IJCES International Journal of Computer Engineering Science ,
Volume 2 Issue 2 ISSN : 2250:3439
https://sites.google.com/site/ijcesjournal http://www.ijces.com/

MS: Multiple Segments with Combinatorial Approach for Mining Frequent Itemsets Over Data Streams

K Jothimani[1], S. Antony Selvadoss Thanmani [2]

[1] Research Scholar, Research Department of Computer Science,
NGM College, 90, Palghat Road, Pollachi - 642 001
Coimbatore District, Tamilnadu, INDIA
Email: jothi1083@yahoo.co.in
[2] Professor and Head, Research Department of Computer Science,
NGM College, 90, Palghat Road, Pollachi - 642 001
Coimbatore District, Tamilnadu, INDIA
Email: selvdoss@yahoo.com

Abstract. Mining frequent itemsets in data stream applications is beneficial for a number of purposes such as knowledge discovery, trend learning, fraud detection, transaction prediction and estimation. In data streams, new data are continuously coming as time advances. It is costly even impossible to store all streaming data received so far due to the memory constraint. It is assumed that the stream can only be scanned once and hence if an item is passed, it can not be revisited, unless it is stored in main memory. Storing large parts of the stream, however, is not possible because the amount of data passing by is typically huge. In this paper, we study the problem of finding frequent items in a continuous stream of items. A new frequency measure is introduced, based on a variable window length. We study the properties of the new method, and propose an incremental algorithm that allows producing the frequent itemsets immediately at any time. In our method, we used multiple segments for handling different size of windows. By storing these segments in a data structure, the usage of memory can be optimized. Our experiments show that our algorithm performs much better in optimizing memory usage and mining only the most recent patterns in very less time.

Keywords: Data stream mining, Frequent itemset, Segment-based and Variable length Window.

1 Introduction

Frequent itemset mining is a KDD technique which is the basic of many other techniques, such as association rule mining, sequence pattern mining, classification, clustering and so on. A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to this reason, it is impossible to maintain all the elements of data streams [1]. This rapid generation of continuous streams of information has challenged our storage, computation and communication capabilities in computing systems. The main challenge is that 'data-intensive' mining is constrained by limited resources of time, memory, and sample size.

1.1 Data Streams

Data Stream mining refers to informational structure extraction as models and patterns from continuous data streams. Data Streams have different challenges in many aspects, such as computational, storage, querying and mining. Based on last researches, because of data stream requirements, it is necessary to design new techniques to replace the old ones. Traditional methods would require the data to be first stored and then processed off-line using complex algorithms that make several pass over the data, but data stream is infinite and data generates with high rates, so it is impossible to store it [12].

Data from sensors like weather stations is an example of fixed-sized data, whereas again, market basket data are an example of variable size data, because each basket contains a different number of items. By contrast, sensor measurements have a fixed size, as each set of measurements contains a fixed set of dimensions, like temperature, precipitation, etc.

A typical approach for dealing is based on the use of so-called sliding windows. The algorithm keeps a window of size W containing the last W data items that have arrived (say, in the last W time steps). When a new item arrives, the oldest element in the window is deleted to make place for it. The summary of the Data Stream is at every moment computed or rebuilt from the data in the window only. If W is of moderate size, this essentially takes care of the requirement to use low memory. The type of objects i.e. windows in a stream impacts the way the data stream is processed. This is due to two facts: We have to handle windows of different types differently, and we are able to tailor the processing to the specific properties of the objects. Hence, for our focus, interesting properties are the size of the windows, what information the objects represent and how they relate to other windows in the stream.

2 Related Work

Recently proposed mining approaches for event logs have often been based on some well-known algorithm for mining frequent itemsets (like Apriori or FPgrowth).In this section we will discuss the frequent itemset mining problem and prominent algorithms for finding optimistic solution in addressing this problem.

The sliding window method processes the incoming stream data transaction by transaction. Each time when a new transaction is inserted into the window, the itemsets contained in that transaction are updated into the data structure incrementally. Next, the oldest transaction in the original window is dropped out, and the effect of those itemsets contained in it is also deleted. The sliding window method [4] also has a periodical operation to prune away unpromising itemsets from its data structure, and the frequent itemsets are output as mining result whenever a user requests.

