Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 41 (2011) 5704-5709

ISSN: 2229-712X

Mining frequent itemsets over data streams using circular queues for efficient maintenance of sliding windows

Mala A[1], Ramesh Dhanaseelan F[2] and K.Pazhani Kumar[3]

[1]Department of Information Technology, KNSK College of Engineering, Nagercoil, Tamilnadu, India

[2]Department of Computer Applications, St. Xavier's Catholic College of Engineering, Nagercoil, Tamilnadu, India

[3]Department of Computer Science, S.T. Hindu College, Nagercoil, Tamilnadu, India.

ARTICLE INFO

Article history:
Received: 13 September 2011;
Received in revised form:
16 November 2011;
Accepted: 28 November 2011;

Keywords
Data Streams,
Frequent item set,
Mining Streams,
Stream Mining algorithms,
Transaction-sensitivesliding windows.

ABSTRACT

Mining frequent item sets from data streams is of great interest recently in many applications. Sliding windows are used in many applications to overcome an important problem with data streams i.e. unbound size. In this paper we propose an effective transaction based one-pass algorithm that uses a circular queue to implement the transaction-sensitive sliding window. The proposed algorithm, FIM_CQTransSWin has three phases representation of transaction, maintenance of the sliding window and generation of frequent item sets in the current sliding window. In the first phase, each transaction is read from the data stream and is converted into a decimal number based on the items present in this transaction. In the second phase, the decimal numbers representing the transactions are put in a circular queue with the front pointer indicating the oldest transaction and the rear pointer indicating the latest transaction. When the circular queue becomes full the oldest transaction is removed using dequeue operation and then the new transaction is appended at the rear end. The first and second phases are repeated indefinitely as long as the transactions keep arriving in the data stream. The third phase gets activated when the user requests for the frequent item sets. When the user request arrives, the frequent item sets in the current sliding window are generated using the candidate generation approach of the apriori algorithm and MASK operation.

Introduction

Data stream mining is a challenging problem in the data mining domain. Data stream are continuous, unbounded, come with high speed and exhibit concept drift also. These features of data streams make the mining of frequent patterns a difficult task to achieve. But this is a basic task required in a large number of applications like network monitoring and traffic management, sensor network monitoring, transaction analysis of e-commerce, web click stream monitoring and mining. The above mentioned applications generate data streams of large volume and also require frequent patterns to be mined from them.

Many challenges have to be overcome to mine such data streams. The inherent challenges for mining streaming data are discussed in detail in Babcock et al., 2002; Golab & Özsu, 2003; Jiang & Gruenwald, 2006. Some of the important issues are

•Each element can be examined at most once
•Memory usage should be restricted even though the stream is unbounded
•Data should be processed quickly and a real time response should be provided to the user
•The possibility of errors is high in such quick processing, but it should be maintained below a threshold

To deal with these issues a single-pass mining of frequent item sets over the data stream is needed. In most applications, the interest is on the current data only. So, sliding windows can be used to store the most recent transactions and mining is done on this window only.

In this paper, we propose an efficient one-pass algorithm for mining frequent item sets present in a transaction-sensitive sliding window implemented as a circular queue. The circular queue makes the removal of oldest transaction from the sliding

In this paper, we propose an efficient one-pass algorithm for mining frequent item sets present in a transaction-sensitive sliding window more efficient. The removal is accomplished by circularly incrementing the front pointer rather than shift operations used in previous algorithms like MFI-TransSW (Li & Lee, 2009).

Related Work

Efficient mining of frequent item sets over data streams have been studied and many contributions made recently. The research on mining data streams is broadly classified into three categories namely landmark-window based mining (Li et al., 2004; Li et al., 2005a; Li et al., 2005b; Manku & Montwani, 2002; Yu et al., 2006), damped-window based mining (Chang & Lee, 2003; Gianella et al., 2003) and sliding-window based mining (Chang & Lee, 2004; Chi et al., 2006; Lee et al., 2005).

Landmark-window model focuses on the values between a specific time called landmark and the current time. One of the landmark-window based algorithms is Sticky sampling and lossy counting which uses BTS (Manku & Montwani, 2002). Sticky sampling and lossy counting is a two-pass algorithm which uses a landmark-window and mines frequent patterns over data

Tele:
E-mail addresses: mala.elango@gmail.com, message_to_ramesh@yahoo.com, kpk_73@yahoo.co.in

streams. BTS is a three-module method which uses a lattice-based in-memory data structure and a landmark-window.

