Search Method of Time Sensitive Frequent Itemsets in Data Streams

Tae-Su Park[1], Ju-Hong Lee[2], Sang-Ho Park[1], Bumghi Choi[2] , and Deok-Hwan Kim[3]

[1,2] Dept. of Computer Science & Information Engineering, Inha University, Incheon, Korea
{taesu, parksangho}@datamining.inha.ac.kr,
{juhong, neural}@inha.ac.kr
[3] Dept. of Electronics Engineering, Inha University
deokhwan@inha.ac.kr

Abstract. Recently, due to technical improvements of storage devices and networks, the amount of data increases rapidly. In addition, it is required to find the knowledge embedded in a data stream as fast as possible. Data stream is influenced by time. Therefore, the itemsets which were not the frequent itemsets can become frequent itemsets. The volume of data stream is so large that it can hardly be stored in finite memory space. Current researches do not offer appropriate method to find frequent itemsets in which flow of time is reflected but provide only frequent items using total aggregation values. In this paper we propose a novel algorithm for finding the relative frequent itemsets according to the time in a data stream. We also propose a method to save frequent items and sub-frequent items in order to take limited memory into account and a method to update time variant frequent items. By applying the proposed technique, we can improve the accuracy of searching for a change in the frequent itemsets according to the time in a data stream. Moreover, it will be able to use the limited memory space efficiently and store all frequent itemsets.

Keywords: Data Stream, Frequent Itemsets, Data Mining.

1 Introduction

Recently, due to technical improvements of storage devices and network developments, data continues to increase in a very short time. Huge amount of data, for example, have been generated in many fields of applications, such as network invasion detection, sensor network and e-commerce. Many efforts have been made to extract valuable information from such application environments. One of the key realms of research is to extract information from data streams through a data mining method.

A data stream is data that continues to be inputted at high speed. In data streams, data continues to increase at high speed. So, there are two requirements for dealing with data streams by a data mining method.

First, since it is impossible to store large increments of data in a limited space, a new method is required to efficiently store data without losing information by using memory space flexibly.

Second, it is imperative to generate mining results at request because data streams generate large amounts of data in a short time period and the result of mining must be output immediately. This implies that the mining result must be generated by one reading of each transaction of data streams.[3]

One of the most fundamental challenges of data streams is to find frequent itemsets [4]. The conventional data mining method takes a lot of time and memory because it initially reads off from static transactions to compose candidates for frequent itemsets, and then searches for items that have values higher than the defined threshold.

In data streams, items that weren't regarded as frequent itemsets can be converted over time. Thus, it is necessary to update actively and store the frequency of each item. In addition, all unit items cannot be stored because of continuous inputs of too much data in data streams, which makes it improper to apply the conventional data mining method. Many researchers come up with new algorithms that can search for frequent itemsets in data streams [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

Frequent itemset searching methods in data streams, however, can't guarantee the reliability of the search because they merely aggregate and search for frequent itemsets or find them within a time, or a range, after randomly setting a certain size of sliding window. In addition, they again aggregate the total frequent itemsets as time goes by, which results in failure to demonstrate efficiently current changes in frequent itemsets and overlooks frequent itemsets converted over time.

To solve these problems, this paper proposes a new mining method which searches for frequent itemsets more efficiently, taking into account time in data streams.

This paper has 5 sections including the introduction. Section 2 describes related works and section 3 proposes a method to search for frequent itemsets. Performance of the proposed method is evaluated in section 4 through various experiments. Section 5 gives the conclusion and suggestions for improvement.

2 Relate Works

Conventional data mining methods search frequent itemsets by searching max itemsets with a support higher than the predefined minimum support, which were derived by simple searching transactions of the database. Most of algorithms on frequent itemsets are based on the principle of Apriori [1]. This principle states that all subset of frequent itemsets must also be frequent itemsets. When a frequent itemset has a full rank of n, the Apriori algorithm searches up to $n+1$ to generate candidate sets and then begins to search a frequent itemset. Therefore, the Apriori algorithm requires a big memory and a lot of time due to repetitive database searches.

