AN EFFICIENT ALGORITHM FOR MINING FREQUENT ITEMSETS OVER DATA STREAMS UNDER THE TIME-SENSITIVE SLIDING-WINDOW MODEL

V. Soujanya and S. Ramanaiah
Department of Computer Science and Engineering, Anna University, Chennai.
Software Engineer, L&T Infotech Limited, Chennai, India

ABSTRACT
Mining frequent itemsets has been widely studied over the last decade, mostly focuses on mining frequent itemsets from static databases. In many of the new applications, data flow through the internet or sensor networks which extend the mining techniques to a dynamic environment. The main challenges include a quick response to the continuous request, a compact summary of the data stream, and a mechanism that adapts to the limited resources. Here, we propose a time-sensitive sliding window model for mining frequent itemsets from data streams. Our approach consists of a storage structure that captures all possible frequent itemsets and a table providing approximate counts of the expired data items, whose size can be adjusted by the available storage space. This approach guarantees that both the execution time and the storage space remain small under various parameter settings. In time critical applications, a sliding window model is needed to remove old data. So, we adopt a model to mine the K most interesting itemsets, or to estimate the K most frequent itemsets of different sizes in a data stream. This method partitions the sliding window into buckets. This model provides frequency counts of the itemsets for the transactions in each bucket and supports to find out the top-k frequent itemsets. This method shows that it utilizes very small memory space.

KEYWORDS: *Data streams, frequent itemsets, sliding window.*

I. INTRODUCTION

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Consequently, the knowledge embedded in a data stream is more likely to be changed as time goes by. Data items continuously flow through the internet or sensor networks in applications like network monitoring and message dissemination. The characteristics of data streams are as follows:
1. **Continuity**: Data continuously arrives at destination at a high rate.
2. **Expiration**: Data can be read only once.
3. **Infinity**: The total amount of data is massively unbounded.
The above leads to the following requirements:
1. Time-sensitivity: A method which provides continuous flow of data streams based on the time sequence.
2. Approximation: As the past data cannot be stored, a method is required for providing the approximate answers with accuracy guarantees.
3. Adjustability: Owing to the unlimited amount of data, a mechanism that adapts itself to available resources is needed.
We focus on the problem of mining frequent itemsets over a data stream. In this problem, a data stream is formed by transactions arriving in series. The support count of an itemset means the number of transactions containing it and a frequent itemset means the one with a sufficient support count.

In static databases, mining frequent itemsets is possible through many methods such as Apriori [1], FP-growth [4], and Opportune Project [6]. Similarly in dynamic databases [3], [5], [2] some of the methods had been proposed. In these methods, all the frequent itemsets and their support counts derived from the original databases are retained. When transactions are added or expired, the support counts of the frequent itemsets contained in them are recomputed. By reusing the frequent itemsets and their support counts retained, the number of candidate itemsets generated during the mining process can be reduced. All these methods have to rescan the original database because non-frequent itemsets can be frequent after the database is updated. Therefore, they cannot work without scanning the entire database and cannot be applied to data streams.
Compared with the previous models such as Landmark model, Time-Fading model, estDec, FP-Pattern, sliding window model considered only the insertion of transactions, the sliding-window model further considers the removal of transactions. Therefore, if a method succeeds in the sliding-window model, it can easily be applied to the previous models. Moreover, all the previous works measured on only fixed number of transactions as the basic unit for mining. By contrast, it is natural for people to specify a time period as the basic unit. Therefore, we propose the time-sensitive sliding-window model, which regards a fixed time period as the basic unit for mining.

1.1 Time-sensitive Sliding-window (TS)

A basic block can be formed from a given time point p, a time period t_p, and a set of all the transactions arriving at $[p-t_p+1,p]$. A data stream is decomposed into a sequence of basic blocks, which are assigned with serial numbers starting at 1. Given a window with length $|l_w|$, we slide it over this sequence to see a set of overlapping sequences, where each sequence is called the time-sensitive sliding-window abbreviated as *TS*.
Let the basic block numbered i be denoted as B_i. The number of transactions in B_i is denoted as $|B_i|$, which is not fixed due to the variable data arrival rate. For each B_i, the TS that consists of the |lw| consecutive basic blocks from $B_{i-|lw|+1}$ to B_i is denoted as TS_i. Let the number of transactions in TS_i be denoted as Σ_i.

