Tenth Annual

Workshop on Information Technologies & Systems

WITS 2000

9-10 December 2000
Brisbane, Australia

A Monitoring Algorithm for Incremental Association Rule Mining

Xiao Fang, Olivia Sheng
Department of Management Information Systems
University of Arizona, AZ 85719
xfang@bpa.arizona.edu, sheng@bpa.arizona.edu

Abstract

As knowledge generated by traditional data mining methods reflects only the current state of the database, a database needs to be re-mined every time an update occurs. For databases with frequent updates, such operations not only could inflict unbearable costs but also often result in repetitive knowledge identical with previous mining because data in real world applications overlap considerably [10]. To tackle such a problem, in this paper, we propose a monitoring algorithm for association rule mining that can determine whether or not it is necessary to re-mine an updated database. Pattern difference is defined to measure changes in patterns between original data and incremental data (i.e., new data). Based on the significance of pattern difference, our algorithm will determine the necessity of re-mining. This paper also presents experiments that test the reliability and efficiency of the monitoring algorithm by using synthetic data sets generated by the extensively used IBM test data generator[1]. The comparison of the proposed algorithm with the traditional Apriori algorithm has shown that the former is not only reliable but also much faster.

1. Introduction

Data mining [1] gives organizations the tools to sift through these vast data stores to find the trends, patterns, and correlations that can guide strategic decision making. Many studies have shown that mining knowledge from databases needs to deal with the following fundamental problems [2,3]:

1. The size of databases is significantly large.
2. The number of rules resulting from mining activity is also large.
3. The rules derived from a database reflect only the current state of the database.

Substantial research has been done to improve the efficiency of the data mining process and to prune rules returned from data mining, but little has been done to solve the third problem.

The problem of mining association rules over market basket data has received a great deal of attention since it was introduced by Agrawal *et al* [4]. Although it has been well researched from the aspect of mining efficiency [5,6,7,8,9], association rules discovered by algorithms in [4,5,6,7,8,9] reflect only current patterns in the database. Incremental data (i.e., new data) could potentially invalidate existing rules or introduce new rules. To keep association rules up-to-date, an obvious solution is to re-compute rules from the whole updated database. Some algorithm regarding re-computing efficiency has been proposed in [3], but this simple solution requires association rules to be re-computed over the whole updated database every time an update occurs. For example, for a data warehouse updated daily, association rules would need to be re-computed every day. This solution could bring an unbearable cost of machinery and personnel. Further, for incremental data quite similar to the original data in patterns, association rules would not be changed significantly after updating. In this situation, re-computation of association rules is not necessary. Such a situation happens often in the real world [10]. Therefore, a monitoring algorithm that can judge the significance of pattern differences between original data and incremental data could reduce costs considerably. In this paper, we propose a monitoring algorithm that can suggest whether or not it is appropriate to re-mine an updated database and present experimental results from using the monitoring algorithm.

[1] http://www.almaden.ibm.com/cs/quest/demos.html

2. A Monitoring Algorithm for Incremental Association Rule Mining

2. 1 Preliminaries of Association Rule Mining

Let $L = \{i_1, i_2, \cdots, i_k\}$ be a set of items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq L$. The support of an itemset (set of items) in D is the fraction of all transactions containing the itemset. An itemset is called large if its support is greater or equal to a user-specified support threshold. Otherwise, the itemset is called small. An association rule is an expression $X \Rightarrow Y$ where $X \cap Y = f$. The support of this rule is the support of $X \cup Y$ and the confidence of this rule is the support of $X \cup Y$ divided by the support of X. For an association rule to hold, it must have a support greater or equal to a user-specified support threshold and a confidence greater or equal to a user-specified confidence threshold. The problem of discovering association rules can be decomposed into two steps [9]:

1. Find all large itemsets.
2. For each larger itemset L, find all subsets S, such that the confidence of $S \Rightarrow L \setminus S$ is greater or equal to the confidence threshold.

