The Seventeenth International Symposium on Artificial Life and Robotics 2012 (AROB 17th '12),
B-Con Plaza, Beppu, Oita, Japan, January 19-21, 2012

Batch fast update algorithm for incremental association rule discovery

Araya Ariya[1] , Worapoj Kreesuradej[2]

King Mongkut's Institute of Technology Ladkrabang, Thailand

[1]araya_aa@hotmail.com , [2]worapoj@it.kmitl.ac.th

Abstract: When new transactions are inserted into an original database, the existing rules may be change. An incremental association rule mining is an approach to deal with such problem. This paper proposes an algorithm for mining incremental association rules, called batch fast update (BFUP). The proposed algorithm improves the performance of FUP algorithm by reducing a number of scanning times of an original database. The experimental results show that an execution time of BFUP is much faster than that of FUP.

Keywords: Incremental association rules mining, Association rule mining, Data mining

1. INTRODUCTION

An association rule mining, one of the important tasks in data mining, is a well-known research topic that many researchers propose a large number of algorithms for solving association rule discovering problems. This problem was first presented by Agrawal et al [1] which analyzes the behavior of customer purchasing to help business make a decision. The results show co-occurrence buying items called frequent patterns, which can generate interesting rules. From that research, the problem statement of an association rule mining is defined as follows.

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of literal items. DB is a database which contains transactions. Each transaction T is a set of items where $T \subseteq I$. Given X is an item and $X \subseteq I$. Each transaction contain X if and only if $X \subseteq T$. Let X and Y are an item where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y \neq 0$. Each set of items , itemsets, is called a frequent itemset if and only if its support is greater than or equal to support threshold s%. It calculates from a number of transactions in DB that contain $X \cup Y$. An association rule can be shown in $X \Rightarrow Y$ form. Each frequent itemset can be made the association rule if and only if it is satisfied by confidence threshold c% which calculates from a number of transactions in DB that contain X and also contain Y. Both s% and c% are specified by user.

After an association rule mining was revealed, it motivated many researchers to extend this research area in a lot of issues. An incremental association rule mining is the one of an association rule mining issue which maintains association rules when new transactions are appended to an original database.

One research issue of an incremental association rule mining is reducing running time of the algorithms by minimizing the number of times to scan an original database. This paper also works on this issue. An algorithm for mining incremental association rules, called batch fast update (BFUP), is proposed. This algorithm has only one original database scanning.

The paper is organized as follows. The literatures of an association rule mining and an incremental association rule mining are reviewed in section 2. The problem statement of an incremental mining on association rule in dynamic database and FUP algorithm are detailed in section 3. The proposed algorithm and its experiment are presented in section 4 and 5 respectively. The paper conclusion and future work are briefed in section 6.

2. RELATED WORK

Mining association rules was first proposed by Agrawal et al [1] which finds a correlation between itemsets in a transaction database. The algorithm has 2 steps: finding frequent itemsets (sometimes they are called large itemsets) and generating rules. Subsequent years, Apriori [2], the most popular algorithm, was proposed to discover frequent itemsets.

When a database, called a dynamic database, is inserted new transactions, frequent itemsets can be changed after inserting new transactions into the dynamic database. Therefore, an association rule discovery algorithm for a dynamic database has to maintain frequent itemsets when new transactions are inserted into the dynamic database.

One approach to find the new frequent itemsets is to rerun Apriori algorithm for the whole transactions of the dynamic database. This approach is not efficient because all the computation done initially at finding out the old large itemsets are wasted and all large itemsets have to be computed again from scratch [3].

Cheung et al [3] proposed fast update algorithm, FUP, to solve a rules maintenance problem by using the previous knowledge to find frequent itemsets in updated database. The concept of FUP is re-using frequent itemsets of previous mining to update with frequent itemsets of an incremental database. Although FUP can decrease a number of candidate itemsets for scanning original database, it still needs to scan an original database k times when new frequent itemsets are found. This can degrade the performance of FUP algorithm.

In our observation, the advantage of FUP are re-using frequent itemsets of a previous mining to prune itemsets which cannot be a frequent itemset in updated database and reducing candidate itemsets to scan in an original database.

However, the disadvantage of FUP is the algorithm needs to scan an original database equal to a size of k frequent itemsets, e.g., if maximum size of k frequent itemsets is k=5, FUP needs to scan an original database for 5 times.