Definition 1.1 Frequent Itemsets in TS_i /B_i
The support count of an itemset in TS_i (B_i) is the number of transactions in $[B_{i-|lw|+1} \ldots B_i]$ (B_i) containing it. Given the support threshold θ, an itemset is frequent in TS_i (B_i) if its support count in TS_i (B_i) is not smaller than $\theta\times\Sigma_i$ ($\theta\times|B_i|$).Owing to the characteristics of data streams, it is not realistic to scan the past basic blocks again and again for mining frequent itemsets in each of the subsequent TS's. Such a scenario is illustrated in Fig.1

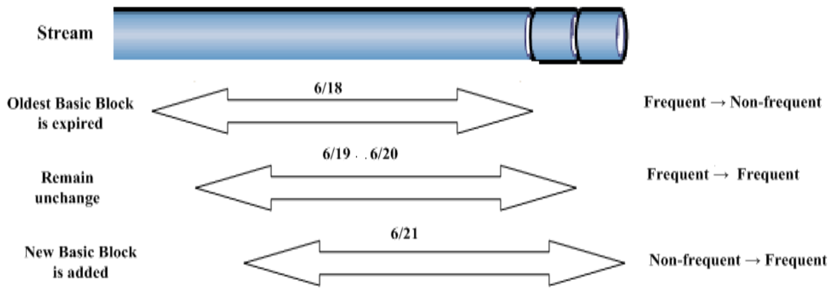

Figure 1: Time-sensitive sliding-window model

where the basic unit is one day and $|l_w|$ is 3. As the new basic block $B_{6/21}$ comes; the oldest basic block $B_{6/18}$ in $TS_{6/20}$ is expired. To find frequent itemsets in $TS_{6/21}$, we consider three kinds of itemsets from two sources, the frequent itemsets in $TS_{6/20}$ and the frequent ones in $B_{6/21}$, as follows:
• For each frequent itemset in $TS_{6/20}$, the support count is discounted if it occurs in$B_{6/18}$ and then updated by examining $B_{6/21}$. A mechanism to keep its support count in $B_{6/18}$ and a way to find their support counts in B6/21 are needed.
• A frequent itemset in $B_{6/21}$, which is not frequent in $TS_{6/20}$, can be frequent in $TS_{6/21}$. The methods for computing its support count in $TS_{6/20}$ and for mining frequent itemsets in $B_{6/21}$ are required.
• An itemset that is not frequent in both $TS_{6/20}$ and $B_{6/21}$ cannot be frequent in $TS_{6/21}$.

Since all the methods developed under other models accumulate the support count for each frequent itemset, no discounting information is provided. First of all, we devise a data structure named the discounting table (D_T) to retain the frequent itemsets with their support counts in the individual basic blocks of the current TS. Moreover, a data structure named the Potentially Frequent-itemset Pool (PFP) is used to keep the frequent itemsets in TS_i and the frequent ones in B_i. We include the itemsets that are frequent in B_i but not frequent in TS_{i-1} in PFP because they are possibly frequent in TS_i.

Definition 1.2 Potentially Frequent Itemset:

A frequent itemset in B_i that is not frequent in TS_{i-1} is called a potentially frequent itemset. Since its support count in TS_{i-1} is not recorded, we estimate that as the largest integer less than $\theta \times \Sigma_{i-1}$, i.e., the upper bound of its support count in $[B_{i-|lw|} \ldots B_{i-1}]$. This is called the potential count and also recorded in PFP. For the itemsets in PFP that are not potentially frequent, the potential counts are set to 0. In this way, each itemset in PFP is associated with the potential count and the accumulated count. Moreover, the sum of the two counts is regarded as the support count of this itemset in TS_i and used to determine whether it should be kept in PFP. When B_i arrives, three pieces of information are available for mining and discounting:

• D_T: Frequent itemsets with support counts in each of the basic blocks $B_{i-|lw|} \ldots B_{i-1}$.
• PFP: Frequent itemsets in TS_{i-1} or B_{i-1}.
• All the frequent itemsets discovered from B_i.

