ISSN:2231-0711
www.ijcset.net

An Algorithm for Mining Frequent Itemsets

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

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

Abstract— The problem of mining frequent itemsets in streaming data has attracted a lot of attention lately. Even though numerous frequent itemsets mining algorithms have been developed over the past decade, new solutions for handling stream data are still required due to the continuous, unbounded, and ordered sequence of data elements generated at a rapid rate in a data stream. The main challenge in data streams will be constrained by limited resources of time, memory, and sample size. Data mining has traditionally been performed over static datasets, where data mining algorithms can afford to read the input data several times. The goal of this article is to analyze the factors for mining frequent itemsets in theoretical manner. By comparing previous algorithms we propose new method using analytical modelling to determine the factors over data streams.

Keywords— Data Mining, Data Streams Frequent Itemset Mining, Sliding Window.

I. INTRODUCTION

Mining association rules in transaction datasets has been demonstrated to be useful and technically feasible in several application areas, particularly in retail sales [1] and document datasets applications [2]. The management and storage of large datasets have always been a problem to solve. An interesting solution is to use compression algorithms on the data because the presence or absence of an item in a transaction can be stored in a bit. To find an algorithm that compresses the data while maintaining the necessary semantics for the frequent itemsets mining problem is the goal of this work.

Frequent itemset mining is one of the essential data mining tasks. Since it was firstly proposed in [1], various algorithms have been proposed, including Apriori [2] and FP-growth [12] algorithms. Many studies have also demonstrated its application in feature selection and associative classifier construction [18, 9, 14, 17, 3, 26, 8, 24, 5, 6]. If we divide the set of all itemset patterns into a set of equivalence classes, where each equivalence class contains a set of itemset patterns which are supported by the same set of input transactions, the closed itemsets are those maximal ones in each equivalence class. It is evident that the set of closed itemsets is just a subset of all itemset patterns, and thus it is possible to identify some parts of search space which are unpromising to generate any closed itemsets and can be pruned. Thus, closed itemset mining can be potentially more efficient than all itemset mining. Due to the concise representation and high efficiency, many algorithms for mining frequent closed itemsets have been proposed [20, 21, 29, 19, 25, 11].

In each equivalence class of itemset patterns, if we call the minimal ones itemset generators, similarly we get that the set of all itemset generators is a subset of all itemset patterns, and itemset generator mining can be potentially more efficient than all itemset pattern mining too. It is also evident that the average length of itemset generators tends to be smaller than that of all itemset patterns (or closed itemset patterns). Since one of the important applications of frequent itemset mining is to be used for feature selection and associative classifier construction. According to the Minimum Description Length (MDL) Principle, generators are preferable in tasks like inductive inference and classification among the three types of itemset patterns (namely, all itemset patterns, closed itemset patterns.

In solving many application problems on data stream, it is desirable to discount the effect of the old data. One way to handle such problem is to use sliding window models[2]. There are two typical models of sliding window[3]: milestone window model and attenuation window model. H.F.Li[4] made use of NewMoment to maintain the set of frequent closed itemsets in data streams with a transaction-sensitive sliding window. MOMENT by Chi[5] is also a typical algorithm which can decrease the size of the data structure. N.Jiang[6] proposed a novel approach for mining frequent closed itemsets over data streams. Y.Chi[7] introduced a compact data structure, i.e. the closed enumeration tree, to maintain a dynamically selected set of itemsets over a sliding window. The selected itemsets contain a boundary between frequent closed itemsets and the rest of the itemsets. F.J.Ao[8] presented an algorithm named FPCFI-DS for mining closed frequent itemsets in data streams. FPCFI-DS uses a single-pass lexicographical-order FP-Tree-based algorithm with mixed item ordering policy to mine the closed frequent itemsets in the first window, and updates the tree for each sliding window. J.Y.Wang[9] proposed an alternative mining task for mining top-*k* frequent closed itemsets of length no less than *min_l*.

In this paper, an efficient mining algorithm (denoted as EMAFCI) for frequent closed itemsets in data stream is proposed. The algorithm is based on the sliding window model, and uses a Bit Vector Table (denoted as BVTable) where the transactions and itemsets are represented by the column and row vectors respectively. The algorithm firstbuilds the BVTable for the first sliding window. Frequent closed itemsets can be detected by pair-test operations on the binary numbers in the table. After building the first BVTable, the algorithm updates the BVTable for each sliding window. The frequent closed itemsets in the sliding window can be identified from the BVTable. The algorithm is also proposed to modify BVTable when adding and deleting a transaction. The experimental results on synthetic and real data sets indicate that the proposed algorithm needs less time CPU time and memory than other similar methods.