On the other hand, the FP-growth using a divide-and-conquer method does not generate candidate sets [8].

FP-growth is very efficient in mining either long or short frequent itemsets and has a feature of expandability. It also proves to be much faster than the Apriori algorithm. But, both methods must search data sets more than once and must re-search the whole database every time whenever a new transaction takes place. Moreover, when a data set continues to increase at a fast pace, performance drops due to a limited memory.

Data is being generated very quickly due to advanced storage devices and network development. These data are called data stream. Many researchers are studying frequent itemset searching methods in data streams [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

The Count Sketch algorithm focuses on frequency of unit items in data streams [4], and the Lossy Counting algorithm searches for frequent itemsets in data streams only when a minimum support and a maximum allowable error are given [9]. These algorithms only focus on finding the frequent itemsets without taking account of time.

The Moment algorithm, which can search for not only frequent itemsets in a sliding window using limited available memory, but also items close to frequent itemsets. This algorithm uses a tree structure similar to a prefix tree known as CET (Closed Enumeration Tree) [6]. CET stores maximally frequent itemsets, frequent itemsets and approximate frequent itemsets. Thus, it can recognize the changes in frequent itemsets over time and search for frequent itemsets managing the memory efficiently.

The FP-stream algorithm is a revision of the FP-growth algorithm to better fit for data streams [7]. This algorithm classifies data sets into a frequent itemset, a sub frequent itemset and a non-frequent itemset by using a minimum support and maximum allowable errors. It uses a Pattern tree and a tilted time window. Also, it accumulates frequent itemsets up to the current point by storing them in a pattern tree and utilizing a fixed window, a tilted time window and recognizes the latest changes in frequent itemsets efficiently. However, unfixed frequent itemsets are difficult to search for since it uses only fixed time slots.

3 Frequent Itemset Search in Data Streams

3.1 Searching for Relatively Frequent Itemsets

In data streams, data continues to accumulate at a fast rate. This is why the data cannot be stored in a conventional way. Also, a data stream is affected by time, changing the frequent itemset over time. Therefore Non-frequent itemsets can't be discarded or overlooked. To resolve such problems effectively, we classifies data into 3 groups, *frequent itemsets*, *sub frequent itemsets*, and *non-frequent itemsets*, by using a predetermined minimum support and maximum support error, as in the FP-stream[7] algorithm. The FP-stream algorithm uses fixed time window, but our method uses unfixed time window due to find relative frequent itemset. If the frequency is greater than the minimum support value, it is considered *a frequent itemset*. When the frequency is below the minimum support value but has a greater error than the predetermined maximum error, it is classified as *a sub frequent itemset*. Data is disregarded when it has less value than the maximum support error.

The conventional methods search frequent itemsets only by comparing the aggregate values between frequent itemsets and sub frequent itemsets. Hence, time-sensitive and relatively frequent itemsets cannot be accounted for. This implies a necessity of a method to take account of items with a relative frequency of appearance higher than the currently frequent itemsets, even though a value aggregated up to now is smaller than the currently frequent itemsets. Therefore, we define the relatively frequent itemsets as follows:

A relatively frequent itemset is defined as a set of items having a frequency smaller than total frequency of the currently frequent itemsets but greater than current frequency (not total frequency) of the currently frequent itemset, f refers to the currently frequent itemset.