Mining frequent itemsets in TS_i consists of four steps. At first, the support counts of frequent itemsets in PFP are discounted according to D_T and then the frequent itemsets in $B_{i-|lw|}$ are removed from D_T. Second, the frequent itemsets in B_i are mined by using FP-growth [4] and added into PFP with their potential counts computed. Third, for each itemset that is in PFP but not frequent in B_i, we scan B_i to accumulate its support count and then delete it from PFP if it is not frequent in TS_i. At last, two alternatives to determine the frequent itemsets for output are provided:

1. Recall-oriented: All the itemsets kept in PFP are output. Since all the frequent itemsets in TS_i are in PFP, it guarantees that no false dismissal occurs.

2. Precision-oriented: We output only those itemsets whose accumulated counts in PFP satisfy $\theta \times \Sigma_i$. Because for the potentially frequent itemsets, these counts are lower bounds of their support counts, it guarantees that no false alarm occurs.

In addition to the mining and discounting methods, we further design the self- adjusting discounting table (SDT) that can automatically adjust its size when maintaining the discounting information. Given a limitation on the size of SDT, we devise a strategy to merge the information of more than one itemset kept in SDT. The main idea is to minimize the difference between the original support count of each itemset and its approximate count after merging. The most important finding is that the two guarantees described above still hold when D_T is deployed. The following are the contributions, corresponding to the three requirements mentioned before.

- **Time-sensitive sliding-window model**: We propose a model that is sensitive to time.
- **Mining and discounting methods**: An approach that continuously provides frequent itemsets over data streams. The accuracy guarantees of no false dismissal or no false alarm are provided.
- **Self-adjusting discounting table**: A mechanism that is self-adjusting under the memory limitation is presented. The accuracy guarantees still holds.

1.2 Mining top-k itemsets

With limited memory storage, it is natural to devise methods to store some kinds of statistics or summary of the data stream. However, in many applications, old data are less important or not relevant, compared with more recent data. There are two common approaches to deal with this issue. The first one is aging [12, 13], where each data is assigned a weight, with more weight for more recent data (e.g. exponential-decay model). Another approach is to use a sliding window[7, 9, 10, 8, 11], so that only the most recent W data elements in the data stream is considered, where W is the width of a sliding window. So, we adopt the second approach for mining top-k itemsets.

A major issue with mining frequent itemsets is that user has to define a support or frequency threshold s on the resulting itemsets, and without any guidance, this is typically a wild guess. In most previous work of data stream mining a major concern is to minimize the error of the false positive to a small fraction of s.

Therefore, it is of interest to replace the requirement of a frequency threshold to that of the simpler threshold on the amount of results. It is much easier for users to specify that say the 20 most frequent patterns should be returned. Some previous work assumed that such a threshold can be applied to itemsets of all sizes. However, there is a major pitfall with such an assumption. It is that it implies a uniform frequency threshold for itemsets of all sizes. It is obvious that small size itemsets have an intrinsic tendency to appear more often than large size itemsets. The result from this assumption is that smaller size itemsets can dominate and hide some interesting large size itemsets. The mining of closed patterns does not help much. For example, an interesting closed itemsets X of size 4 may have a frequency of 0.01, while many smaller size closed itemsets have frequencies above 0.l, and hence X cannot hope to reach the top K frequency. Therefore, some previous work has proposed to mine the K most frequent itemsets of size l, for each l that is within a range of sizes specified by user. We shall focus on this mining problem for data streams.