II. Related Work

A. Frequent Closed Itemsets

Let $I = \{i_1, i_2, □, i_m\}$ be a set of distinct data items, and a subset $X □ I$ is called an itemset. Each transaction t is a set of items in I . A data stream $DS = \{(tid_1, t_1), ..., (tid_n, t_n), ...\}$ is an infinite sequence of transactions in which tid_k is the identifier of a transaction and $t_k □ I$ ($k = 1,2,□,n$) is an itemset. For all transactions in a given window of the data stream, the support $sup(X)$ of an itemset X is defined as the number of transactions with X as a subset. In general, the more transactions a sliding window has, a larger amount of frequent itemsets could be produced. In this case, there are many redundancies among those frequent itemsets. For example, in the frequent itemsets$\{acd, ad, a\}$, the only useful information is the set acd according to Apriori property, because it includes ad and a . Frequent closed itemset is a solution to this problem. A frequent itemset X is a closed one if it has no superset $Y □ X$ so that $sup(X) = sup(Y)$ .Frequent closed itemset is a condensed, i.e. both concise and lossless, representation of a collection of frequent itemsets.

B. Sliding Window

The basic idea of mining frequent closed itemset in the sliding window model is that it makes decisions from the recent transactions in a fixed time period instead of all the transactions happened so far. Formally, a new data element arriving at the time t will expire at time $t + w$, in which w is the length of the window. At every time step, when a new transaction comes to the window, the oldest one in the window should be deleted. Since the transactions in the window are updated over time, the frequent itemsets should be renewed accordingly.

III. The EMAFCI Algorithm

In this section, we illustrate the framework of the algorithm EMAFCI for mining frequent closed itemsets in data streams based on the model of sliding window. First we introduce the data structure of BVTable used in the algorithm.

A. *The BVTablet*

The EMAFCI algorithm is based on the data structure of BVTable. To compress the itemsets and the database, VTable consists of a set of binary integer where each bit epresents an item. It consists of three parts, the left, middle and right part. The ith row of the BVTable is a vector (s_i, t_i, c_i) where binary integers s_i, t_i are the left and middle parts respectively, and the right part c_i is the support of the itemset corresponding to the ith row. In the left part of BVTable, each column represents an item and each row is a binary integer corresponding to a candidate itemset.

Denote the jth bit of s_i, as s_{ij}. If the jth item is included in the ith itemset, then $s_{ij} = 1$, otherwise $s_{ij} = 0$. In the middle part of BVTable, each column represents a transaction in the current time window and each row is a binary integer indicating whether the itemset represented by this row is included in the transaction or not. Denote the jth bit of r_i , as r_{ij} . If the ith itemset is included in the jth transaction, then $r_{ij} = 1$, otherwise $r_{ij} = 0$. In the right part of the ith row, $c_i = H(r_i)$ is the support of the itemset corresponding to the ith row, here $H(r_i)$ is the number of bits "1" in r_j . Since c_i can be calculated easily from r_i , it doesn't need to be physically stored in the memory. To mine the frequent closed itemsets from the current sliding window, the algorithm EMAFCI first builds a BVTable for all 1-itemsets that are denoted as L_1 . Based on L_1 , all the frequent 2-itemsets can be detected and L_2 can be built. Repeat this procedure until all the r -itemsets are detected, and here r is the maximum length of the transactions. Here, L_1 consists of all the 1-itemsets that include the frequent and nonfrequent ones in order to store all the transactions in the current window.

B. Framework of algorithm

The algorithm EMAFCI receives a transaction from the data stream at each time step, and forms a new sliding window by adding this new transaction into the window and emitting the oldest one. To identify the requent closed itemsets in this new sliding window, EMAFCI should modify the BVTable accordingly. Since two adjacent windows are overlapped except the added and deleted transactions, the frequent closed itemsets of the two windows do not change abruptly. EMAFCI needs only to process the part of BVTable involving these two transactions. Therefore, procedures are proposed to modify BVTable when adding and deleting a transaction. The framework of algorithm EMAFCI is as follows:

```
Algorithm: EMAFCI( D, L )
Input: D : the data stream;
Output: L : the BVTable;
Begin
        BuildFirstBVTable( D, n, L );
        while not the end of the stream do
        Receive a new transaction x from the stream;
        DeleteTransFCI( L );
        AddTransFCI( L , x );
end while
End
```

C. Build the BVTable for the first window

The ItemList for the first window is constructed by a procedure BuildFirstBVTable(). First the BVTable for the 1-itemsets L_1 should be generated. Then the BVTable L_2 for the frequent 2-itemsets are generated from the frequent 1-itemsets by performing bitwise OR operation (denoted as □) in the left part of the BVTable and AND operation (denoted as ∩) in the middle part of the BVTable. Similarly, the frequent 3-itemsets and 4-itemsets are generated from the 2-itemsets. Iterate this procedure until all the frequent itemsets are detected. Let n be the maximum length of the transactions, so the maximum number of such iterations is $\log_2 n$. Among the frequent itemsets detected, the sub-itemsets with the same support are labeled "*", because they are nonclosed frequent itemsets. Details of the operation are as follows.