Table 1. Elements of relative frequent itemset

Symbol	Mean
N	Total number of transactions
m	Number of consecutive transactions
f	A frequent itemset
h_t	Time that occurs t'th transactions
T_t	t'th transaction
R_t	Relative Frequent itemset at time of t'th transactions
$A_i(x)$	Appearance function of x in i'th transactions
$F(x)$	Frequency of item x
$C_{m,t}(x)$	Interval of appearance of item x with fixed m and time t'th transactions
$E_{m,t}(x)$	Relative frequency of item x with fixed m and time t'th transacions

$$R_t = \{x \mid F(f) > F(x),\ E_{m,t}(x) > E_{m,t}(f)\} \tag{1}$$

A relative frequency can be regarded as the number of consecutive transactions (m) divided by the sum of differences among intervals of appearances. This is the starting point of finding the relatively frequent itemsets.

$$E_{m,t}(x) = \frac{m}{C_{m,t}(x)}\ ,\ C_{m,t}(x) = h_t - h_{t-m+1} \tag{2}$$

Also, a frequency is an aggregate value of appearance of a specific item in continuous transactions.

$$F(x) = \sum_{i=1}^{N} A_i(x)\,,\ A_i(x) = \begin{cases} 1 & x \in T_i \\ 0 & otherwise \end{cases} \tag{3}$$

Also, the difference of each interval of appearance is meaningful in such that it can determine the time when relatively frequent itemsets exist. It represents a time difference between its current appearance time and the previous appearance time. The y_t implies the point of time when a transaction T_t containing item x happens.

We examine relative frequent itemsets based upon the abovementioned materials. Figure 1 depicts the concept of relatively frequent itemsets. Where each x, y, z is a frequent itemset with a value greater than the minimum support or a sub frequent itemset with a value smaller than the minimum support but greater than the maximal support error, and each dot on the arrow is the frequency. T_t stands for transactions up to now. Here, window size is user-defined because window size effects the result, also, the window not overlapped previous that. In the first row *x*, we see that the frequency is 28, the highest among the three, implying a frequent itemset. The second and third rows are sub frequent itemsets. By examining Figure 1, we see that the first item shows high frequency in the early period but is dispersed gradually as time goes by. The second row shows a sharp increase in frequency in the middle of the row. Seemingly, it makes more sense to designate the second item as the frequent itemset in this specific time period since it appears more frequently than the past, and even more than the first item which is the currently frequent itemset. Thus, the second item becomes the relatively frequent itemset compared to the first item in the middle part of the time period.

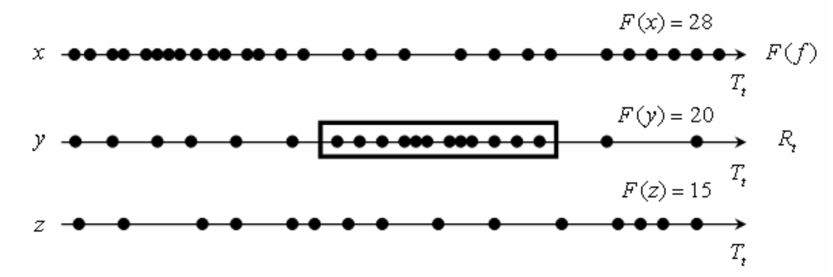


Fig. 1. Concept of relative frequent itemset

The important point here is to find the time when it begins to become a relatively frequent itemset. Therefore, this study suggests a standard for relatively frequent itemsets as follows:

1. A relative frequency must be greater than the relative frequency of frequent itemsets.

$$E_{m,t}(f) < E_{m,t}(x) \tag{4}$$

2. The appearance time of a relatively frequent itemset is the interval from the time(*a*) to the time(*b*). *a* is the latest time at which the currently relative frequency is larger than the just previous relative frequency. *b* is the latest time at which the currently relative frequency is less than zero.

$$E_{m,a-1}(x) \le E_{m,a}(x)\,,\ E_{m,b-1}(x) > E_{m,b}(x),\ a > b \tag{5}$$

That is, it is the time period when a relatively frequency of frequent itemsets becomes higher and the interval between each appearance has becomes very short, compared to the frequent itemset. This study takes the relative frequencies only to tenths for generalizing the numbers and easing the calculation process.

3.2 Storing Method Using the FP-Tree Algorithm

In this section, we suggest an efficient storing method using the FP-Tree algorithm of a prefix tree structure that can maintain and manage all frequent itemsets and relatively frequent itemsets within a limited memory.