Let us call an itemset of size l an l-itemset. Our problem is about mining K l-itemsets with the greatest frequencies (supports) for each l up to a certain L. We shall tackle this problem for a data stream with a sliding window of size m (contains m transactions). In our approach, the sliding window is divided into n_B partitions, called buckets. Each bucket corresponds to a set of transactions and we maintain the statistics for the transactions in each bucket separately. The window is slided forward one bucket at a time. When the window is advanced, the oldest bucket is discarded and a newly generated bucket is appended to the sliding window. At the same time, the candidate top K interesting itemsets are adjusted.

The data stream is considered a sequence of equisized data buckets with s_B transactions each. The most recent n_B full buckets in the data stream are considered as the sliding window. Given two positive integers K and L. For each l, where $l \leq L$, and let the K-th highest frequency among all l-itemsets in the sliding window be f(l), find all l-itemsets with frequencies greater than or equal to f(l) in the sliding window. These are called the top K l-itemsets.

At any time, there will be a most recent bucket B, which may or may not be full. A bucket is full when it contains s_B transactions. When a transaction T arrives at the data stream, it will be inserted into B if it is not full; otherwise, a new bucket containing only T will be created and becomes the most recent bucket B. Let $m = n_B \times s_B$. Hence the size of the sliding window is m (number of transactions). The sliding window contains buckets B_1, B_2, ..., B_{nB}, in chronological order, where bucket B_{nB} represents the most recently created bucket.

In the process of mining top-k itemsets, assume the sizes of itemsets as l, $1 \leq l \leq L$. Without loss of generality let us consider size l itemsets, for a certain l. We are going to find the top K l-itemsets. Let l-itemset denote itemset of size l. Each bucket B_i stores a list of entries (e, f), where e is one of the top $K'_{i,l} \geq K$ l-itemsets and f is the frequency of e in the bucket. We use $f_{i,e}$ to denote the frequency of e in bucket Bi. We say that $f_{i,e}$ is recorded if e is among the top $K'_{i,l}$ itemsets. Therefore, each bucket stores information about the top $K'_{i,l}$ frequent itemsets.

Let $f_{i,min}$ be the frequency of the $K'_{i,l}$-th frequent l itemset in bucket B_i. For entry e in bucket B_i, $f_{i,e} \geq f_{i,min}$. We define min(e) and max(e). min(e) = $\sum B_i$ and $f_{i,e}$ is recorded $f_{i,e}$ and max(e) = $\sum B_i$ and $f_{i,e}$ is recorded $f_{i,e} + \sum B_i$ and $f_{i,e}$ is not recorded $(f_{i,min} - 1)$.

When we sum up all recorded frequencies $f_{i,e}$ of itemset e in different buckets B_i, this value should be the least possible frequency of itemset e. However, in some buckets B_i, there may be no recorded frequencies. The itemset e may appear in those buckets. To estimate the maximum possible frequency, we assume the maximum possible frequency for itemsets e with no recorded frequency, and this frequency is $f_{i,min} - 1$ for buckets Bi. Therefore min(e) is the minimum possible frequency of itemset e in the sliding window while max(e) is the maximum possible frequency of itemset e in the sliding window.

Let f_e be the frequency of e in the sliding window. Thus, min(e) ≤ fe ≤ max(e). We define f(1) = $\max_e$\{min(e)\}, which is the greatest value of min(e) among all e. We define $M_l = \min_{Bi}\{f_i,min\}$, which is the minimum value of f_i,min among all buckets.

II. DISCUSSION

2.1 Mining and Discounting

Figure 2 shows the system Architecture. The data stream is a series of transactions arriving continuously. Four parameters, the support threshold θ, the basic unit of time period for each basic block P, the length of TS $|l_w|$, and the output mode M, are given before the system starts. As time-sensitive sliding window states, a data stream is divided into blocks with different numbers of transactions according to P_B. The buffer continuously consumes transactions and pours them block-by-block into our system. After a basic block triggers these operations goes through our system, it will be discarded directly.