From this problem, this paper proposes an incremental association rule mining algorithm to improve the performance of FUP algorithm by reducing a number of scanning times of an original database.

3. INCREMENTAL ASSOCIATION RULES MINING

3.1 Problem statement

In a dynamic database, new transactions are appended to a database; accordingly, the previous valid rules may be invalid. The problem statement for an incremental association rule is defined as follows.

Let DB is an original database. An increment database db is the new transactions which are inserted into DB. Updated database UD is the combining between original database and increment database, i.e, UD = DB $\cup$ db. A number of transactions of an original database, an increment database and an updated database are |DB|, |db| and |UD| = |DB|+|db| respectively.

Before updating activity, L is the frequent itemsets in DB if and only if X.support $\geq$ s×|DB|. After updating activity, L' is the frequent itemsets in updated database if and only if X.support $\geq$ s×|UD|.

According to Tsai et al [4], when news transactions are insert into an original database, an itemset, i.e. X, can be categorized into 4 cases:

Case 1: X is a frequent itemset in both DB and UD

Case 2: X is a frequent itemset in DB and an infrequent itemset in UD

Case 3: X is an infrequent itemset in DB and a frequent itemset in UD

Case 4: X is an infrequent itemset in both DB and UD

Form these cases of itemsets are mentioned above, it is easy to discovered updated frequent itemsets for the itemset of case 1 and 2 because their count in DB and UD are known, therefore, an updating activity is a trivial task. The itemset in case 4 is unimportant because it cannot change an association rule. The most serious case is the 3rd because it needs to rescan an original database for updating its count. Thus, discovering itemsets in case 3 is the most important problem in an incremental association rule mining. In section 3.2, the method to solve that problem is reviewed and section 4, batch fast update algorithm is presented how to improve a performance of FUP.

3.2 FUP algorithm

Apriori [2] is successful for finding frequent itemsets in a database. However, it is unsuitable for mining in dynamic database. Cheung et al [3] have been proposed an incremental algorithm which has a good performance for mining association rules in dynamic database. The concept of FUP is reviewed briefly in this section.

The operation of FUP has 2 phases: 1-iteration and k-iteration where k $\geq$ 2. In the first phase, an increment database is scanned for finding candidate 1-itemsets C_1 and its count. After that loser and winner itemsets are found. Finding loser and winner itemsets, C_1 are divided into 2 types: a member and not a member of previous frequent itemsets in an original database. The first type is updated its count and pruned loser itemsets if its updated count is less than s*|UD|. The second type is scanned to an original database if and only if it is a frequent itemset (winner) in an increment database, i.e., its count is greater than or equal to s*|db|. Both types which satisfy by them threshold can be frequent 1-itemsets in updated database L_1'(winner).

The second phase has 3 steps: filtering out loser itemsets, generating candidate k-itemsets $C_{k\geq 2}$ and finding new frequent itemsets. Firstly, FUP filters out losers from L_k (L_k in DB). Given $Y \in L_k$ and $X \in L_{k-1} - L_{k-1}'$, Y is a loser iff $X \in Y$. For the remaining L_k, they are scanned to an increment database and updated their count. Then they are checked for finding a winner or loser itemset similar to the first phase.

Secondly, C_k is generated by using Apriori-gen. Any C_k is pruned if and only if $Y \in C_k$ where Y is the loser from L_k. This step is a key to reduce a number of C_k before scanning an original database. Finally, new frequent itemsets L_k' are found with the same method as the first phase.

4. BATCH FAST UPDATE ALGORITHM

In this section, an algorithm for mining incremental association rules, batch fast update (BFUP) is presented. The proposed algorithm is assumed that two thresholds, minimum support s% and minimum confidence c%, are static. This algorithm needs only one original database pass and infrequent itemsets are not required. The notation used in this section is defined in Table 1.

