TDMCS: An Efficient Method for Mining Closed Frequent Patterns over Data Streams Based on Time Decay Model

Meng Han, Jian Ding, and Juan Li
School of Computer Science and Engineering, North Minzu University, China

Abstract: *In some data stream applications, the information embedded in the data arriving in the new recent time period is important than historical transactions. Because data stream is changing over time, concept drift problem may appear in data stream mining. Frequent pattern mining methods always generate useless and redundant patterns. In order to obtain the result set of lossless compression, closed pattern is needed. A novel method for efficiently mining closed frequent patterns on data stream is proposed in this paper. The main works includes: distinguished importance of recent transactions from historical transactions based on time decay model and sliding window model; designed the frame minimum support count-maximal support error rate-decay factor (θ-ε-f) to avoid concept drift; used closure operator to improve the efficiency of algorithm; design a novel way to set decay factor: average-decay-factor $f_{average}$ in order to balance the high recall and high precision of algorithm. The performance of proposed method is evaluated via experiments, and the results show that the proposed method is efficient and steady-state. It applies to mine data streams with high density and long patterns. It is suitable for different size sliding windows, and it is also superior to other analogous algorithms.*

Keywords: *data stream mining, frequent pattern mining, closed pattern mining, time decay model, sliding window, concept drift.*

Received January 15, 2015; accepted August 12, 2015

1. Introduction[1]

Data stream as a new data model is widely used in many applications. Data stream which is different from traditional database is time ordered, rapidly changing, massive and unlimited. Searching for frequent patterns in a continuous data stream has became important and challenging.

In recent years, some algorithms for mining frequent patterns or itemsets on data streams have been proposed. Algorithms such as sticky sampling [15], lossy counting [15], XSM [1] and FDPM [26] mine frequent patterns which meet maximal support error rate and minimum support count. These methods do not distinguish between recent and historical transactions and do not consider the importance of recent transactions. In addition, these methods for mining complete result sets will produce a lot of useless patterns. For reducing the number of patterns, concise pattern set should be mined, mainly including: maximal frequent patterns, closed frequent patterns, top-*k* frequent patterns or a combination of them and so on. Algorithms max-FISM [6] and GUIDE [19] discover recent maximal frequent patterns based on sliding windows. WMFP-SW [10] mines weighted maximal frequent patterns based on sliding windows. Algorithms moment [5], newmoment [11], clostream [24], Stream_FCI [20], TMoment [17], IncMine [4] and CloStream*[25] discover closed frequent patterns based on sliding windows. TOPSIL- Miner [23] uses landmark windows to mine top-*k* frequent patterns. Methods Top-*k* Lossy Counting [22], MSWTP [2] and Top-*k* Miner [18] discover top-*k* frequent patterns based on sliding windows. FCI_max [21] mines closed top-*k* frequent patterns based on sliding windows and so on. The drawbacks of above algorithms are that:

1. Using only the minimum support threshold for frequent patterns mining and unprocessed concept drift problem of data streams.
2. Although window model are used in these methods, the weights of transactions in window are same.

As can be seen from the above algorithms, mining frequent patterns on data streams usually based on window model, especially the sliding window. The reason is that recent transactions normally contain more information than historical ones. Besides sliding window model, Time Decay Model (TDM) [3, 5, 8, 9, 12, 13, 14, 19] is also used to process recent transactions.

TDM-based methods to mine frequent patterns on data stream emphasize that the importance of recent and historical transactions should be distinguished in the window. Recent years, the ways to set decay factor in TDM usually divide into two categories. The first

[1]This paper is supported by National Nature Science Foundation of China (61563001) and Nature Science Foundation of Ningxia (No. NZ17115) and North Minzu University (2014XYZ13).

one set decay factor to random value in the range of (0, 1) [9, 14, 19]. Such ways lead to the instability of the mining results because of the random values of decay factor. The second method assumes that algorithm meets 100% Recall or 100% Precision to get the upper and lower bounds of decay factors [3, 12]. Then set decay factor to the upper bound or lower bound or random value between them. The problems of these two ways to set decay factor are that they can get high recall or high precision of algorithm, while get the low corresponding precision or recall of algorithm. Or because of the uncertainty of the decay factor value, the pattern results of algorithm are instability.