As stated earlier, a data stream is assumed to be an infinite set of data. It follows that all data cannot be stored. Hence, the FP-Tree stores only three kinds of key relevant information, items, frequency and TID, to maintain and manage frequent itemsets and relative frequent itemsets efficiently. Here, items mean either frequent itemsets or relatively frequent items, and frequency refers to the total number of appearance of such items. TID refers to the current transaction id, which is used to measure the starting point of a relatively frequent itemset.

The FP-Tree algorithm has four main steps, each of which is reiterated when a new transaction is added.

The first step is a phase where data stream transactions are searched to update the frequency of each item. The total dataset $|S_N|$ is incremented by 1 when a new transaction occurs. The frequency and TID values are updated when items appearing in such a new transaction happen to exist in a node of FP-Tree.

The second step is a phase where a sub frequent itemset is added. When there is no node in the FP-Tree corresponding to the items that appear in the new transaction, items with a value smaller than the minimum support but greater than the maximum support error are added to the FP-Tree node as sub frequent itemsets. Those with errors smaller than the maximum allowable error are discarded in that it has little chance to become frequent itemsets. It results in less performing time because it uses less memory and reduces a process to insert new nodes to the FP-Tree.

The third step is searching for the currently frequent itemset. Upon users' request, this step outputs those items' information, items, frequency and TID, with the greatest total frequency up to now and a support more than the minimum support value.

The fourth step is searching for the relatively frequent itemset. When the relative frequency of a frequent itemset becomes bigger, that is, when the intervals become much shorter, searching items that appear relatively more frequently than the current frequent itemset starts at this step.

This method guarantees reliability and accuracy by searching not only the currently frequent itemset but also the relatively frequent itemset which can be easily overlooked, moreover, it can efficiently utilize the limited memory because it holds only three kinds of information, items, frequency and TID.

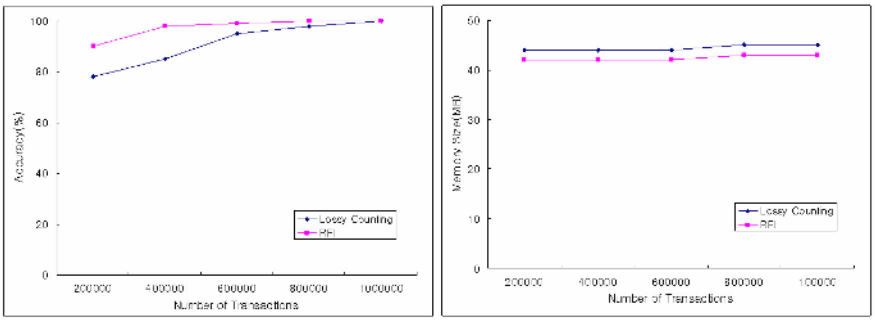
4 Test and Evaluation

Our algorithm was written in C and compiled using gcc. The stream data was generated by the IBM test data generator [1]. Generated datasets are T10.I4.D1000K and T15.I6.1000K, where the numbers denotes the average transaction size (T), the average large itemset size (I) and the number of transactions respectively.

The experiment evaluated the accuracy and memory space of the proposed method in comparison with Lossy Counting algorithm [9]. Jeffrey Xu Yu's method is using less memory than Lossy Counting algorithm[10]. James Cheng's method runs faster than Lossy Counting algorithm[5]. Figure 2 shows the results of this experiment.

The Lossy counting algorithm using a minimum support finds frequent itemsets through the aggregation. Therefore if the number of transactions is small, then the accuracy of the Lossy counting algorithm is low. On the other hand, the proposed method finds frequent itemsets with relative frequent itemsets as time goes by.

Although the number of transactions is small, the proposed method achieves better accuracy than the Lossy counting algorithm. As the number of transactions increases, the accuracy of both methods is getting higher because transactions include more frequent itemsets.