Damped-window model gives more weightage to recent transactions and less weightage to old transactions. EstDec algorithm (Chang & Lee, 2003) is a damped-window based algorithm for mining data streams where each transaction has a weight that keeps decreasing with its age.

Sliding-window model focuses on the most recent set of transactions. This model is further classified into transaction-sensitive and time-sensitive sliding window models. The basic unit of processing in a transaction-sensitive sliding-window model is a transaction. Transactions are added to the window as they arrive and when the window is full, sliding takes place by removing the oldest transaction and appending the latest transaction. MFI-TransSW (Li & Lee, 2009) uses a transaction-sensitive sliding-window model. In a time-sensitive sliding-window model, a time unit is the basic processing unit. All transactions belonging to a time unit is added to the window and when it becomes full, all transactions belonging to the oldest transaction unit are removed and all transactions belonging to the new time unit are added. MFI-TimeSW (Li & Lee, 2009) uses a time-sensitive sliding-window model. In this paper we propose a one-pass algorithm FIM_CQTransSWin to mine the set of frequent item sets within the current transaction-sensitive sliding window. This algorithm shall use a circular queue implementation of the sliding window to enable more efficient sliding operation.

Problem Definition

Let I= {i1, i2, i3 … im} be a set of items. A transaction Ti is a subset of these items with an associated transaction identifier that is unique to each transaction. A transaction data stream, TDS is a continuous sequence of transactions, T1, T2, T3 … TN. T1 is the oldest transaction and TN is the new transaction. A transaction sensitive sliding window is represented as TransSWin1, TransSWin2 … and so on. As the transactions arrive, they are put in the window one-by-one. Once the window is full, it slides forward for every new transaction that arrives.

A circular queue of size N is used to implement the sliding window. The current sliding window will be the circular queue between the front and rear pointers.

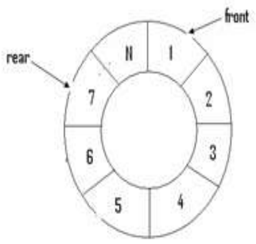

Fig. 3.1: Sliding Window of size N

When N transactions have arrived, the window (CQ) becomes full. The arrival of the next transaction causes a sliding, removal of the oldest transaction by means of dequeue operation and appending the new transaction at the rear end. 's' is a user-defined minimum support threshold percentage. Support(X) is the number of transactions in the sliding window that contain item set, X and it is said to be frequent if support(X) >= s.N/100. The problem is to find the frequent item sets in the current sliding window.

Example: Let the window size be 3 i.e. N=3 and support threshold s=60% and a portion of the data stream TDS= {< T_1, (acd) >, < T_2, (bce) >, < T_3, (abce) >, < T_4, (be) >}.

Therefore, frequency threshold = s.N/100=60x3/100 ≈ 2. And also this leads to two sliding windows, $TransSWin_1$=<T_1, T_2, T_3> and $TransSWin_2$=<T_2, T_3, T_4>

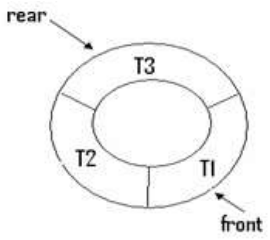

Fig. 3.2 Sliding Window, $TransSWin_1$

Frequent item sets in this are {a, b, c, e, ac, bc, be, ce, bce}

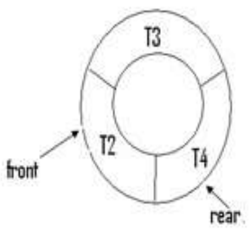

Fig. 3.3 Sliding Window, $TransSWin_2$

Frequent item sets in this are {b, c, bc, be, ce, bce}

FIM_CQTransSWin: Frequent Itemset mining within a Circular Queue based Transaction-sensitive Sliding Window

In this section, we describe our proposed single-pass mining algorithm, called FIM_CQTransSWin. It uses a decimal representation of each transaction, Compared with other bit-sequence based sliding window techniques, this algorithm shall use less memory and shall be more fast since removal of oldest transaction from the window requires only a pointer adjustment instead of the costly shift operation.