In order to avoid concept drift, distinguish recent transactions from historical ones, discover compact pattern result set efficiently, and apply to mine high dense transactions and long patterns, a novel algorithm is proposed in this paper. Mainly works and innovations are that:

1. Design a novel way to set decay factor *f*. Existed methods set *f* to boundary value of lower bound or high bound by assuming 100% Recall and 100% Precision [3, 12, 26], or to a random value in range of (0, 1) [13, 16]. The former will lead to corresponding algorithm low Precision or low Recall. And the later will make unstable performance of algorithm. In order to balance Recall and Precision of algorithm, proposed an average way to set decay factor in this paper.
2. Propose a three layers frame: minimum support-maximum support error-decay factor to solve the concept drift problem and avoid loss of possible frequent patterns.
3. Propose a novel algorithm to mine closed frequent patterns on data streams based on time decay model and sliding window model. It can get lossless compression result set. Time decay model [2, 11, 12, 15] is used to further emphasize the importance of recent transaction and reduce the importance of historical one. By comparisons of precisions of novel algorithm and existed algorithms, the novel algorithm can get more accurate pattern result.

The rest of this paper is organized as follows. Section 2 presents background knowledge; mainly about closure operator and time decal model. The efficient novel algorithm based on time decay model to discover closed patterns is introduced in section 3. Section 4 describes the experiments and explains the experimental results. Section 5 concludes this work.

2. Preliminaries

A data stream *DS*=<*T*1, *T*2, … , *Ti*, …> is a continuous and unbounded sequence of transactions in a timely order, where *Ti* (*i*=1, 2, …) is the *ith* transaction. Each transaction contains a unique transaction identifier t_{id}, as shown in the first column of Table 1. The support count of frequent pattern *P*, denoted as *freq*(*P*, *N*)[5], is the number of transactions in existed *N* transactions in which *P* occurs.

- *Define 1.* (Frequent Pattern [12]) Let *N* be the sliding window size, and θ ($\theta \in (0,1]$) be the minimum support. If itemset *P* meets $freq(P, N) \geq \theta \times N$, *P* is a frequent pattern.
- *Define 2.* (Half-Frequent Pattern, Non-Frequent Pattern [12]) Let *N* be the sliding window size, $\theta(\theta \in (0,1])$ be the minimum support and ε ($\varepsilon \in (0, \theta)$) be the maximal support error. If itemset *P* meets $\theta \times N \geq freq(P, N) \geq \varepsilon \times \theta \times N$, *P* is a half-frequent pattern. Else if $freq(P, N) < \varepsilon \times \theta \times N$, *P* is a non-frequent pattern.

Table 1. Transaction data stream.

TID	Transaction
t_1	1 3 4
t_2	2 3 5
t_3	1 2 3 5
t_4	2 3 4 5

Data stream changes in real time and the infrequent patterns over time may become frequent patterns. That is concept drift. Therefore, in order to reduce the number of missing possible patterns, frequent patterns and half-frequent patterns need to be maintained during mining process. In addition, in order to reduce the cost of maintaining patterns, non-frequent patterns need to be lost. By this way, the possible error of missing patterns is not greater than ε [3, 12]. Therefore, using θ-ε framework can solve the problem of concept drift.

A heavy problem of mining frequent pattern from data stream is generated a large number of useless patterns. Therefore, mining useful and compressed patterns are needed. Discovering closed frequent pattern is a common method, which is lossless compressed and contains all the information of the complete result. Meanwhile, in order to improve the efficiency to discover closed patterns, closure operator [24, 25] is used in this paper. The performance of the algorithm with closure operator is better than classic closed pattern mining algorithms such as Moment [5], CFI-Stream [9] and NewMoment [11]. Take closure operator into account, the concepts of closed patterns are shown in definitions 3 to 5.

- *Define 3.* (Closure Operator [24, 25]) Let *T* be the subsets of all that transactions in *D*, denotes as $T \subseteq D$. Let *Y* be the subset of all items *I* ($Y \subseteq I$) which appears in *D*. Concept of closed itemset is based on the following two functions *h* and *g*:

$$h(T) = \{i \in I \mid \forall t \in T, i \in t\} \quad (1)$$

$$g(Y) = \{t \in D \mid \forall i \in D, i \in t\} \quad (2)$$

Function *h* takes *T* as input and returns an itemset included in all transactions belonging to *T*. Function *g* takes an itemset Y as an input and returns a set of transactions including Y. A function

$$C = h \circ g = h(g)$$

is called Closure Operator.

- *Define 4.* (Closed Itemset [25]) An itemset P is called a closed itemset if and only if it satisfy Formula 3. Otherwise, P is non-closed. The $C(P)$ is called the closure of P.

$$C(P) = h(g(P)) = P \tag{3}$$

- *Define 5.* (Closed Frequent/Half-Frequent Pattern) If itemset $P=C(P)$ and its support is no less than minimum support, P is called a closed frequent pattern. If itemset $P=C(P)$ and its support is no less than maximal support error, P is called a closed half-frequent pattern. Otherwise, P is a non-frequent pattern.