Two types of sliding widow, i.e., *transaction- sensitive sliding window* and *time-sensitive sliding window*, are used in mining data streams[1][2]. The basic processing unit of window sliding of first type is an expired transaction while the basic unit of window sliding of second one is a time unit, such as a minute or an hour. In the damped window model, recent sliding windows are more important than previous ones.

As long as the window size is reasonably large, and the conceptual drifts in the stream is not too dramatic, most itemsets do not change their status (from frequent to non frequent or from non-frequent to frequent).[18] In other words, the effects of transactions moving in and out of a window offset each other and usually do not cause change of status of many involved nodes.

To find frequent itemsets on a data stream, we maintain a data structure that models the current frequent itemsets. We update the data structure incrementally. The combinatorial explosion problem of mining frequent itemsets becomes even more serious in the streaming environment. As a result, on the one hand, we cannot afford keeping track of all itemsets or even frequent itemsets, because of time and space constraints. On the other hand, any omission (for instance, maintaining only *M*, *C*, or *F* instead of all itemsets) may prevent us from discovering future frequent itemsets. Thus, the challenge lies in designing a compact data structure which does not lose information of any frequent itemset over a sliding window.

3 Problem Description

Let $I = \{i1,..., in\}$ be a set of items. If X _ I, X is called an *itemset*, and if $|X| = k$, X is also called a *k-itemset*. A *transaction* is a tuple $T = (tid, X)$ where *tid* is a transaction identifier and X is an itemset. A *transaction database D* is a set of transactions, and the *cover* of an itemset X is the set of identifiers of transactions that contain X: $cover(X) = \{tid \mid (tid, Y)$ _ D, X _ $Y\}$. The *support* of an itemset X is defined as the number of elements in its cover: $supp(X) = |cover(X)|$. The task of *mining frequent itemsets* is formulated as follows – given the transaction database D and the *support threshold s*, find itemsets $\{X \mid supp(X)$ _ $s\}$ and their supports [13] (each such set is called a *frequent itemset*).

A specific point in time is modeled as the number of base time units that have elapsed since the epoch. We denote points in time with t, or ti if we want to distinguish several points in time. From the base time unit, other time units with coarser granularities can be composed[17] .Let $I = \{x1, x2, \ldots, xz\}$ be a set of items (or attributes). An itemset (or a pattern) X is a subset of I and written as $X=xi\, xj\ldots xm$. The length (i.e., number of items) of an itemset X is denoted by $|X|$. A transaction, T, is an itemset and T supports an itemset, X, if $X \subseteq T$.

A transactional data stream is a sequence of continuously incoming transactions. A segment, *S*, is a sequence of various number of transactions, and the size of *S* is indicated by *s*. A window, *W*, in the stream is a set of successive *w* transactions, where $w \geq s$. A sliding window in the stream is a window of different number of most recent *w* transactions which slides forward for every transaction or every segment of transactions. We adopt the notation *IL* to denote all the itemsets of length *l* together with their respective counts in a set of transactions (e.g., over *W* or *S*). In addition, we use *Tn* and *Sn* to denote the latest transaction and segment in the current window, respectively. Thus, the *current window* is either $W = < Tn\text{-}w+1, \ldots, Tn >$ or $W = < Sn\text{-}m+1, \ldots, Sn >$, where *w* and *m* denote the size of *W* and the number of segments in *W*, respectively.

Given a data stream in which every incoming transaction has its items arranged in order, and a *changeable value* of *ms* specified by the user, the problem of mining FIs over a sliding window in the stream is to find out the set of frequent itemsets over the window at different slides.

We remark that most of the existing stream mining methods[3] [5] [9] work with a basic hypothesis that they know the user-specified *ms* in advance, and this parameter will remain un changed all the time before the stream terminates. This hypothesis may be unreasonable, since in general, a user may wish to tune the value of *ms* each time he/she makes a query for the purpose of obtaining a more preferable mining result.