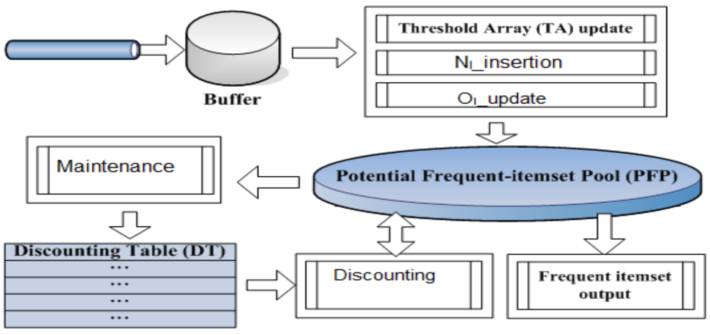

Figure 2: System Architecture

The basic blocks may have different numbers of transactions, we dynamically compute the support count threshold $\theta\times|B_i|$ for each basic block B_i and store it into an entry in the threshold array (TA), denoted as TA[i]. In our approach, only $|l_w|+1$ entries are maintained in TA. As B_i arrives, TA[j] keeps the support count threshold of B_{i-j} for $1\le j\le|l_w|+1$ and $i-j>0$. After B_i is processed, the last entry $TA[|l_w|+1]$ is ignored and the others are moved to the next positions, i.e., $TA[j]\rightarrow TA[j+1]$ for $1\le j\le|l_w|$. Finally, the support count threshold of B_i is put into TA [1].
In addition to TA_update, the arrival of a basic block also triggers the other operations in Fig 3.1, which are differently executed in three cases. For each case, requent_itemset_output is used to pick up the answers satisfying M from PFP. Fig 3.2 shows the main algorithm. First, as B_1 comes, two operations are executed one by one:
• **N_I_insertion**: An algorithm for mining frequent itemsets is applied to the transactions in the buffer. Each frequent itemset is inserted into PFP in the form of (ID, Items, Acount, Pcount), recording a unique identifier, the items in it, the accumulated count, and the potential count, respectively. Since an itemset is added into PFP, Acount accumulates its exact support counts in the subsequent basic blocks, while Pcount estimates the maximum possible sum of its support counts in the past basic blocks. For B_1, Pcount is set as 0.
• **Maintenance**: Each itemset in PFP is inserted into D_T in the form of (B_ID, ID, BCNT), recording the serial number of the current basic block, the identifier in PFP, and its support count in the current basic block, respectively. For B_1, B_ID is set as 1.
Input: Stream S, Parameters θ, P_B, $|l_w|$, M
Output: All the frequent itemsets satisfying M
Main algorithm

```
1: Let TA, PFP, and $D_T$ be empty  //$\forall j$, $TA[j] = 0$
2: While $B_i$ comes                  //from the buffer
3:   If  (i = 1)                      //$B_1$
// $N_I$_insertion
4:    Discover $F_i$ from $B_i$
5:    For each itemset $f$ in $F_i$
6:      If ($f \in$ PFP) Increase $f$.Acount
7:        Else  Insert $f$ into PFP   //Estimate f.Pcount
// Maintenance
8:      For each itemset $f$ in PFP
9:          Append $f$ to $D_T$
10:         Else If ($i \le |l_w|$)        //$B_2 \ldots B_{|lw|}$
// $N_I$_insertion
```

```
11:       Discover $F_i$ from $B_i$
12:       For each itemset $f$ in $F_i$
13:         If ($f \in PFP$) Increase  $f$.Acount
14:      Else Insert $f$ into PFP   //Estimate f.Pcount
// $O_I$_update
15:    For each itemset y in PFP but not in $F_i$
16:      Increase $y$.Acount by scanning  $B_i$ once
17:           If ($y$.Acount + $y$.Pcount < $\theta \times \Sigma_i$)
18:                Delete $y$ from PFP
 // Maintenance
19:       For each itemset $f$ in PFP
20:         Append $f$ to $D_T$
21:       Else                                  //$B_{|lw|+1...}$
// Discounting
22:   For each itemset y in PFP
23:       If ($y$.Pcount = 0)
24:         Find entry $h$ in DT where
          (y.ID = $h$.ID) and ($h$.B_ID = $i-|l_w|$)
25:       $y$.Acount = y.Acount – $h$.BCNT
26:     Else
27:      $y$.Pcount = $y$.Pcount – TA [$|l_w|+1$]
28:     If ($y$.Pcount < TA [$|l_w|$])  y.Pcount = 0
29:   For each entry $h$ in $D_T$
30:     If ($h$.B_ID = $i-|l_w|$)  Remove $h$ from $D_T$
// $N_I$_insertion
31:  Discover $F_i$ from $B_i$
32:   For each itemset $f$ in $F_i$
33:       If ($f \in$ PFP) Increase $f$.Acount
34:        Else   Insert $f$ into PFP //Estimate f.Pcount
// $O_I$_update
35:       For each itemset y in PFP but not in $F_i$
36:            Increase $y$.Acount by scanning $B_i$ once
37:        If ($y$.Acount + $y$.Pcount < $\theta \times \Sigma_i$)
38:            Delete $y$ from PFP
//Maintenance
39:   For each itemset $f$ in PFP
40:       Append $f$ to $D_T$
41: TA_updat
  //Frequent_itemset_output
42: If ($M = NFD$)          //No-false-dismissal mode
43: For each itemset $f$ in PFP
44:       O = O s+ {f}
45:        Else                                  //$M=NFA$
46:       For each itemset $f$ in PFP
47:        If ($f$.Acount $\geq \theta \times \Sigma i$) $O = O + \{f\}$
```