Due to the continuous and infinite, knowledge contained in data stream may change with the passage of time. Under normal circumstances, the value of recent transaction is more important than historical one. Therefore, it is necessary to increase the weight of recent transaction. The TDM is developed to gradually decay the occurrence count of itemset contained in the transaction [2, 11]. Let the decay ratio of support count in the unit time to decay factor f ($f\in(0,1]$). When T_n arrives, support count of frequent pattern P is denoted as $freq_d(P, T_n)$. Each time a new transaction arrives, $freq_d(P, T_n)$ is multiplied by a decay factor f. When the mth transaction T_m arrives, r is 1 if it contains P, otherwise r=0. The $freq_d(P, T_m)$ based on decay factor is shown in Formulas 4 and 5.

$$\begin{aligned} &freq_d(P,T_m) = r, \quad if \quad m=1 \\ &freq_d(P,T_m) = freq_d(P,T_{m-1}) * f + r, \quad if \quad m \geq 2 \end{aligned} \tag{4}$$

$$\begin{aligned} &r = 1, \quad if \quad P \subseteq T_m \\ &r = 0, \quad otherwise \end{aligned} \tag{5}$$

3. Algorithm TDMCS

In this section, data structures and the new way to define decay factor f are introduced, and the proposed algorithm (TDMCS TDM-Based Closed Frequent Pattern Mining on Data Stream) is introduced in detail which mines frequent closed patterns based on f-θ-ε framework.

Three data structures are used in algorithm TDMCS, including: ClosedTable [24], CidList [24] and New-TransactionTable / OldTransactionTable. ClosedTable which is used to maintain the information of closed itemsets consists of three fields: Cid, CP and SCP. Each closed itemset CP is assigned to a unique closed identifier Cid, and its support count is denoted as SCP. CidList maintains each item in data stream and the corresponding Cid points to pattern in ClosedTable. NewTransactionTable is used to maintain the information of a new transaction T_{new}. It consists of two fields: TempItem and Cid. TempItem contains the information of itemsets which satisfy $\{T_i \cap T_{new}, T_i \in$ ClosdeTable$\}$. The T_i is the *ith* transaction in data stream and T_{new} is the new transaction. Structure of OldTransactionTable is same as NewTransaction-Table. It is used to maintain the information of old transaction T_{old}.

The core issue of removing old transactions from sliding window is how to effectively prune the existing data structures. Existing methods are often pruning step by step which are inefficient. A sliding step M is used in this paper. Pruning data structure after the sliding window move M transactions, that is, pruning when transactions T_{sw+i*M} (sw is the size of sliding window, i=1, 2, ...) are arrived.