Table 1. The notation for Batch fast update algorithm

notation	meaning
DB	original database
db	increment database
UD	updated database
s	minimum support
L_k^{DB}	frequent k-itemset in DB
L_k^{UD}	frequent k-itemset in UD
C_k	candidate k-itemset
|DB|	a number of transactions in DB
|db|	a number of transactions in db
|UD|	a number of transactions in UD
X.count	a support of an itemset
Temp_scanDB	itemsets which are scanned in DB

The algorithm has 2 phases: an increment updating phase and a re-scanning original database phase. The first phase is shown in figure 1. At each iteration, an increment

database is scanned to find candidate itemsets, i.e., C_k, and their support counts. Basically, candidate itemsets are divided into 2 types: a member and not a member of previous frequent itemsets of an original database. Candidate itemsets of the first type becomes updated frequent itemsets, i.e., L_k^{UD}, if and only if their updated support count is greater than or equal to s*|UD|. Candidate itemsets of the second type are kept in temp_scanDB for rescanning in an original database if and only if their count plus |DB|-1 is greater than s*|UD|. On the other hand, Candidate itemsets of the second type are pruned if and only if their count plus |DB|-1 is less than s*|UD|.

For generating candidate itemset C_k, Apriori-gen is applied. Apriori generates C_k with L_{k-1}*L_{k-1}, whereas BFUP generates C_k with L_k^{UD} * $temp\_scanDB_k$.

The second phase of BFUP algorithm is shown in figure 2. After an increment updating phase is ended, all itemsets in temp_scanDB are re-scanned in an original database and updated their support count. Then, all itemsets in temp_scanDB are checked to find updated frequent itemsets. Let X ∈ temp_scanDB, X can be an updated frequent itemset, i.e., L_k^{UD}, if and only if X.count ≥ s*|UD|.

Algorithm 1 An increment updating phase ()

```
Input : db,s,L_k^DB,C_1^DB,|DB|
Output: L_k^UD
1   |UD|=|DB|+|db|
2   k = 1
3   scan db for all X
4   for all X∈L_1^DB or X∈C_1^DB
5     update X.count
6     if X.count ≥ s*|UD|
7         X → L_k^UD
8   for all X ∉ L_1^DB or X∉C_1^DB
9     if X.count+|DB|-1 ≥ s*|UD|
10        X → temp_scanDB
11  k=2
12  while (L_k-1^UD ∪ temp_scanDB_k) > 1
13    C_k = L_k-1^UD * temp_scanDB_k
14    // using Apriori_gen()
15    scan db for all C_k
16    for all X∈ C_k do
17        for all X ∈ L_k^DB
18            update X.count
19            if X.count ≥  s*|UD|
20                X → L_k^UD
21        for all X ∉ L_k^DB
22            if X.count+|DB|-1 ≥ s*|UD|
23                X → temp_scanDB
24    k++
25  end loop
26  rescan_original()
27  L_k^UD = L_k^UD ∪ tempL
28  return L_k^UD
```

Fig. 1. An increment updating phase

Algorithm 2 a re-scanning original database phase ()

```
Input DB, temp_scanDB,s,|UD|
Output tempL
1   if temp_scanDB ≠ ∅
2       scan DB for all X ∈ temp_scanDB
3       update X.count
4       if X.count ≥ s*|UD|
5           X → tempL
6   endif
7   return tempL
```

Fig. 2. A re-scanning original database phase

5. EXPERIMENT

The proposed algorithm in this paper aims to improve the performance of FUP. To evaluate the performance of batch fast update (BFUP) algorithm, this algorithm is implemented and tested on a PC with a 2.93 GHz Intel Core i7 and 3 GB main memory. The experiment is tested with 2 synthetic datasets which are generated by using technique in Agrawal [1]. The first dataset is T10I4D100K which has 100,000 transactions. The second dataset is T10I4D50K which has 50,000 transactions. Both datasets are appended by an increment database that has 1,000 transactions.

For the first original database, i.e., T10I4D100K, the experiment is conducted with 0.1%, 0.3% and 0.5% minimum support thresholds. The average of an execution time is shown in table 2 and figure 3 respectively. For comparison, the results are compared with FUP.

Table 2. Average of Execution time for I10T4D100K

min sup	algorithm	execution time (sec.)	a number of frequent itemset	maximum frequent itemset (size of k)
0.1%	FUP	175890.2901	17,127	L_{10}
	BFUP	43580.3014		
0.3%	FUP	19645.1002	1,991	L_7
	BFUP	11678.5801		
0.5%	FUP	9771.1612	862	L_2
	BFUP	9105.8170		

For the second original database, i.e., T10I4D50K, the experiment is conducted with 0.2%, 0.3% and 0.4% minimum support thresholds. The average of an execution time is shown in table 3 and figure 4 respectively.