When B_i arrives, where $1<i\leq|l_w|$, three operations are executed one by one:

N_I_insertion: In this case, we further check every frequent itemset discovered in B_i to see whether it has been kept by PFP. If it is, we increase itsAcount. Otherwise, we create a new entry in PFP and estimate its Pcount as the largest integer that is less than $\theta \times \Sigma_{i-1}$.

O_I_update: For each itemset that is in PFP but not frequent in B_i, we compute its support count in B_i by scanning the buffer to update its Acount. After that, an itemset in PFP is deleted if its sum of Acount and Pcount is less than $\theta \times \Sigma_i$.

Maintenance: This operation is the same as described previously except that B_ID is set as i. At last, when B_i arrives, where $i>|l_w|$, the window slides and 4 operations are executed one by one. Before that, an extra operation is executed.

Discounting: Since the transactions in $B_{i-|lw|}$ will be expired, the support counts of the itemsets kept by PFP are discounted accordingly. We classify the itemsets into two groups by Pcount. If it is nonzero, we repeatedly subtract the support count thresholds of the expired basic blocks from Pcount and finally set Pcount to 0. If Pcount is already 0, we subtract BCNT of the corresponding entry in D_T from Acount. Finally, each entry in D_T where B_ID = i−|l_w| is removed.

2.2 Self-Adjusting Discounting Table

In this section, we refine Maintenance to address the issue of the limited memory space. Among the data structures maintained for mining and discounting in our approach, D_T often consumes most of the memory space. When the limit is reached, an efficient way to reduce the D_T size without losing too much accuracy is required. A straightforward way is to merge the entries in D_T as needed. The main challenge is how to quickly select the entries for merging such that the resultant D_T still performs well in discounting. In the following, a naive solution called the naive adjustment is introduced first and then our proposed method named the selective adjustment is presented.

2.2.1 Naive Adjustment

The naive adjustment is fast but provides inaccurate information for discounting when too many entries are merged together. When discounting itemsets some unexpected errors occur. We call the sum of these errors as the merging loss. Next, we will introduce our method that uses the merging loss to select the entries for merging.