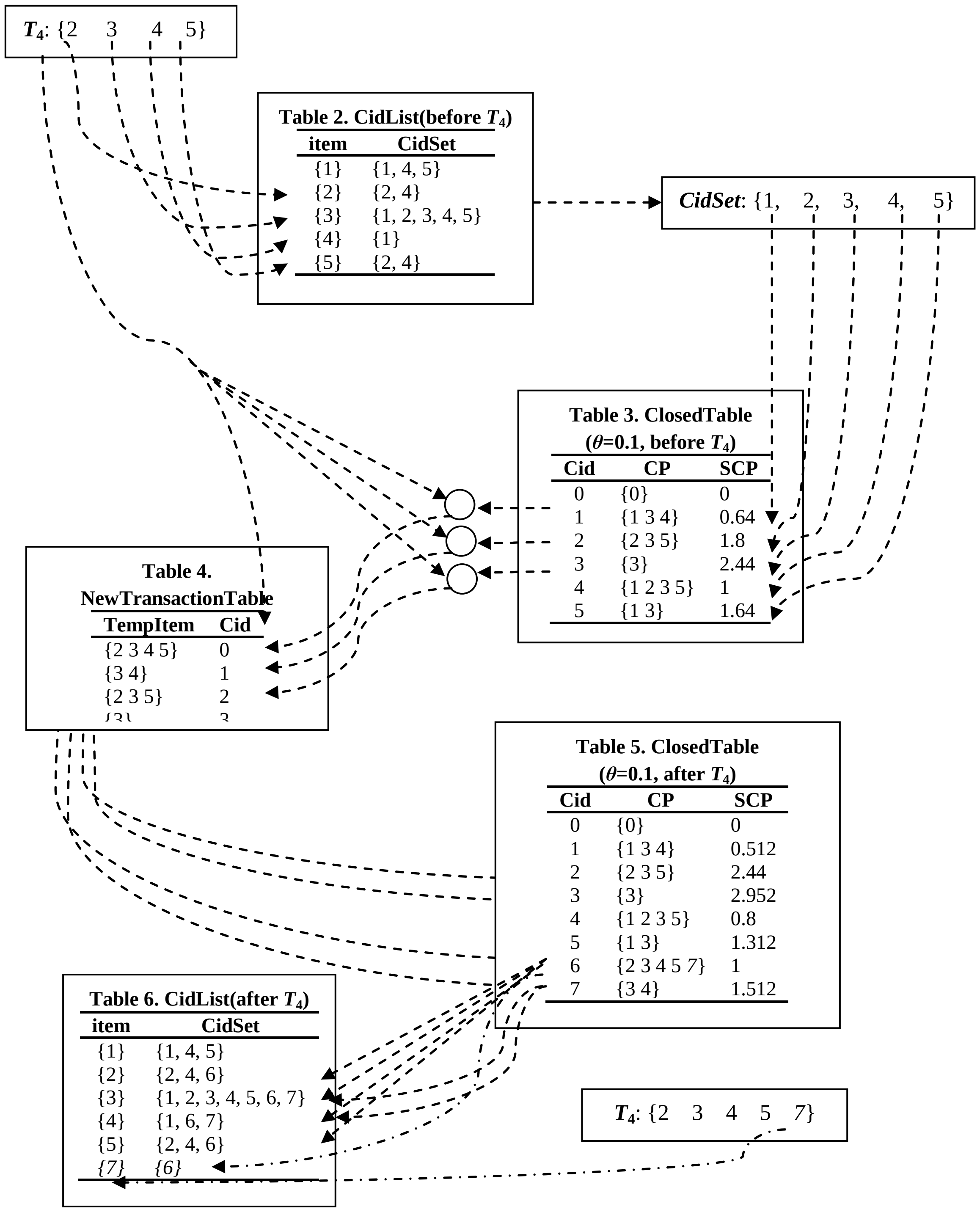
In order to distinguish the weights of the historical transactions and the recent transactions, improve the accuracy of result set, and avoiding the missing of possible frequent patterns, a novel algorithm TDMCS is proposed in this paper. TDMCS mines closed frequent patterns on data streams based on frame θ-ε-f (minimum support-maximal support error-decay factor). This algorithm uses data structures *ClosedTable*, *CidList*, *NewTransactionTable and OldTransactionTable* to maintain frequent itemsets information. It uses TDM to estimate the support count of pattern, and it maintains the frequent and half-frequent closed frequent itemsets which satisfy frame θ-ε. The description of algorithm TDMCS is shown as Algorithm 1. The main idea is processing the information of new transaction T_{new} at first. Secondly, if the number of processed transactions exceeds the size of sliding window, delete the information of old transaction T_{old}. If the processing steps of the transactions meet the pruning step M, do pruning operation. In order to increase the efficiency of the algorithm, it only adds delete flags (*DeleteFlag*) when processing old transactions and does the actual delete operations when pruning.

Algorithm 1: TDMCS()

```
Mining closed frequent patterns on data streams
1  For Each Transaction $T_{new}$ In S Do
2  Call TDMCSADD($T_{new}$);
3  If NUM>N Then TDMCSREMOVE($T_{old}$);
4  If NUM%M==0 Then Call PRUNNING();
5  End For
```

Specifically, there are three methods in algorithm TDMCS. Method *TDMCSADD*(T_{new}) is used to process the new transactions, method *TDMCS-REMOVE*(T_{old}) to process old transactions and method *PRUNING*() to process pruning.

When new transaction T_{new} arrived, TDMCSADD (T_{new}) is described as Algorithm 2. For example, if new transaction is $T4$ as shown in Table 1, this algorithm processes it in four steps. First, generate CidSet associated with $T4$ to discover the intersections between *T*4 and existed frequent itemsets. Second, it builds NewTransactionTable to maintain possible frequent itemsets associated with *T*4. Then it updates ClosedTable and CidList referring to NewTransactionTable and ClosedTable. The processing is as shown in Figure 1.



T_4: {2 3 4 5}

Table 2. CidList(before T_4)

item	CidSet
{1}	{1, 4, 5}
{2}	{2, 4}
{3}	{1, 2, 3, 4, 5}
{4}	{1}
{5}	{2, 4}

CidSet: {1, 2, 3, 4, 5}

Table 3. ClosedTable (θ=0.1, before T_4)

Cid	CP	SCP
0	{0}	0
1	{1 3 4}	0.64
2	{2 3 5}	1.8
3	{3}	2.44
4	{1 2 3 5}	1
5	{1 3}	1.64

Table 4. NewTransactionTable

TempItem	Cid
{2 3 4 5}	0
{3 4}	1
{2 3 5}	2
{3}	3

Table 5. ClosedTable (θ=0.1, after T_4)

Cid	CP	SCP
0	{0}	0
1	{1 3 4}	0.512
2	{2 3 5}	2.44
3	{3}	2.952
4	{1 2 3 5}	0.8
5	{1 3}	1.312
6	{2 3 4 5 *7*}	1
7	{3 4}	1.512

