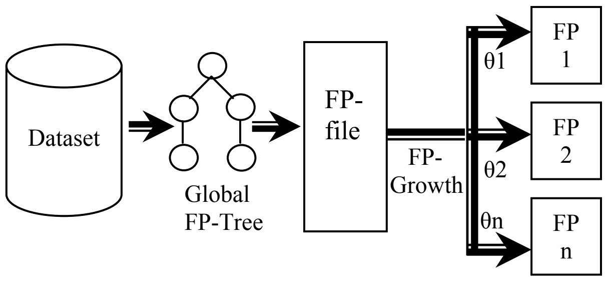

Figure1: Building once – Mining anytime

2.2. Reusing Frequent Patterns

Let ϴ1, ϴ2 (ϴ1> ϴ2) be two support thresholds and S1, S2 be their two corresponding frequent pattern sets, obviously S2 ⊇ S1. Call S2 = S1 + P in which P is a set of frequent patterns that have support ϴ, where ϴ2<=ϴ<ϴ1. It is very easy to extract S1 from S2, if S2 has already been mined, because the frequent patterns in S2 are sorted by the descending of their supports. The problem reveals in the converse case. Assume that S1 has already been mined before, and S2 is requested to be mined. The traditional approach is to mine S2 from the beginning. While S2 is being mined by this way, all frequent patterns that satisfy ϴ1 (included in S1) have to be remained again. The computation time can be reduced

* Graduate School of Science and Engineering,
Ritsumeikan University, Biwako-Kusatsu Campus Noji
Higashi 1 chome, 1-1 Kusatsu, 525-8577 Shiga-ken, JAPAN

if S1 is reused, and P is the only one has to be mined. After that, P and S1 are merged together to obtain S2.

With the advantages of the "conditional patterns" file mentioned above, P can easily be mined as described bellow:

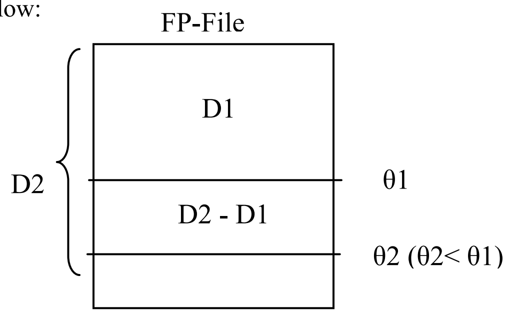

Figure2: Accumulative mining

Figure 2 is a conditional patterns file in which appropriate portions of data can be read according to given support thresholds. Assume that `setofPattern mine(int θ, dataset D)` is a routine to mine frequent patterns in dataset D with support threshold θ. In traditional way, we can mine S1, S2 by calling this routine, given a corresponding portion of dataset with an appropriate support threshold: `S1=mine(θ1, D1); S2=mine(θ2, D2)`. As mentioned above, S2=S1+P and S1 has been already mined. To mine S2 we just need to mine P only. Analyzing more detail, we can see that P=P12+P22, where P12 and P22 are frequent pattern sets which satisfy *θ2* only (not satisfy *θ1*) in D1 and in D2-D1 respectively. P12 and P22 can be mined as following:

```
P12=mine(θ2<=θ<θ1, D1)
P22=mine(θ2, D2-D1)
```

Finally, merge them together with S1 to obtain S2:

S2=S1+P12+P22.

3. Experimental Evaluation

The experiment was taken in a synthetic dataset of 50000 transactions, 1000 items and the average of transaction length is 10. Figure 3, displays the computation time needed to mine frequent patterns with the support thresholds, *θ*, changed from 30 to 3 according to three methods above. The last column represents for the "accumulative mining" method. For this method, at support threshold equal to 30 (*θ2=30*), it was assumed that the frequent pattern set with support threshold equal to 50 (*θ1=50*) has already been mined and was reused. In the rest groups of columns, the computation time of this method is presented with the assumption that the frequent pattern set for the previously adjacent group was reused. For example, at *θ2*=20, it was assumed that the frequent pattern set with *θ1*=30 was reused, and so on.

The results show that "building once – mining anytime" approach is significantly better than the original FP-tree method which saves about 25% (up to 50%) of the computation time. The method of reusing frequent patterns for accumulative mining is slightly better than "building once – mining anytime" method and it seems to be much better in the case of two support thresholds are almost the same and are both small, for instance *θ2=3 and θ1=4*.

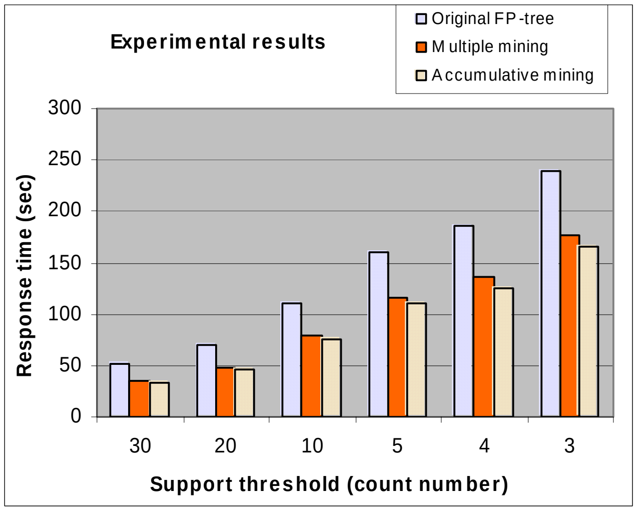

Figure3: Experiment result

4. Discussions and Future Work

In this research we have proposed an extension to the FP-tree method for frequent patterns mining. The experimental results show a substantial improvement in performance. The improvement is due to three main reasons. Firstly, the tree is only built once at the first time and can be reused to mine any time later. Second, characteristics of the "conditional patterns" file permit FP-growth algorithm to recognize conditional pattern bases more directly without traversing the tree which save a great amount of time. Finally, the frequent pattern set with the greater support threshold is reused for mining a frequent pattern set with a smaller support threshold in an accumulative manner. One of the most disadvantages of this method is the limitation of the memory which may not fit the "global" FP-tree. This problem can be solved by database partitioning. The idea of database partitioning also leads us a new possible direction for future work in which parallel computing should be applied into the association rules mining accompanied with this extension.

References:

[1] Han, J., Kamber,M. *Data Mining Concepts and Techniques*, San Francisco: Morgan Kaufmann, 2001

[2] Agrawal, R, and Srikant, R. *Fast algorithm for mining association rules*. VLDB, 1994.

[3] Savasere, A., Omiecinski, E., and Navathe, S. *An efficient algorithm for Mining Association rules in large databases*. Proceedings of VLDB Conference. 1995.

[4] Han, J., Pei, J., and Yin, Y. *Mining frequent patterns without candidate generation.* SIGMOD, 2000, pp. 1-12.