2.2.2 Selective adjustment

In this method, each entry D_{Tk} is in the new form of (B_ID, ID, BCNT, AVG, NUM, Loss). *D*$_{Tk}$.AVG keeps the average of support counts for all the itemsets merged into D_{Tk}, *D_{Tk}.NUM* is the number of itemsets in D_{Tk}, while *D_{Tk}.Loss* records the merging loss of merging D_{Tk} with D_{Tk-1}. The main idea of our method is to select the entry with the smallest merging loss, called the victim, and merge it into the entry above it. D_{T1}.Loss and D_{Tk}.Loss are set ∞ to avoid being the victim, ∀k, D_{Tk}.B_ID≠D_{Tk-1}.B_ID. Since the merging loss of an entry depends on the output mode *M*, we formulate it as follows:

Definition 3.1 Merging loss

or k>1 and DTk.B_ID=DTk-1.B_ID, DTk.Loss under the NFD mode is computed as follows:

(D_T.NUM X D_{TK}.AVG = D_{TK-1}.NUM X D_{TK-1}.AVG)- min{ D_{Tk}.Bcnt, D_{Tk-1}.Bcnt} x (D_{Tk}.NUM + D_{TK-1}.NUM) – (2.1)

DTk.Loss under the NFA mode is computed as follows:

max{ D_{Tk}.Bcnt, D_{Tk-1}.Bcnt} x (D_{Tk}.NUM + D_{TK-1}.NUM)-(D_T.NUM X D_{TK}.AVG = D_{TK-1}.NUM X D_{TK-1}.AVG) –(2.2)

The following illustrates the Maintenance with selective adjustment.

Input: PFP, D_T, D_T_size, D_T_limit

Output: updated D_T

Maintenance with selective adjustment

```
1. For each itemset f in PFP
2.    If (D_T_size = D_T_limit)
3.       Scan D_T once to select the victim        //D_Tk
3.    D_Tk-1.ID = D_Tk-1.ID ∪ D_Tk.ID
4.    If (M=NFD)
5.    D_Tk-1.BCNT=min{D_Tk-1.BCNT, D_Tk.BCNT}
6.    Else                                     //M=NFA
7.    D_Tk-1.BCNT=max{D_Tk-1.BCNT, D_Tk.BCNT}
8.     D_Tk-1.NUM = D_Tk-1.NUM + D_Tk.NUM
9.    Compute D_Tk-1.AVG
10.     If (D_Tk-1.Loss≠∞) Recalculate D_Tk-1.Loss
11.        Remove D_Tk from D_T; D_T_size--
12.      If(D_Tk+1.Loss≠∞)   Recalculate D_Tk+1.Loss
13.        Append f to D_T; D_T_size++
```

At the beginning of selective adjustment, we scan D_T once to find the victim. Suppose that D_{Tk} is the victim and will be merged into D_{Tk-1}. For the new D_{Tk-1}, the assignment of ID and BCNT follows the

same way in the naïve adjustment. Moreover, NUM is assigned with the sum of D_{Tk-1}.NUM and D_{Tk}.NUM, while AVG is computed as follows:
D_{TK}.AVG x D_{TK}.NUM + D_{TK-1}.AVG x D_{TK-1}.NUM -(2.3)D_{TK}.NUM + D_{TK-1}.NUM
Based on the new D_{Tk-1}.BCNT, D_{Tk-1}.AVG, and D_{Tk-1}.NUM, D_{Tk-1}.Loss can be computed. Note that if the old D_{Tk-1}.Loss has been set ∞, it is unchanged. After the merging, the merging loss of the entry below the victim, i.e., D_{Tk+1}.Loss, is also updated as Step 1.1.9 indicates.

Example 2.1

Consider Table 1, D_T_limit=4, and M=NFD. Table 2(a) shows the D_T as itemset *A* is added. Since it is the first entry, its merging loss is set ∞. As itemset B is added, we compute its merging loss by Formula (2) and the result is shown in Table 4(b). In the same way, we add itemsets *C* and *F* to form the DT in Table 2(c). Since D_T is full now, the selective adjustment is executed before the addition of AF. Specifically, the entry (1, 3, 13, 13, 1, 1) is selected as the victim and merged with (1, 1, 12, 12, 1, ∞). The result after merging forms the first entry in Table 2(d), where D_{T1}.Loss is ∞ and D_{T1}.AVG (=12.5) is computed by Formula (4). Notice that D_{T2}.Loss is changed from 11 to 21 as Step 1.1.9 indicates. Finally, we add itemset G in a similar way to obtain the final result in Table 2(e).