Representation of Transaction

Consider that any transaction in the data stream may consist of only items a, b, c, d, and e. Each item is represented by a unique number 0, 1, 2, 3, and 4. Each transaction is represented by a decimal number calculated from the representation of items present in it. (Ramraj & Venkatesan, 2009)

< T_1, (acd) >

Numeric equivalent of T_1

$= 2^0+2^2+2^3$

$= 1+4+8$

$=13$

Since a's equivalent is 0, c's equivalent is 2 and d's equivalent is 3.

< T_2, (bce) >

Numeric equivalent of T_2

$= 2^1+2^2+2^4$

$= 2+4+16$

$=22$

< T_3, (abce) >

Numeric equivalent of T_3

$= 2^0+2^1+2^2 +2^4$

$= 1+2+4+16$

$=23$

So before representing the transactions we should know the number of unique items possible in any transaction in our data stream. Then we should assign numbers to items accordingly in some specific order. We use lexicographic order.

Window Maintenance phase of FIM-CQTransSWin algorithm

The first part of this phase is window initialization where the transactions are put into the window at the rear end as they arrive after conversion into their corresponding decimal representation. This continues till the window becomes full. When the window is full the second part of this phase starts which is the sliding of the window. The sliding is performed in two steps: Removal of the oldest transaction by circularly incrementing the front pointer and then adding the new transaction (decimal representation) after incrementing the rear pointer also circularly.

In our example, we are in the first part of our window maintenance phase till the 4th transaction arrives. When the 4th transaction arrives the window is already full, so we move into the next part of this phase, the sliding of the window. First the front pointer is incremented circularly which causes the oldest transaction to be removed. Then the rear pointer is also incremented circularly and new transaction is added into the window.

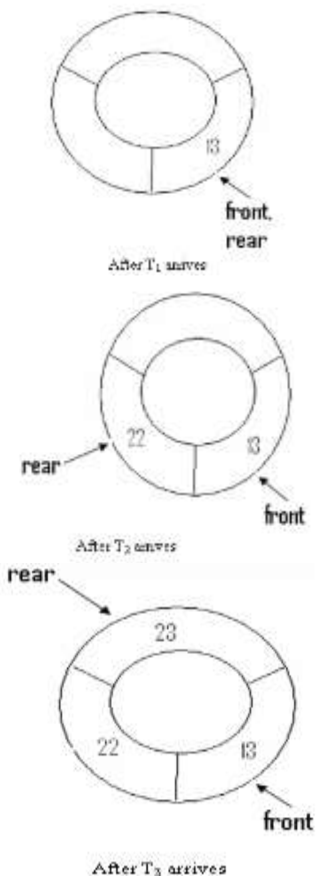

Fig. 4.1: Initialization of sliding window

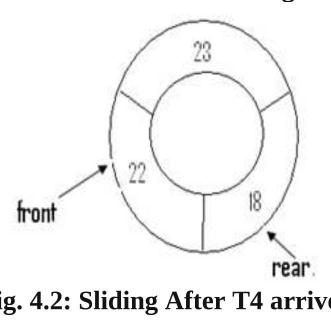

Fig. 4.2: Sliding After T4 arrives

Now the current window has the transactions T_2, T_3 and T_4 with representations 22, 23 and 18.

Frequent Item set generation phase of FIM_CQTransSWin

Only when requested by the user, this phase begins and the frequent item sets in the current sliding window are generated. This phase uses the Apriori algorithm to find the frequent itemsets using the candidate generation approach. The algorithm uses the MASK operation to find the support of the candidates generated to test whether they are frequent or not. This process continues till no new candidates are generated.

Example: The user requests for itemsets after the arrival of 4th transaction. At this time the current sliding window is TransWin2 and it has the numeric representations of the transactions T_2, T_3, T_4.