Table 6. CidList(after T_4)

item	CidSet
{1}	{1, 4, 5}
{2}	{2, 4, 6}
{3}	{1, 2, 3, 4, 5, 6, 7}
{4}	{1, 6, 7}
{5}	{2, 4, 6}
{7}	*{6}*

T_4: {2 3 4 5 *7*}

Figure 1. Schematic of the process of handling new transaction T_4.

```
Algorithm 2: TDMCSADD()

Processing new transactions
1 Add $T_{new}$ To NewTransactionTable
2 Let setcid($T_{new}$)={ $\cup$CidSet(item$_i$),item$_i$ $\in T_{new}$ }
3 For $C_{id}$ In setcid($T_{new}$) Do
3.1 interS= $T_{new} \cap$ClosedTable($C_{id}$)
3.2 For TempItem In NewTransactionTable Do
        If interS $\in$ClosedTable
        Then update support(interS)
        Else If support(interS) $\geq N\times\varepsilon\times\theta$
          Then Add (interS, $C_{id}$) To ClosedTable
    End For
3.3 For (TempItem, $C_{id}$) In NewTransactionTable Do
        If(TempItem==ClosedTable($C_{id}$))
        Then update support(ClosedTable($C_{id}$))
        Else update support(TempItem)
        If(newsupport(TempItem) $\geq N\times\varepsilon\times\theta$)
        Then Add (TempItem, newsupport(TempItem))
          To ClosedTable
        If item $\in T_{new}$ And item Is Not In CidList
        Then Add item To CidList
    End For
4 End For
```

Illustrate the process of algorithm *TDMCSADD*(T_{new}) to handle new transactions. Data stream is shown as in Table 1 including 4 transactions. Let decay factor f=0.8, minimum support threshold θ=0.1. When new transaction T_4={2, 3, 4, 5} is arrived, information of *ClosedTable*, *CidList* and *NewTransactionTable* are as shown in Tables 2-4. There are 5 frequent itemsets in *ClosedTable*, 5 items in *CidList* and NULL in initialized *NewTransactionTable*. When new tranaction T_4 arrived, there are some steps to process it.

- *Step 0*. Add values <T_4, 0> to *New-TransactionTable*.
- *Step 1*. Compare items in T_4 and items in *CidList* to get the *CidSet* associated with T_4.
- *Step 2*. Add itemsets associated with T_4 to *NewTransactionTable* according with *CidSet*, as shown in Table 4. That is for each element *Cid* of *CidSet* to get the intersections of T_4 and *ClosedTable*.
- *Step 3*. Update *ClosedTable* with information of *NewTransactionTable*, then get Table 5.
- *Step 4*. Update *CidList* with information of novel *NewTransactionTable* and *ClosedTable*. It will be updated under two conditions: the emergence of a new frequent itemset or a new item. Assuming T_4 contains a new item (7) represented in italics in Figure 1. Then add value < {7}, {6} > to *CidList*. Meanwhile, there are two new frequent itemsets in *ClosedTable*, then update *CidList* too.

For continuous generation of new transactions, TDMCSADD() repeats steps above for processing. When you need to remove the information of old transactions from sliding window, use algorithm TDMCSREMOVE(T_{old}). The main process is similar to algorithm TDMCSADD(T_{new}). First, generate OldTransactionTable to maintain the information about old transaction T_{old}. Second, find the intersections of OldTransactionTable and ClosedTable. Next, update or delete ClosedTable. In order to improve efficiency of algorithm, it only adds deleting flags and does not do the actual deletion.

When steps of processing transactions meet the pruning step, call function *PRUNING*() to do pruning operations. This is the actual process of deleting operations, and it updates and deletes *ClosedTable* and *CidList*. The algorithm is described as shown in Algorithm 3.

```
Algorithm 3: PRUNING()

Dropping information of historical transactions.
1 For Each $C_{id}$ In ClosedTable
2 Remove itemsets(with DeleteFlag) From closedTable
3   If support($C_{id}$)< $N \times \varepsilon \times \theta$
    Then Remove itemsets From closeTable
4   Update cidlist
5 End For
```

If only the parameters minimum support threshold θ and decay factor f are used in algorithms, some possible frequent patterns might be lost. Such as, let minimum support θ=0.1, then complete result set is mined. Therefore, many useless patterns may be discovered. If setting θ=0.3, when T_4 arrived, frequent itemsets should meet the support count 4×0.3=1.2. Then generate three frequent itemsets in *ClosedTable* as shown in Table 7. From Table 7 and Table 5, it is clear that the pattern {3 4} ($freq_d$({3 4})=1.512>1.2)) is missing.

Table 7. ClosedTable (θ=0.3).