2.2 Metrics for Pattern and Pattern Difference

In the rest of the paper, DB denotes original data, db denotes incremental data and DB+ denotes updated data (i.e., original data + incremental data). L_{DB}, L_{db}, and L_{DB+} denote the collection of large itemsets in DB, db and DB+ respectively. t_{DB}, t_{db}, and t_{DB+} denote the number of transactions in DB, db and DB+ respectively. $\sup p_{DB}(k), \sup p_{db}(k)$, and $\sup p_{DB+}(k)$ denote the support of itemset k in DB, db and DB+ respectively. And $t_{DB}(k), t_{db}(k)$, and $t_{DB+}(k)$ denote the number of transactions in DB, db and DB+ respectively containing itemset k.

The purpose of association rule mining is to discover knowledge from data in the form of association rules. Since association rules are computed from and determined by large itemsets [9], large itemsets and their supports can be used to measure patterns in data. Differences between L_{DB} and L_{db} can be used to measure pattern differences between DB and db. There could be three types of differences between L_{DB} and L_{db} [3,10].

1. Some large itemsets in DB become small in db, termed as disappearing.
2. Some large itemsets in DB are still large in db, but with different supports, termed as shifting.
3. Some small itemsets in DB become large in db, termed as emerging.

To compute pattern difference between DB and db, for every large itemset $k \in L_{DB} \cup L_{db}$, supports of k in DB and db need to be compared. Assuming no pattern difference between DB and db, the expected support of an itemset k in db is the support of k in DB. Hence, difference between the expected support of k in db and the observed support of k in db can be used to measure pattern difference between DB and db.

Definition: Pattern difference between DB and db is defined by the following equation:

$$pd = \sum_{\forall k \in I} \frac{(\sup p_{db}(k) \times t_{db} - \sup p_{DB}(k) \times t_{db})^2}{\sup p_{DB}(k) \times t_{db}}$$

pd : *p*attern difference between DB and db

$I = L_{DB} \cup L_{db}$

2.3 The Monitoring Algorithm

Lemma 1: Let s be any itemset and $s \notin L_{DB}$. Then $s \in L_{DB+}$ only if $s \in L_{db}$

Proof: Assume that there is an itemset s such that $s \notin L_{db}, s \notin L_{DB}$ and $s \in L_{DB+}$

Since $s \notin L_{db}$ then, $t_{db}(s) < \min \sup \; port \times t_{db}$ [2] ---------- (1)

Since $s \notin L_{DB}$ then, $t_{DB}(s) < \min \sup \; port \times t_{DB}$ ---------- (2)

(1) + (2) gives us $t_{db}(s) + t_{DB}(s) < \min \sup \; port \times (t_{db} + t_{DB})$

So, $t_{DB+}(s) < \min \sup \; port \times t_{DB+}$

Thus, $s \notin L_{DB+}$ which is a contradiction.

Lemma 2: If $L_{DB} = L_{db}$ and for each itemset $k \in L_{DB}, \sup \; p_{DB}(k) = \sup \; p_{db}(k)$ then $L_{DB+} = L_{DB}$ and for each itemset $k \in L_{DB+}, \sup \; p_{DB+}(k) = \sup \; p_{DB}(k)$

Proof: For each itemset $k \in L_{DB}$, since $L_{DB} = L_{db}$, therefore $k \in L_{db}$

$$t_{DB}(k) = \sup \; p_{DB}(k) \times t_{DB} \quad \text{---------- (1)}$$

$$t_{db}(k) = \sup \; p_{db}(k) \times t_{db} \quad \text{---------- (2)}$$

Since $\sup \; p_{DB}(k) = \sup \; p_{db}(k)$, (1)+ (2) gives us

$$t_{DB+}(k) = \sup \; p_{DB}(k) \times t_{DB+}$$

$\sup \; p_{DB}(k) >$ minsupport, thus $k \in L_{DB+}$ and $\sup \; p_{DB+}(k) = \sup \; p_{DB}(k)$

For each itemset $k \notin L_{DB}$, since $L_{DB} = L_{db}$, therefore $k \notin L_{db}$

According to Lemma 1, $k \notin L_{DB+}$

Therefore, $L_{DB+} = L_{DB}$ and for each itemset $k \in L_{DB+}, \sup \; p_{DB+}(k) = \sup \; p_{DB}(k)$.