Apart from the type of data in the stream, one important property of the data stream is the stream rate. It can be measured in objects per time unit, like objects per minute or per hour. Another possible measure is the amount of data per time unit. This measure is especially useful in settings with objects of variable size. Taking the size of the objects into account is more accurate in such a setting, because it better reflects the actual load of the data stream processing system

.

4 Multiple Segments For Handling Streams

The transaction-by-transaction sliding of a window leads to excessively high frequency of processing. In addition, since the transit of a data stream is usually at a high speed, and the impact of one single transaction to the entire set of transactions (in the current window) is very negligible, making it reasonable to handle the window sliding in a wider magnitude.[20] Therefore, for an incoming transactional data stream to be mined, we propose to process on a *segment-oriented window sliding*. We conceptually divide the sliding window further into several, say, *m*, segments.

Each of the *m* segments contains a set of successive transactions and is of the different size *s* (i.e., contains the number of *s* transactions). Besides, n each segment, the summary of transactions belonging to that segment is stored in the data structure. Multiple segments can be used for storing the transactions. Thus, even if various size of windows w will be coming, it can be easily handled by the segments.

By taking this segment-based manner of sliding, each time when a segment in-out operation occurs, we delete (or drop out) the earliest segment, which contains the summary of transactions of that segment, from the current window at each sliding. As a result, we need not to maintain the whole transactions within the current window in memory all along to support window sliding.

In addition, we remark that the parameter m directly affects the consumption of memory. A larger value of m means the window will slide (update) more frequently, while the increasing overhead of memory space is also considerable. Let us consider the following example. The first group of itemsets that needs to be restored contains those k-itemsets that have the same $(k - 1)$-prefix as some itemset in the current frequent set.

Consider then in pass k, an itemset X in the Multiple Segments (MS) and an itemset Y in the current frequent set such that $|X| > k$. Suppose that the first $k - 1$ items of Y are in X and the $(k - 1)$st item of Y is equal to the j th item of X. We obtain the k-subsets of X that have the same $(k - 1)$-prefix as Y by taking one item of X that has an index greater than j and combining it with the first $k-1$ items of Y, thus getting one of these k-subsets. After these k-itemsets are found, we recover candidates by combining them with itemset Y.

```
Algorithm: The retrieving procedure
Input: $Ck+1$ from join procedure, $Lk$, and current MS
Output: a complete transaction set $Tk+1$
1. for all itemsets $l$ in $Lk$
2. for all itemsets $m$ in MS
3. if the first $k-1$ items in $l$ are also in $m$
4. /* suppose $m.itemj = l.itemk-1$ */
5. for $i$ from $j+1$ to $|m|$
6. $Tk+1 := Tk+1 \cup \{\{l.item1, l.item2, \ldots, l.itemk, m.itemi\}\}$
```

There are four possible cases when we combine two frequent k-itemsets, say I and J, which have the same $(k - 1)$-prefix, to generate a $(k + 1)$-itemset as a new preliminary candidate. In this proof, we will show that, even though that we remove the subsets of the MS from the current frequent set, our new candidate generation algorithm will handle all these cases correctly

```
Algorithm: The Multiple Segment algorithm
Input: a database and a user-defined minimum support
Output: MS which contains all maximal frequent itemsets
1. $L0 := \varnothing$ ; $k := 1$; $T1 := \{\{i\} \mid i \in I\}$
2. FCS: $= \{\{1, 2, \ldots, n\}\}$; MS: $= \varnothing$
3. while $Tk \neq \varnothing$
4. read data streams and count supports for $Tk$ and MS
5. remove frequent itemsets from FCS and add them to MS
```

```
6. $Lk$ := {frequent itemsets in T$k$} \ {subsets of MS}
7. $Sk$ := {infrequent itemsets in T$k$}
8. Call the FCS-gen algorithm if $Sk \neq \varnothing$
9. call the join procedure to generate $Tk+1$
10. if any frequent itemset in T$k$ is removed in line 6
    11. call retrieving procedure to recover transactional
        itemsets T$k$+1
12. $k := k + 1$
13. end-while
14. return MS
```

Theorem 1 *The Multiple Segment algorithm generates all frequent itemsets over data streams.*

Proof: The Multiple Segment Algorithm will explicitly or implicitly discover all frequent itemsets. Therefore, the Multiple Segment algorithm generates frequent itemsets. Maintaining all the data streams in the main memory requires too much space. So we have to store only relevant itemsets and drop itemsets when the tail-dropping condition holds.