Cid	CP	SCP
0	{0}	0
1	{2 3 5}	2.44
2	{3}	2.952
3	{1 3}	1.312

The reason is that the frequency with decay factor of pattern P is smaller than its original frequency, that is $freq_d(P) < freq(P)$. Let f=0.8, then $freq_d(P)$<1/(1-0.8)=5 as calculated by Formula 6. Therefore, if only the minimum support threshold θ is used under the time decay model, some frequent patterns may be lost, for its support count may be less than $\theta \times N$. To solve this problem, ε is introduced as the maximum support error. Therefore, the frequent support of mined pattern needs to meet $N \times \varepsilon \times \theta$ instead of $N \times \theta$.

$$\begin{aligned} freq_d(P,T_m) &= freq_d(P,T_{m-1}) \times f + r \\ &= \sum_i r_i \times f^{m-i} = r_1 \times f^{m-1} + r_2 \times f^{m-2} + \ldots + r_m \\ &\le f^{m-1} + f^{m-2} + \ldots + 1 \le \frac{1}{1-f} \end{aligned} \tag{6}$$

The next question is how to determine the value of the decay rate f after given parameters: minimum support, maximum support error and sliding window size. Suppose recall is 100%, a lower bound of the decay factory is showed by Formula 7 [3, 12]. Formula 8 [3] shows the upper bound of f under the condition of precision=100%. The usual methods set f to random value between lower bound and upper bound or set f to one bound of them [3, 12]. Because both recall=100% and precision=100% cannot be achieved at same time, selected f should balance these two conditions.

$$f \ge \sqrt[(2N-\theta N-1)]{[(\theta-\varepsilon)/\theta]^2}, \quad when \;\; recall = 100\% \tag{7}$$

$$f < \frac{(\theta-\varepsilon)N-1}{(\theta-\varepsilon)N}, \quad when \;\; precision = 100\% \tag{8}$$

In this paper, a new way to set decay factor is proposed. Let sliding window size be 10K. Parameters θ, ε and f are shown in Table 8. The third column f_{recall} means the lower bound when assuming recall is 100%. The last column $f_{\text{precision}}$ implies the upper bound when assuming precision is 100%. There are three policies to select *f*, as shown in Formula 9. For example, let θ=0.025 and ε=0.05, then set $f=f_{\text{recall}}$=0.999995, $f=f_{\text{precision}}$=0.995789 or $f=f_{\text{average}}$=0.997892. Verified by experiments (in Section 4), setting f to $f_{average}$ can get the more balanced recall and precision of algorithm. Therefore, setting $f=f_{average}$ is more reasonable than setting *f* to random value between f_{recall} and $f_{precision}$ or one of it.

Table 8. Time decay factor.

θ	ε×θ	frecall (recall=100%)	fprecision (precision=100%)
0.05	0.05×θ	0.999995	0.997895
0.05	0.1×θ	0.999989	0.997778
0.05	0.5×θ	0.999929	0.996
0.025	0.05×θ	0.999995	0.995789
0.025	0.1×θ	0.999989	0.995556
0.025	0.5×θ	0.99993	0.992

$$f_1 = f_{recall} = \sqrt[(2N-\theta N-1)]{[(\theta-\varepsilon)/\theta]^2}$$

$$f_2 = f_{precision} = \frac{(\theta-\varepsilon)N-1}{(\theta-\varepsilon)N} \tag{9}$$

$$f_3 = \frac{\sqrt[(2N-\theta N-1)]{[(\theta-\varepsilon)/\theta]^2} + \frac{(\theta-\varepsilon)N-1}{(\theta-\varepsilon)N}}{2}$$

4. Performance

The experiments were performed on a 2.1 GHZ CPU with 2GB memory, and run on Win7. All the algorithms were coded in Java language. To evaluate the performance of these algorithms, real and synthetic datasets were used. Real dataset from UCI [7] describes the page visits of users who visited msnbc.com on September 28, 1999. Visits are recorded at the level of URL category (see below) and are recorded in order. There are 989818 transactions and the average length of transactions is 5.7. It is a high dense and similar data stream. Synthetic datasets were generated from IBM data generator. There are four synthetic datasets with different average pattern and transaction sizes: T5I5D1000K, T10I4D1000K, T10I5D1000K, T10I10D1000K, T20I5D1000K and T20I20D1000K. These were used to analyze the performance of data stream on different density. The parameters are described as follows: *D* is the total number of transactions; *I* is the average size of maximal potential patterns; *T* is the average length of transactions. Such as, T10I5D1000K means the average length of transactions is 10, average length of maximal potential patterns is 5, and number of transactions is 1000K.
The mainly purpose of experimental was to analyze:

1. The ways to set decay factor *f*. Compared the algorithm performances with setting *f* to random value, boundary value and average value.
2. The effects of sliding window sizes on performance of algorithm TDMCS.
3. The effects of pruning steps on performance of algorithm TDMCS.
4. The comparison the performances of algorithms TDMCS and CloStream* [25], MSW [12] and SWP [3].