We denote the BVTable after the kth iteration as L_k . Let $m = 2^{k-2}$, and denote the set of all frequent j -itemsets as C_m ,

then L_k consists of all the frequent itemsets $C_{m+1} \ldots C_{2m}$. To construct L_{k+1}, pair-test operation should be performed on each pair of frequent itemsets to generate possible larger frequent itemset. Let (s_i, t_i, c_i) and (s_i, t_j, c_j) be two frequent itemsets in L_k, then a pair-test operation can generate an itemset $(s_i □ s_j, t_i □ t_j, c)$, here $c = H(t_i \cap t_j)$ is the number of bits "1" in $t_i \cap t_j$. In each iteration we generate the frequent itemsets in L_{k+1} based on L_k which consists of the frequent itemsets $C_{m+1} \ldots C_{2m}$, and entirely ignore the itemsets in
$L_{k-1}, L_{k-2} \ldots L_1$.

IV. Experimental Results

In order to evaluate the performance of our algorithm EMAFCI, we test it and compare the memory requirement, the processing time for the first window andeach sliding window with the algorithm Moment. The experiments are performed on a Pentium 2.4GHz S4800A (AMD Opteron 880) CPU with 4GB RAM memory, 300GB hard drive. The algorithm is coded using the VC + + 6.0 on Linux operating system.

A. Data Set

In our experiments, we use the real database Mushroom and the synthetic databases proposed by Agrawal and Srikant for evaluating the algorithms. Mushroom which can be downloaded from [13] is a dense dataset with 8124 transactions,. Database T40I5D10K which produces data simulating the transactions of retail stores is generated by using the synthetic data generator described by Agrawal et al.

B. Experimental results and analysis

(1) Mushroom

We have adopted the commonly used parameters: the number of transactions is set as 8124 while the size of window as 8000. We report the average performance over 124 consecutive sliding windows.

In Fig.1, the times for processing the first window by the algorithms of Moment and EMAFCI are compared. From Fig.1 we can see that when the *support=minsup/*8000 is set between 1 and 0.8, the processing time of EMAFCI is equal to that of Moment. But when the *support* is lower than 0.8, the time of Moment is much more than that of EMAFCI. For instance, when *support* is set as 0.2, time cost of Moment is more than 200s, while the time of EMAFCI is less than 50s.

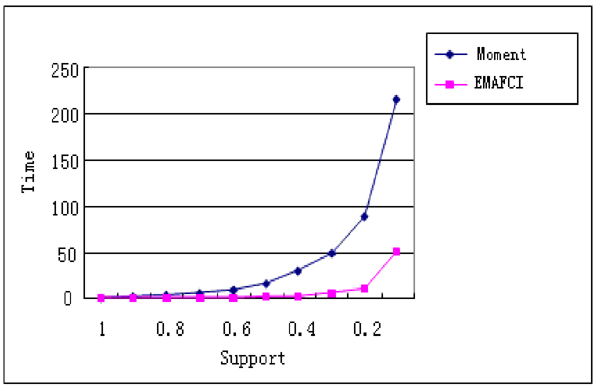

Figure 1. The time of processing the first window of Mushroom by EMAFCI compared with Moment

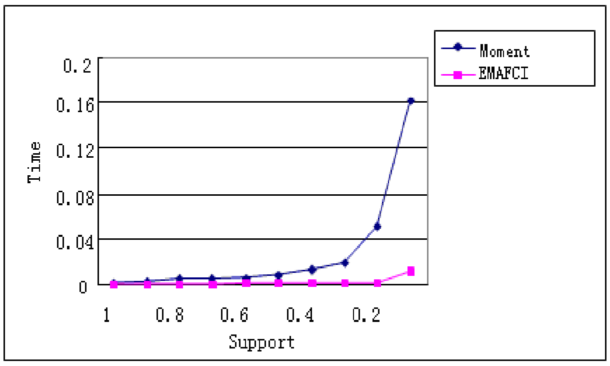

Figure 2. The average time of processing one window of Mushroom byEMAFCI compared with Moment

In Fig.2, the average time for processing a sliding window y the two algorithms are compared. From Fig.2, we can see that the average time for processing a sliding window by EMAFCI is much less than that of Moment especially when *support* is small. The time required for one window by EMAFCI is less than 0.04s, while the time of Moment increases very quickly and can go beyond 0.16s.