T_2=22=10110
T_3=23=10111
T_4=18=10010

<u>CI_1 and FI_1 formation</u>

Initial candidates are items a, b, c, d and e. find the frequency of each item using MASK operation as follows

```
T2 = 22 =  10110
Mask (a) = 00001
           -----------
              00000
          -----------
```

The result of masking operation is not Mask (a). So, we conclude ‘a’ is not present in T_2.

```
T3= 23    = 10111
Mask (a) = 00001
           -----------
              00001
          -----------
```

The result of masking operation is equal to Mask (a). So, we conclude ‘a’ is present in T_3.

```
T4= 18 =   10001
Mask (a) = 00010
           -----------
              00000
           -----------
```

The result of masking operation is not equal to Mask (a). So, we conclude ‘a’ is not present in T_4. So support (a) =1 in the $TransSWin_2$. Similarly we can calculate the support of each item.

<u>CI_2 and FI_2 formation</u>

Combinations to check are bc, be and ce. To find support of ‘bc’ first find the mask of ‘bc’ this is 00110.

```
T2= 22     =  10110
Mask (bc) =  00110
           ---------------
                 00110
          ---------------
```

The result of masking operation is equal to Mask (bc). So, we conclude ‘bc’ is present in T_2.

```
 T3= 23    = 10111
Mask (bc)= 00110
           --------------
               00110
          ---------------
```

The result of masking operation is equal to Mask (bc). So, we conclude ‘bc’ is present in T_3.

```
T4= 18      = 10010
Mask (bc) = 00110
           --------------
```

```
              00010
          --------------
```

The result of masking operation is not equal to Mask (bc). So, we conclude 'bc' is not present in T_4. So support (bc) =2 in the $TransSWin_2$. Similarly we can calculate the support of the other combinations 'be' and 'ce'.

CI_3 and FI_3 formation

Combinations to check are only 'bce'.

```
T2= 22       = 10110
Mask (bce) = 10110
          -----------------
               10110
          -----------------
```

The result of masking operation is equal to Mask (bce). So, we conclude 'bce' is present in T_2.

```
T3= 23       = 10111
Mask (bce) = 10110
          -----------------
                10110
          ----------------
```

The result of masking operation is equal to Mask (bce). So, we conclude 'bce' is present in T_3.

```
T4= 18       = 10010
Mask (bce) = 10110
          -----------------
               10010
          -----------------
```

The result of masking operation is not equal to Mask (bce). So, we conclude 'bce' is not present in T_4.

So support (bce) =2 in the $TransSWin_2$.

FIM_CQTransSWin Algorithm

```
Input:    TDS       a transaction data stream
          s         an user defined support
                    threshold percentage
N         size of circular used to
          Represent the sliding win.
Output: A table of frequent item sets and their support
Begin
1.Front = rear = -1//Cir. queue empty
2.Calculate freq. threshold= s.N/100
3.Repeat
4.For each transaction, $T_i$ in TDS do
5.Form the decimal rep. of $T_i$
6.If CQ full then
7.Remove oldest transaction,  $T_{i-N}$ from the front end and increment front circularly
8Add $T_i$ at rear end after incrementing  rear circularly
9.Until (interrupted)
/* When user requests for frequent
       Item sets the following phase starts */
10.$FI_1$ = {frequent 1- itemsets}
11.k=2;
12.While ($FI_{k-1} \neq$ NULL) do
13.Form candidate set $CI_k$ from
                 $FI_{k-1}$ using apriori property to
                 form candidates and MASK
                 operation to find its support
14.If $CI_k$ is NULL
15.break
16. Add the candidates in $CI_k$ to $FI_k$
                  if its support >= freq. threshold
end
```

This algorithm is used for finding the frequent item sets in the current transaction-sensitive sliding window maintained as a circular queue at the time that the user requests for the frequent item sets. Initially till the circular queue becomes full the initialization phase is done where each transaction gets converted into a decimal number based on the items present in it and it is stored in the circular queue representing the sliding window by incrementing the rear pointer (steps 4, 5 and 8). When the circular queue is full, slide the window to remove the oldest transaction and then insert the new transaction. (steps 4,5,6,7 and 8). When the user requests for frequent item sets, the current sliding window is considered and the frequency calculation is done using steps 10-16.