Compared to algorithm CloStream [24], algorithm CloStream* used the sliding window to deal with recent transactions to mine closed frequent patterns. CloStream handled all the transactions, so it did not apply to mine unlimited data stream. Therefore, in this paper we compared TDMCS with CloStream*. Similar pattern tree structures were used in algorithms MSW and SWP. And both of them set decay factor as lower bound. In this paper, some modifications were made to the two original algorithms for mining closed frequent patterns instead of complete patterns.

The maximum support error ε was 0.1. The value of the time decay factor f was average of low bound and high bound to balance 100% recall and 100% precision. The sliding window sizes N were 0.1M to 0.8M and the minimum support thresholds were 0.06 to 0.1. The values of f in the experiments are shown in Table 9.

Table 9. Values of decay factors.

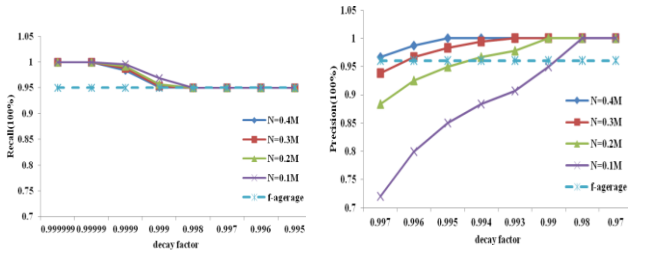
fid	θ	ε×θ	N	f
f_1	0.06	0.006	0.1M	0.990686
f_2	0.06	0.006	0.2M	0.995343
f_3	0.06	0.006	0.3M	0.996895
f_4	0.06	0.006	0.4M	0.997672
f_5	0.06	0.006	0.5M	0.998137
f_6	0.06	0.006	0.7M	0.998669
f_7	0.06	0.006	0.8M	0.998836
f_8	0.07	0.007	0.1M	0.992009
f_9	0.08	0.008	0.1M	0.993001
f_{10}	0.09	0.009	0.1M	0.993772
f_{11}	0.1	0.01	0.1M	0.994389

At first, verify the reasonableness of setting $f=f_{\text{average}}$. The relationship between decay factor *f* and minimum support threshold θ, maximal support error threshold ε needs to be discussed in algorithm TDMCS, in order to determine the optimum parameter value of *f*.

Let minimum support θ=0.05, the values of recall and precision of TDMCS on *msnbc* with different decay factors are shown in Figure 2. Abscissa axis means the random value between f_{recall} and $f_{\text{precision}}$ at different window size *N*. Vertical axis means the recall and precision at different *N*, and the dashed line means to set $f=f_{\text{average}}$. From this figure, it can be concluded that:

1. With the decreasing of *f*, recall is decreasing and precision is increasing.
2. The trends of recall and precision at different *N* are similar.
3. When setting $f=f_{\text{average}}$, the values of recall and precision are fixed. They are almost unaffected by sizes of sliding windows.
4. The values of recall and precision can be balanced by setting $f=f_{\text{average}}$.

When setting $f=f_{recall}$, $f=f_{precision}$ and $f=f_{average}$, we compare the average values of recalls and precisions of algorithm with different sliding window. It can be concluded that almost 100% recall when setting $f=f_{recall}$, and get lowest recall with setting $f=f_{precision}$ are proposed. When setting $f=f_{average}$, the value of recall is between them. Setting $f=f_{precision}$ and $f=f_{average}$ can get almost the same precisions. But the value of precision is lowest when setting $f=f_{recall}$. Therefore, recall and precision of algorithm can be more balanced by setting $f=f_{average}$ than setting $f=f_{recall}$ and $f=f_{precision}$.



a) Recall. b) Precision.

Figure 2. Variations of recall and precision with different decay factors.

Next, we compare the performances of algorithms with setting f to $f_{average}$ and random values. To make the random value more reasonable, set them in the range of (0.9, 1), and denoted as f_{random}. Use function *Math.random*() to generate 5 random values to set decay factors. The performance of TDMCS on *msnbc* is shown in Figure 3. As can be seen, when setting $f=f_{average}$ and $f=f_{random}$, the values of precision are little different. But the values of recall are very different when setting f to random values. Therefore the performance of algorithm is unstable. The performance with $f=f_{average}$ is significantly better than $f=f_{random}$, and the result set is stable.

It can get the same conclusions when processing synthetic data streams. Thus, setting decay factor to average value is reasonable.