When all windows of different sizes frequent itemsets are dropped the entire itemset is dropped from segments. As the result of the tail-dropping we no longer have an exact support over L, rather an approximate support. Now let us denote *supportL(X)* the frequency of the itemset X in all batches and *supportL'(X)* the approximate frequency. With $\varepsilon \ll \sigma$ this approximation is assured to be less than the actual frequency according to the following inequality's in [3]: $SupprtL(X) - \varepsilon L \leq SupportL'(X) \leq SupportL(X)$

5 Experimental Results

We evaluate the performance of our MS algorithm by varying the usage of the memory space. We also analyze the execution time. The simulation is performed in Visual C++ and conducted in a machine with a 3GHz CPU and 1GB memory. We use two sets of synthetic databases from an IBM Quest data generator.

The following figures illustrates some of the parameters that we have controlled: the size of the sliding window 10K, the average size of the transaction T, the average size of the frequent itemsets I and we randomly generate the weight of each item in transaction, ranging from 0.2 to 0.8. Synthetic dataset T10I4D100K denotes the average size of the transactions and I the average number of frequent itemsets.

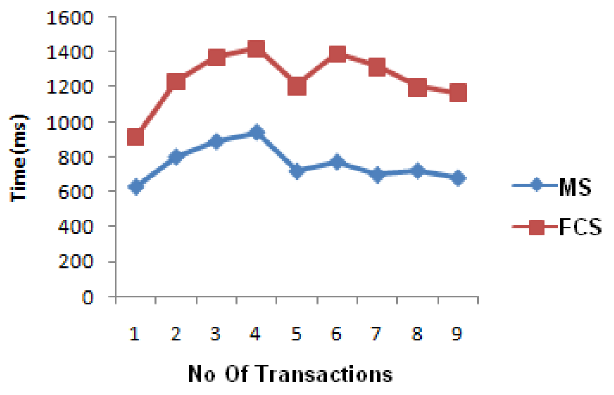

Fig.1 Compare execution time between FCS and MS T10I4D100K

In this experiment, we examine the execution time and memory usage between MS and FCS Frequent Candidate Set by dataset *T*10*I*4*D*100*K*. In Fig. 1, we can see that the execution time incurred by MS is quite steady and is shorter than that of FCS. The experiment shows that MS performs more efficiently than FCS. In Fig. 2, the memory usage of MS is more stable and smaller than that of FCS. This is because MS delete a lot of common nodes between various regions does not need to search these spaces for some computations and enumerate all subsets of each incoming transaction. The amount of all subsets is an enormous exponential number for long transaction. Hence, it shows that SFIDS is more suitable for mining frequent itemsets in data streams.

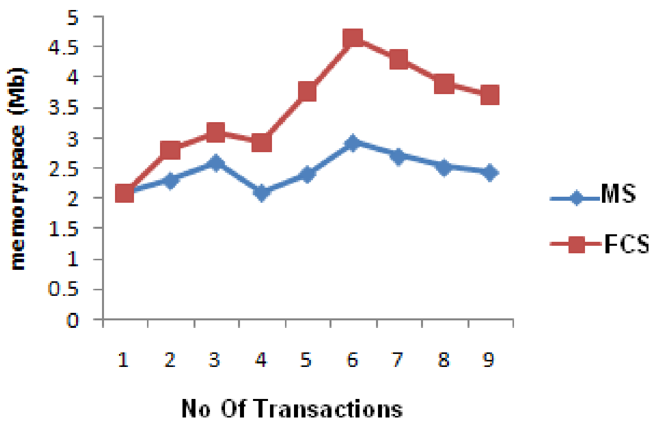

Fig. 2 Compare space consumption between FCS and MS *T*10*I*4*D*100*K*.

5 Conclusions

In this paper, a structure was designed to dynamically maintain the up to date contents of data streams by scanning it only once, and a new method MS was proposed to mine the frequent patterns in a sliding window. This method could answer a request with no false negative. We study the problem of mining frequent itemsets over the sliding window of a transactional data stream.