Performance Evaluation

In this section, we will describe the experimental evaluation of the proposed algorithm, FIM-CQTransSWin. The programs are implemented in Java NetBeans version 6.8 and executed on a system with the following configuration: Intel® Core™2 Duo CPU, E7500 @2.93 GHz, 1.98 GB RAM and 250GB Hard Disk running on Windows XP. For testing frequent itemset mining over the transaction-sensitive sliding windows implemented using circular queue concept, we generated synthetic data streams of size approximately 1K. To simulate data streams, the transactions were stored in a file and they were read one by one, converted into a number and stored in the sliding window. The parameters used in the experiments are shown in Table 5.1 below

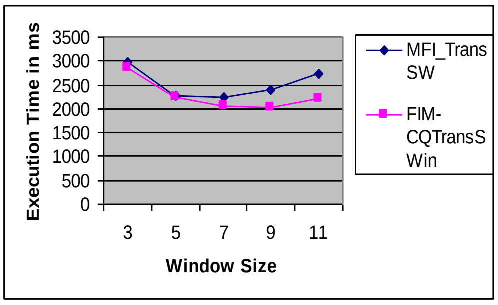

Figure 5.1 Execution Time Vs Window Size D=1024, I=5 and s=60%

Figure 5.1 gives the comparison of execution time in milliseconds as the window size is varied from three to twelve. The other parameters were no. of transactions=1024, minimum support threshold=60% and no. of items= 5. It clearly shows that execution time of MFI_TransSW increases faster than our proposed algorithm FIM-CQTransSWin.

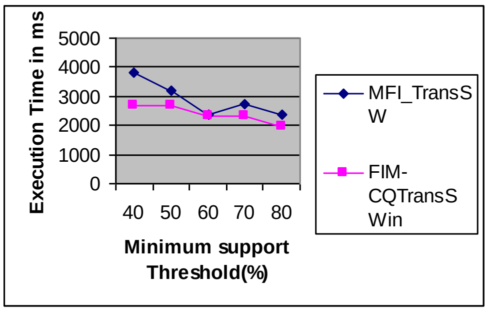

Figure 5.2 Execution Time Vs Threshold D=1024, W=5, and I=7

Figure 5.2 shows that execution time decreases as the threshold increases. The other parameters were no. of transactions=1024, window size=5, and no. of items= 7 and our proposed algorithm consistently outperforms the MFI_TransSW algorithm.

Figure 5.3 shows that execution time increases as the number of items in transactions increase. The other parameters were no. of transactions=1024, minimum support threshold=60% and window size= 5. In this case also our proposed algorithm consistently outperforms the MFI_TransSW algorithm.

Figure 5.4 shows that execution time decreases as the number of transactions decrease. The other parameters were window size =5, minimum support threshold=60% and no. of items= 5 and in this case also our proposed algorithm consistently outperforms the MFI_TransSW algorithm except when the number of transactions is 800 in which case both algorithms show the same execution time.

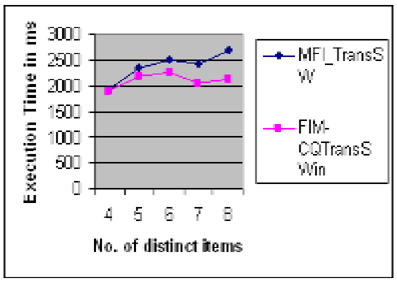

Figure 5.3 Execution Time Vs Number of Items D=1024, W=5, and s=60%

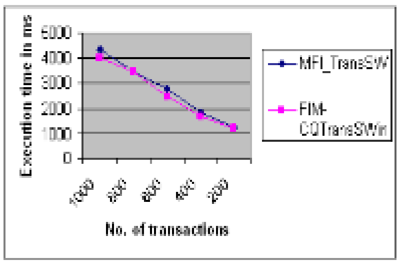

Figure 5.4 Execution Time Vs Number of Transactions W=5, I=5 and s=60%

Conclusion

In this paper, we propose an efficient single-pass algorithm, called FIM-CQTransSWin to mine the frequent item sets from the current transaction-sensitive sliding windows. The window is implemented using a circular queue that improves the efficiency of the sliding operation. The entire transaction is represented as a single number which helps to reduce the memory usage. Experiments show that the proposed algorithm runs faster than the existing algorithm taken for comaparison purpose, MFI_TransSW. In future our work is to extend the algorithm to handle time-sensitive transaction window and also to mine closed frequent item sets rather than all frequent item sets.