Table 1: The itemsets to be inserted

B_ID	ID	ITEMSET	BCNT
1	1	A	12
1	3	B	13
1	4	C	2
1	5	F	10
1	6	AF	10
1	8	G	8

Table 2: The process of the selective adjustment

B_ID	ID	BCNT	AVG	NUM	LOSS
1	1	12	12	1	∞

(a)

B_ID	ID	BCNT	AVG	NUM	LOSS
1	1	12	12	1	∞
1	3	13	13	1	1

(b)

B_ID	ID	BCNT	AVG	NUM	LOSS
1	1	12	12	1	∞
1	3	13	13	1	1
1	4	2	2	1	11
1	5	10	10	1	8

(c)

B_ID	ID	BCNT	AVG	NUM	LOSS
1	1-3	12	12-5	2	∞
1	4	2	2	1	21
1	5	10	10	1	8
1	6	10	10	1	0

(d)

B_ID	ID	BCNT	AVG	NUM	LOSS
1	1-3	12	12-5	2	∞
1	4	2	2	1	21
1	5-6	10	10	2	16
1	8	8	8	1	4

(e)

III. CONCLUSION

Mining data streams is an interesting and challenging research field. The characteristics and requirements of data streams are specified. An efficient algorithm for mining frequent itemsets over data streams under the time-sensitive sliding-window model was introduced. If the memory is limited the proposed two SDT strategies performs well. Mining top-k itemsets over data streams was discussed clearly. In this way of mining over the data streams the memory usage and the execution time are many times smaller compared with a naive approach.

REFERENCE

[1]. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. of VLDB Conf.,pp. 487-499, 1994
[2]. C.H. Chang and S.H. Yang, "Enhancing SWF for Incremental Association Mining by Itemset Maintenance," Proc. of Pacific-Asia Conf. on Knowledge Discovery and Data Mining, 2003.
[3]. D. Cheung, J. Han, V. Ng, and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," Proc. of ICDE Conf., 1996.
[4]. J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," Data Mining and Knowledge Discovery, 8(1): 53-87, Kluwer Academic Publishers, 2004.
[5]. J. Liu, Y. Pan, K. Wang, and J. Han, "Mining Frequent Item Sets by Opportunistic Projection," Proc. of ACM SIGKDD Conf., 2002.
[6]. C. Lee, C. Lin, and M. Chen, "Sliding-window Filtering: An Efficient Algorithm for Incremental Mining," Proc. of ACM CIKM Conf., 2001.
[7]. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In PODS, 2002.
[8]. B. Babcock, M. Datar, R.Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In SIGMOD, 2003.
[9]. C.-R. Lin C.-H. Lee and M.-S. Chen. Sliding-window filtering: An efficient algorithm for incremental mining. In Intl. Conf. on Information and Knowledge Management, 2001.
[10]. M. Datar, A. Gionis, P. Indyk, and R.Motwani. "maintaining stream statistics over sliding windows". In SIAM Journal on Computing, 2002.
[11]. C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. In Next Generation Data Mining, 2003.
[12]. Gilbert, Y. Kotidis, and S. Muthukrishnan. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In VLDB, 2001.
[13]. J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In SIGKDD, 2003.

BIOGRAPHY

Vemulamada Soujanya received the Bachelor's of engineering degree in Information Technology from Jawaharlal Nehru Technological University, Hyderabad, Andhra Pradesh, India in 2000 and 2004. She received masters of engineering degree in computer science and engineering from Jawaharlal Nehru Technological University, Anantapur, Andhra Pradesh, India in 2009 and 2011 respectively. Currently, she works in Madha Engineering College, Chennai.

Suram Ramana received the Master of Science in computer Science from Nagarjuna University, Guntur. He received Master's of engineering degree in computer science and engineering from Jawaharlal Nehru Technological University, Anantapur, Andhra Pradesh, India in 2009 and 2011 respectively. Currently, he works in L&T infotech Limited, Chennai.