From the results of the both datasets, they are shown that an execution time of BFUP is much faster than that of FUP. Furthermore, we observe that the more size k is increasing, the execution time between FUP and BFUP is more different.

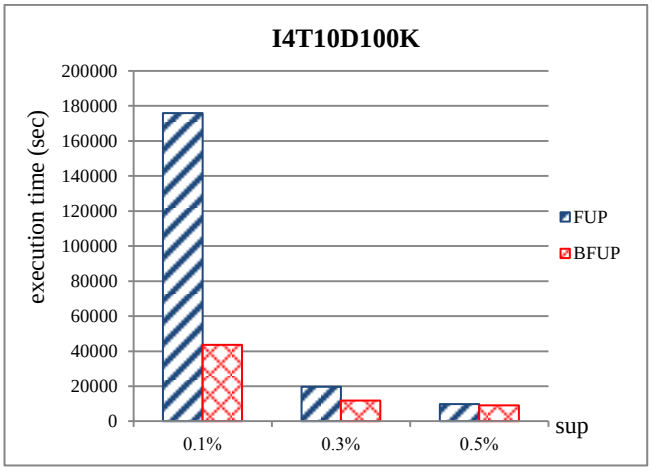

Fig. 3. Execution time comparison for I10T4D100K

Table 3. Average of Execution time for I10T4D50K

min sup	algorithm	execution time (sec.)	a number of frequent itemset	maximum frequent itemset (size of k)
0.2%	FUP	36012.0041	5,307	L_{10}
	BFUP	15402.0908		
0.3%	FUP	17695.8894	1,995	L_7
	BFUP	12906.0013		
0.4%	FUP	12363.9417	1,051	L_3
	BFUP	10866.7162		

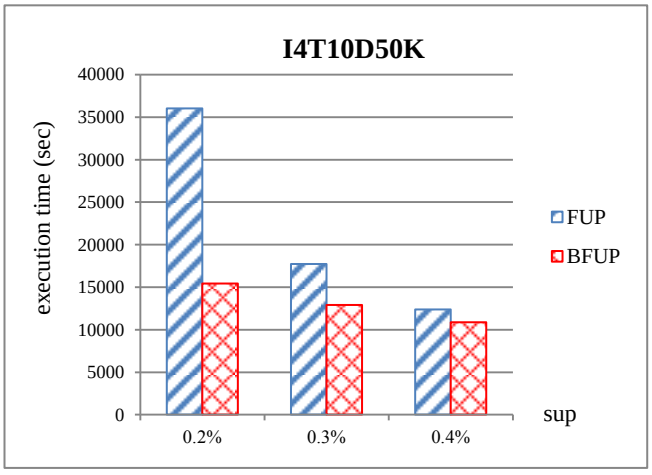

Fig. 4. Execution time comparison for I10T4D50K

6. CONCLUSION

An incremental association rules mining algorithm called batch fast update (BFUP), is proposed. The concept of this algorithm is based from Apriori and FUP algorithm. Although batch fast update algorithm has an execution time better than that of FUP, a large number of temp_scanDB are kept. In the future research, the algorithm for reducing temp_scanDB will be proposed.

7. REFERENCES

[1] Agrawal R, Imielinski T, Swami A (1993), A mining association rules between sets of items in large database, In Proceeding of the ACM SIGMOD Int'l Conf. on Management of Data (ACM SIGMOD'93), Washington, USA, May 1993, pp.207-216.

[2] Agrawal R, Srikant R (1994), Fast algorithm for mining association rules, In Proc. 20th Int. Conf. Very Large DataBases (VLDB'94), Santiago, Chile, September 12-15, 1994, pp.487-499.

[3] Cheung D.W., Han J, Ng V.T., Wong C.Y., Maintenance of Discovered Association rules in Large Databases: An incremental updating technique, In 12th IEEE International Conference on Data Engineering, pp 106-114, 1996.

[4] Tsai Paulry S.M., Lee Chih-Chong, Chen Abree L.P. (2005), An Efficient Approach for Incremental Association Rule Mining, Proceedings of the third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining, Lecture Notes In Computer Science, Vol. 1574 archieve, 1999.