According to Lemma 2, if there is no pattern difference between DB and db, there will be no pattern difference between DB and DB+. Since association rules are determined by large itemsets, there will be no change in association rules between DB and DB+. Therefore, if we can determine whether or not pattern difference between DB and db is significant, we could know whether or not it is necessary to re-mine DB+. If we assume the chi-squared distribution of pattern difference under the hypothesis **H0** that there is no pattern difference between DB and db, **H0** could be tested with a chi-squared goodness-of-fit test [11].

The Monitoring Algorithm in Figure 1 is used to compute chi-squared statistic and degree of freedom. Inputs of the algorithm are DB, db and L_{DB}. Outputs are chi-squared statistic and degree of freedom. First, we compute L_{db} from db. Then, we match itemsets in L_{DB} with those in L_{db} and copy the matched itemsets and their supports in DB and db into CHI. For itemsets in L_{DB} but not in L_{db} or vice-versa, we construct multiple-hashtree to go through db or DB once to obtain their supports.

Compute L_{db} from db;

for each itemset $k \in L_{DB}$ do

    if $k \in L_{db}$ // shifting

        copy k with $\sup \; p_{DB}(k)$ and $\sup \; p_{db}(k)$ into CHI;

[2] minsupport is the user-specified support threshold.

else // disappearing
copy k with sup $p_{DB}(k)$ into CHI;
copy k into I_{db};
end if
end for
for each itemset $k \in L_{db}$ do
if $k \notin L_{DB}$ // emerging
copy k with sup $p_{db}(k)$ into CHI;
copy k into I_{DB};
end if
end for
if $I_{db} \neq f$
Create multiple-hashtree based on I_{db};
Go through db once;
for each itemset $k \in I_{db}$
copy sup $p_{db}(k)$ into CHI;
end for
end if
if $I_{DB} \neq f$
Create multiple-hashtree based on I_{DB};
Go through DB once;
for each itemset $k \in I_{DB}$
copy sup $p_{DB}(k)$ into CHI;
end for
end if
Compute chi-squared statistic and degree of freedom from CHI;

Figure 1: The Monitoring Algorithm

Re-computing association rules with traditional algorithms [4,5,6,9] requires n_1 passes over DB+ where n_1 is the size of maximal large itemset in DB+. A more efficient one [3] that decreases the search space by leveraging the information in L_{DB} still requires n_1 passes over DB+. The monitoring algorithm proposed in this paper needs at most one pass over DB and n_2 to n_2 +1 passes over db where n_2 is the size of maximal large itemset in db. Table 1 lists time complexity comparison of these algorithms.

Algorithm	Number of Passes over Old Database	Number of Passes over Incremental Database
Apriori	n_1	n_1
Hashing	n_1	n_1
FUP	n_1	n_1
Monitoring	0 -- 1	n_2 -- n_2 +1

Table 1: Time Complexity Comparison of Different Algorithms for Incremental Association Rule Mining

Since I/O dominates the computation cost and DB is much larger than db, the monitoring algorithm that requires least passes over DB is more efficient than previous algorithms. Further, rules returned from previous algorithms need to be pruned and evaluated before knowledge can be generated. This procedure may take much more time than data mining itself. With the monitoring algorithm, if the algorithm suggests not mining updated database, knowledge mined from original data could be reused even if new data have come in.

3. Experimental Study

We conducted a set of experiments to assess the performance of the monitoring algorithm. The experiments were performed on a PC with 256M RAM and a Pentium 450 processor. The widely used IBM test data generator was employed to generate experimental data. The data set used as DB was T10.P200.I5.D100k (Average transaction length=10, Number of patterns=200, Average length of pattern =5, Number of transactions=100k). Three db -- T10.P202.I5.D1k, T10.P198.I5.D5k, and T10.P199.I5.D10k – were generated with similar pattern to DB. We set minimum support as 4% and minimum confidence as 95%. Running the monitoring algorithm, we got the results in Table 2.

Number of Transactions in db	Computed Chi-squared Statistic	Degree of Freedom	Critical Value (alpha = 0.05)	Accept H0 ? (yes/no)
1k	181.18	400	447.63	yes
5k	188.56	393	440.22	yes
10k	119.97	394	441.28	yes

Table 2: Results of Monitoring Algorithm for db with similar pattern to DB

Results in Table 1 predict that there would be no significant change in association rules after db was added. Hence, re-mining is not necessary. To verify the judgement, we mined association rules from both DB and DB+ using the traditional Apriori algorithm. Metrics are defined to measure rule changes.