References

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., “Models and issues in data stream systems”. *In Proceedings of the PODS*, 2002, pp. 1–16.

Chang, J., Lee W., “Finding recent frequent itemsets adaptively over online data streams”. *In Proceedings of the ACM SIGKDD, 2003*, pp. 487–492.

Chang, J., & Lee W., “A sliding window method for finding recently frequent itemsets over online data streams”. *Journal of Information Science and Engineering*, 2004, 20(4), pp. 753–762.

Chi Y., Wang H., Yu P. S., & Muntz R. R., “Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window”. *Knowledge and Information Systems*, 2006, 10(3), pp. 265–294.

Giannella C., Han J., Pei J., Yan X., & Yu P. S., “Mining frequent patterns in data streams at multiple time granularities”. *In H. Kargupta, A. Joshi, K. Sivakumar, & Y. Yesha (Eds.), Data mining: Next generation challenges and future directions. AAAI/ MIT, 2003.*

Golab L., & Özsu M. T., “Issues in data stream management”, *SIGMOD Record*, 2003, 32(2), pp. 5–14.

Hua-Fu Li & Suh-Yin Lee “Mining frequent item sets over data streams using efficient window sliding techniques”. *Expert Systems with Applications, 36(2009*), pp. 1466–1477

Jiang N., & Gruenwald L., “Research issues in data stream association rule mining”, *SIGMOD Record, 2006*, 35(1).

Lee C.H., Lin C.R., & Chen M.S., “Sliding window filtering: An efficient method for incremental mining on a time-variant database”. *Information Systems*, 2005, 30, pp. 227–244.

Li H.F., Lee S.Y., Shan M.K., “An efficient algorithm for mining frequent itemsets over the entire history of data streams”, *In Proceedings of the IWKDDS, 2004.*

Li H.F., Lee S.Y., Shan M.K., “Online mining (recently) maximal frequent itemsets over data streams”, *In Proceedings of the IEEE RIDE, 2005a.*

Li H.F., Lee S.Y., & Shan M.K., “Online mining changes of items over continuous append-only and dynamic data streams”, *Journal of Universal Computer Science: Special Issue on Knowledge Discovery in Data Streams*, 2005b, 11(8), pp. 1411–1425.

Manku G. S., Motwani R., “Approximate frequency counts over data streams”, *In Proceedings of the VLDB, 2002*, pp. 346–357.

Ramraj E., Venkatesan N.: “Bit Stream Mask-Search Algorithm in Frequent Itemset Mining”; *European Journal of Scientific Research;* Vol.27 No.2, 2009, pp. 286-297

Yu J.X., Chong Z., Lu H., Zhang Z., & Zhou A., “A false negative approach to mining frequent itemsets from high speed transactional data streams”, *Information Sciences,2006*, 176(14), pp. 1986–2015.

Table 4.1 Item Equivalent Number

Item	*Equivalent Number*
a	*0*
b	*1*
c	*2*
d	*3*
e	*4*

Table 4.2 Candidate itemset C_1

Item	*Support*
a	***1***
b	***3***
c	***2***
d	***0***
e	***3***

Prune the non-frequent items ‘a’ and‘d’. Since frequency threshold calculated is 2, items with support less than 2 are not frequent.

Table 4.3 Frequent itemset F_1

Item	Support
b	3
c	2
e	3

Table 4.4 Candidate item set C_2

Itemset	Support
bc	2
be	3
ce	2

Support of all candidates is greater than or equal to 2. So FI_2 is same as CI_2

Table 4.5 Candidate itemset C_3

Itemset	Support
bce	2

Support of the only candidate is greater than or equal to 2. So FI_3 is same as CI_3.

No further combinations are possible. So finally the frequent item sets in $TransSWin_2$ are

Table 4.6 Frequent itemset in TransSWin_2

Itemset	Support
b	3
c	2
e	3
bc	2
be	3
ce	2
bce	2

Table 5.1 Parameters used in the experiments

Parameter	Description	Value
D	Number of Transactions	1000, 800, 600, 400, 200
I	Number of Distinct Items	4, 5, 6, 7, 8
s	Minimum Support Threshold	40%, 50%, 60%, 70%, 80%
N	Window Size	3, 5, 7, 9, 11