Conclusions

An algorithm of EMAFCI is proposed for mining the closed frequent itemsets from data stream. The algorithm is based on the sliding window model, and uses a BVTable where the transactions and itemsets are recorded by the column and row vectors respectively. The algorithm first builds the BVTable for the first sliding window. Frequent closed itemsets can be detected by pair-test operations on the binary numbers in the table. After building the first BVTable, the algorithm updates the BVTable for each sliding window. The frequent closed itemsets in the sliding window can be identified from the BVTable. Algorithms are also proposed to modify BVTable when adding and deleting a transaction.

The EMAFCI algorithm is implemented and compared its performance with Moment in terms of processing time and memory requirement. Our experimental results on both synthetic and real data show that EMAFCI is more effective with the guaranty of accuracy[12,13].

References

[1] J.W.Han, J.Pei, Y.W.Yin, R.Y.Mao. Mining frequent patterns without candidate generation: frequent-pattern tree approach, Data Mining and Knowledge Discovery, No.8, pp.53-87, 2004..

[2] K.T.Chuang, H.L.Chen, M.S.Chen. Feature-preserved sampling over streaming data. ACM Transactions on Knowledge discovery from data, Vol.2, No.4, Article 15, 2009

[3] Y.Y.Zhu, D.Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. Proceedings of the 28th International Conference on VLDB, Hong Kong, China, pp.358-369, 2002.

[4] H.F.Li, C.C.Ho, S.Y.Lee. Incremental updates of closed frequent itemsets over continuous data streams. Expert Systems with Applications, Vol.36, pp.2451-2458, 2009.

[5] Y.Chi, H.Wang, P.S.Yu, R.R.Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. Proceedings of the 2004 IEEE International Conference on Data Mining. Brighton, UK, pp.59-66,2004

[6] N.Jiang, L.Gruenwald. Research issues in data stream association rule mining. SIGMOD Record 35 (1), pp.14-19, 2006.

[7] Y.Chi, H.Wang, P.S.Yu, R.R.Muntz. Catch the moment: Maintaining closed frequent itemsets over a data stream sliding

window. Knowledge and Information Systems, 10 (3), pp.265-294, 2006

[8] F.J.Ao, J.Du, Y.J.Yan, B.H.Liu, K.D.Huang. An efficient algorithm for mining closed frequent itemsets in dataStreams. Proceedings of the IEEE 8th InternationalConference on Computer and Information Technology, pp.37-42, 2008.

[9] J.Y.Wang, J.W.Han, Y.Lu, P.Tzvetkov. TFP: An efficient algorithm for mining Top-K frequent closed itemsets. IEEE Transaction on knowledge and Engineering, Vol.17, No.5, pp.652-664, 2005

[10] J.Dong, M.Han. BitTableFI: An efficient mining frequent itemsets algorithm. Knowledge-Based Systems, Vol.20, pp.329-335, 2007.

[11] Dataset available at http://fimi.cs.helsinki.fi/.

[12] H.F.Li, H.Chen. Improve frequent closed itemsets mining over data stream with BitMap. Ninth ACIS international onference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp.399-404, 2008.

[13] L.Chen, L.J.Zou, L.Tu. Stream data classification using improved fisher discriminate analysis. Journal of Computers. Vol.4, No.3, pp.208-214, 2009.

[14] L. Bhuvanagiri, S. Ganguly, D. Kesh, C. Saha, “Simpler algorithm for estimating frequency moments of data streams”, Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp.708–713, 2006

[15] D. Coppersmith, R. Kumar, “An improved data stream algorithm for frequency moments”, Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pp.151–156, 2004.

[16] Y. Chi, H. Wang, P. S. Yu, R. R. Muntz, “Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window”, Fourth IEEE International Conference on Data Mining (ICDM’04), pp. 59–66, 2004.

[17] L. K. Lee, H. F. Ting, “Maintaining significant stream statistics over sliding windows”, Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp.724–732, 2006.

[18] 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.

[19] L. Golab , M. T. O¨ zsu, “Processing sliding window multi-joins in continuous queries over data streams”, Proceedings of the 29th international conference on Very large data bases, pp.500–511, 2003.

[20] G. S. Manku, R. Motwani, “Approximate frequency counts over data streams”. In Proceedings of the 28th International Conference on Very Large Data Bases, pp.356–357, 2002