Number of Same Rules = Number of rules hold in both DB and DB+
Rule Loss = (Number of Rules in DB - Number of Same Rules)/ Number of Rules in DB
Rule Gain = (Number of Rules in DB+ - Number of Same Rules)/ Number of Rules in DB
Total Rule Change = Rule Loss + Rule Gain

Number of Transactions in db	Number of Rules in DB	Number of Rules in DB+	Number of Same Rules	Rule Loss (%)	Rule Gain (%)	Total Rule Change (%)
1k	571	569	568	0.5	0.2	0.7
5k	571	567	562	1.6	0.9	2.5
10k	571	584	569	0.4	2.6	3

Table 3: Rule Change Between DB and DB+

Data in Table 3 show us no significant rule difference between DB and DB+, confirming that re-mining is not necessary. We also compare the efficiency of the monitoring algorithm and Apriori. On average, the monitoring algorithm was almost 3 times faster. Another experiment was done to verify whether the monitoring algorithm could detect significant rule changes. Two db of pattern different from the original data in transaction length and number of patterns were generated with sizes of 10k and 20k respectively. Running the monitoring algorithm, we got the results shown in Table 4.

Number of Transactions in db	Computed Chi-squared Statistic	Degree of Freedom	Critical Value (alpha = 0.05)	Accept H0 ? (yes/no)
10k	12,755,671.3	4089	4,238.88	no
20k	25,773,235.7	4104	4,254.15	no

Table 4: Results of Monitoring Algorithm for db with different pattern from DB

Results in Table 4 predict that there would be significant change in association rules after db was added. Hence, re-mining is necessary. To verify this judgment, we mined association rules from both DB and DB+ using Apriori. The data listed in Table 5 show that the change in rules is significant and indicate that re-mine is necessary.

Number of Transactions in db	Number of Rules in DB	Number of Rules in DB+	Number of Same Rules	Rule Loss (%)	Rule Gain (%)	Total Rule Change (%)
10k	571	546	461	19.264	14.886	34.150
20k	571	259	80	85.989	31.349	117.338

Table 5: Rule Change Between DB and DB+

4. Conclusion and Future Work

In this paper, we presented a monitoring algorithm to determine whether or not it is necessary to re-mine an updated database. Our method outperformed traditional methods because it prevented unnecessary re-mining of an updated database. Experimental study showed that our algorithm is not only reliable in detecting changes but also much faster than traditional re-computation algorithms. The algorithm will be very useful in real world applications because data in real world applications overlap considerably [10]. We are planning to apply the algorithm in other data mining areas, such as classification and clustering. We also intend to use the algorithm on some real world data, such as web log and call center records.

References

1. Ganti, V., J. Gehrke, 1999, "Mining Very Large Databases," *IEEE Computer*, 32 (8), pp. 38-45.
2. U. Fayyad, 1996, "Advances in Knowledge Discovery in Databases," MIT Press.
3. David W. Cheung et al., 1996, " Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proc. of 1996 Int'l Conf. on Data Engineering*, pp. 106-114.
4. R. Agrawal et al., 1993, "Mining Association Rules between Sets of Items in Large Database, " *ACM SIGMOD, pp. 207-216*
5. J. S. Park et al., 1995, " An Effective Hash-Based Algorithm for Mining Association Rules," *ACM SIGMOD, pp. 175-186*
6. A. Sacasere et al., 1995, " An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. 21st Int'l Conf. Very Large Data Bases, pp.432-444*
7. H. Tovonen, 1996, " Sampling Large Databases for Association Rules," *Proc. 22nd Int'l Conf. Very Large Data Bases, pp.134-145*
8. S. Brin et al., 1997, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," *ACM SIGMOD, pp. 255-264*
9. R. Agrawal and R. Srikant, 1994, " Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases, pp.487-499*
10. Venkatesh Ganti et al., 1999, "A Framework for Measuring Changes in Data Characteristics, " *Proc. of the Eighteenth ACM SIGMOD Symposium on Principles of Database Systems, pp. 126-137*
11. C. T. Le, 1998, " Applied Categorical Data Analysis," *Jon Wiley & Sons, Inc.*