(a) (b)

Fig. 2. The result of experiment according to the accuracy and memory space; a) Accuracy according to the number of transactions, b) Memory size according to the number of transactions

Fig. 2 (b) shows the memory size according to the number of transactions. Above all, we must find the optimal minimum support value. Because too low minimum support value leads to a wide range of permission for frequent itemsets and too high minimum support value leads to a narrow range of permission for those. Here, minimum support value sets 0.3 according to our experiments. In the proposed method, the FP-Tree stores only three kinds of key relevant information, items, frequency and TID, to maintain and manage frequent itemsets and relative frequent itemsets efficiently. Therefore, it is clear that the proposed method uses less memory space than the Lossy counting algorithm.

As shown above, we can see that the proposed method find the frequent itemsets more accurately and our method is able to compute all frequent itemsets using less memory than the Lossy Counting algorithm.

5 Conclusion

One of the most fundamental problems in data streams is how to search frequent itemsets generated from the stream. The frequent itemsets change because the stream itself is affected by time. Therefore Non-frequent itemsets can not be discarded. Data streams may be defined as infinite sets of data, making it impossible to store every item in the stream. To solve these problems, we introduced relatively frequent items and the FP-Tree algorithm in this paper.

We classify data into 3 groups, frequent itemsets, sub frequent itemsets, and non-frequent itemsets, by using a predetermined minimum support and maximum support error. The suggested algorithm computes the total number of frequencies and relative frequencies derived from intervals between frequencies using unfixed time window. It searches the frequent itemset and, by comparing relative frequencies of sub frequent itemsets, the relatively frequent itemset that can be easily overlooked. The FP-Tree tries to efficiently manage frequent itemsets and sub frequent itemsets by storing only 3 kinds of information, items, frequency and TID. The FP-Tree has four main steps, updating the frequency of each item, adding a sub frequent itemset, searching for the currently frequent itemset, searching for the relative frequent itemset. All these enable us to search time-sensitive frequent itemsets, to increase reliability of the searches and to utilize limited memory efficiently.

Acknowledgement. This work was supported by INHA university Research Grant.

References

1. R. Agrawal and R. Srikant.: Fast algorithms for mining association rules. In Proc. of the 20th Intl. Conf. on Very Large Databases (1994)
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom.: Models and issues in data stream systems. In Proc. of SIGMOD/PODS, Madison, Wisconsin, USA (2002) 1–16
3. J. Chang. and W. Lee.: Finding recent frequent itemsets adaptively over online data Streams. In Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery & Data Mining, Washington, DC (2003) 226–235
4. M. Charikar, K. Chen, and M. Farach-Colton.: Finding frequent items in data streams. In Procedings of the International Colloquium on Automata, Languages and Programming (2002) 693–703
5. James Cheng, Yiping Ke, and Wilfred Ng, "Maintaining Frequent Itemsets over High-Speed Data Streams", PAKDD 2006
6. Y. Chi, H. Wang, P. Yu, and R. Muntz.: MOMENT: Maintaining closed frequent itemsets over a stream sliding window. In Proc. of 4th IEEE Intl. Conf. on Data Mining, Brighton, UK (2004) 59–66
7. C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT (2003)
8. J. Han, J. Pei, and Y. Yin.: Mining frequent patterns without candidate generation. In Proceedings of the SIGMOD Conference, Dallas, Texas, USA: ACM Press (2000) 1–12
9. G. Manku and R. Motwani.: Approximate frequency counts over data streams. In Proceedings of 28th International Conference on Very Large Data Bases (2002) 346–357
10. Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Aoying Zhou, "False Positive or False Negative : Mining Frequent Itemsets from High Speed Transactional Data Streams", VLDB 2004, 204-215
11. D. Zhang, D. Gunopulos, V. J. Tsotras and B. Seeger.: Temporal Aggregation over Data Streams using Multiple Granlarities. Proc. of 8th International Conference on Extending Database Technology (EDBT), Prague, Czech Republic (2002)