Based on applying the theory of Multiple Segments for varying different windows, we devise and propose an algorithm called M*S* finding frequent itemsets over data streams. It conceptually divides the sliding window into *segments* and handles the sliding of window in a segment-based manner. According to the experimental results, MS is quite efficient and possesses good scalability with varying minimum support threshold. Extensive experimental results show that MS decrease required time for processing batches and amount of memory for storing history of data.

We compare our algorithm with FCS algorithm and show that MS perform better than FCS in various conditions. The main contributions on evolving data streams are giving a unified framework for data mining with time change detection with varying length of windows. Experiments show that the proposed algorithm not only attain highly accurate mining results, but also run significant faster and consume less memory than existing algorithms for mining frequent itemsets over online data streams.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proceedings of the 1993 International Conference on Management of Data, pp. 207-216, 1993.
2. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499, 1994
3. M.N. Garofalakis, J. Gehrke, & R. Rastogi, Querying and mining data streams: you only get one look (A Tutorial), Proc. 2002 ACM SIGMOD Conf. on Management of Data, Madison, Wisconsin, 2002, p. 635.
4. J.H. Chang & W.S. Lee, "A sliding window method for finding recently frequent itemsets over online data streams', Journal of Information Science and Engineering, 20(4), 2004, pp. 753–762
5. Y. Zhu & D. Shasha, "Stat Stream: statistical monitoring of thousands of data streams in real time", Proc. 28th Conf. on Very Large Data Bases, Hong Kong, China, 2002, pp. 358–369.
6. G.S. Manku & R. Motwani, "Approximate frequency counts over data streams", Proc. 28th Conf. on Very Large Data Bases, Hong Kong, China, 2002, pp. 346–357.
7. J.H. Chang & W.S. Lee, "A sliding window method for finding recently frequent itemsets over online data streams", Journal of Information science and Engineering, 20(4), 2004, pp. 753–762.
8. J. Cheng, Y. Ke, & W. Ng," Maintaining frequent itemsets over high-speed data streams", Proc. 10th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Singapore, 2006, pp.462–467.

9. C.K.-S. Leung & Q.I. Khan, "DSTree: a tree structure for the mining of frequent sets from data streams," Proc. 6th IEEE Conf. on Data Mining, Hong Kong, China, 2006, pp. 928–932.
10. Y. Chi, H. Wang, P.S. Yu, & R.R. Muntz, "Moment: maintaining closed frequent itemsets over a stream sliding window", Proc. 4th IEEE Conf. on Data Mining, Brighton, UK, 2004, pp. 59–66.
11. K.-F. Jea & C.-W. Li, "Discovering frequent itemsets over transactional data streams through an efficient and stable approximate approach, Expert Systems with Applications", 36(10), 2009, pp. 12323–12331.
12. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference, was supported in part by the National Science Council in 2006.
13. F. Bodon, "A fast APRIORI implementation", Proc. ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), 2003.
14. Risto Vaarandi, "Tools and Techniques for Event Log Analysis". In SIGMOD Record, Vol. 35, No. 1, June 2005.
15. Frequent Itemset Mining Implementations Repository (FIMI). Available: http://fimi.cs.helsinki.fi/
16. Mahnoosh Kholghi, Mohammadreza Keyvanpour, "An Analytical Framework for Data Stream Mining Techniques Based on Challenges and Requirements", International Journal of Engineering Science and Technology (IJEST) ISSN: 0975-5462 Vol. 3 No. 3 Mar 2011.
17. N. Jiang & L. Gruenwald, "CFI-Stream: mining closed frequent Itemsets in data streams", Proc. 12th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, Philadelphia, PA,USA, 2006, pp. 592–597.
18. Quest Data Mining Synthetic Data Generation Code. Available: http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html
19. H.F Li, S.Y. Lee, M.K. Shan, "An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams", In Proceedings of First International Workshop on Knowledge Discovery in Data Streams 9IWKDDS, 2004.
20. P. Indyk, D. Woodruff, "Optimal approximations of the frequency moments of data streams", Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp.202–208, 2005