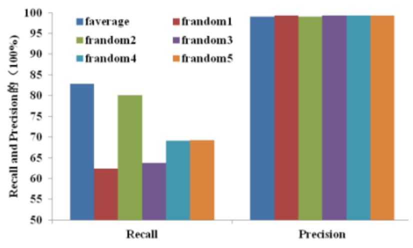


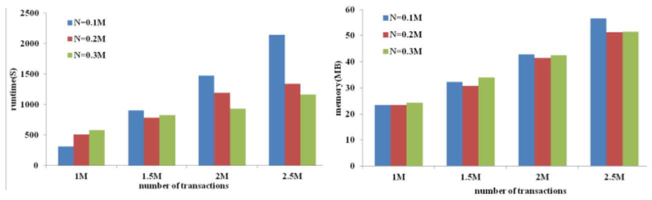
Figure 3. The performance studies on msnbc with $f_{average}$ and f_{random}.

The second experiment analyzes the influence of window sizes on algorithm TDMCS. Let sliding window size N=0.1M, 0.2M and 0.3M; the minimum support θ=0.06; the decay factor $f= f_1, f_2, f_3$ as shown in Table 9; pruning step P=0.1M [3, 12]. The runtime and space cost of algorithm TDMCS on msnbc are compared in Figure 4.

The performance of TDMCS on data stream *msnbc* is shown in Figure 4 when processing 1M, 1.5M, 2M and 2.5M transactions. Figure 4-a shows the runtime and in which abscissa axis means number of transactions. It can be seen that:

1. When the number of transactions is small, the increment of window size leads to a slight increment of runtime;
2. With the increment of processing transaction number, the runtime with big window size is lower than runtime with small window size.

Figure 4-b shows the memory usage. It is clear that the effect of different window size on memory usage is small. From time and space consumptions, it can be concluded that the runtime of algorithm TDMCS on data stream *msnbc* is greatly different as different size N under the same number of transactions. And the memory usage is almost same as different size N. Thence, in terms of space complexity, TDMCS applies to discovering frequent patterns of any window size.



a) Runtimes. b) Memory usage.

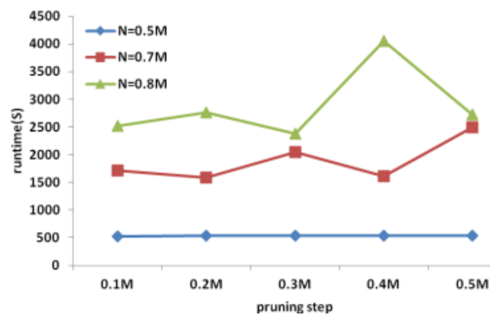
Figure 4. The performance of TDMCS on msnbc based on different sizes of windows.

Thirdly, we analyze the effect of pruning step on performance of algorithm TDMCS over msnbc. Sliding window size N was in {0.5M, 0.7M, or 0.8M}. Pruning step P ($P \leq N$) was in {0.1M 0.2M, 0.3M, 0.4M, 0.5M}. Minimum support θ was set to 0.06 and f was in {f_5, f_6, f_7}.

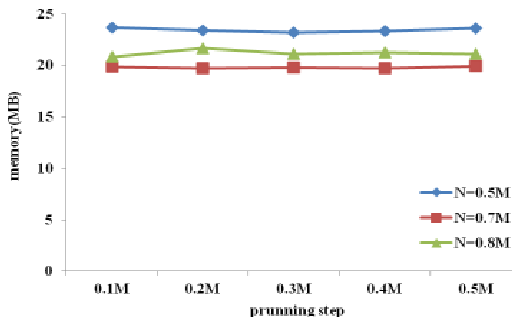
Figure 5-a shows the runtime of TDMCS on data stream *msnbc* with different window sizes and different pruning steps. From the performance of runtime it can be concluded that:

1. Optimal pruning step length is related to sliding window size.
2. When value P is different, the runtime is obvious different with N.

The memory usage is shown in Figure 5-b. It is clear that the effect of pruning step on memory usage is small. From Figure 5 it can be seen that the length of pruning step has little influence on runtime and memory usage when window size is small. And when window size is big, pruning step has a wider range influence on runtime. Therefore, set the parameter P=0.1M when N=0.1M is reasonable. Figure 5-b also shows that algorithm TDMCS applies to discover frequent patterns of any size of window.



a) Runtimes

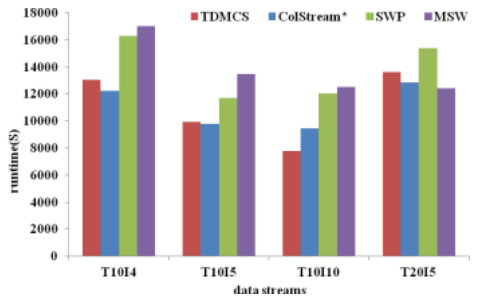


b) Memory usage

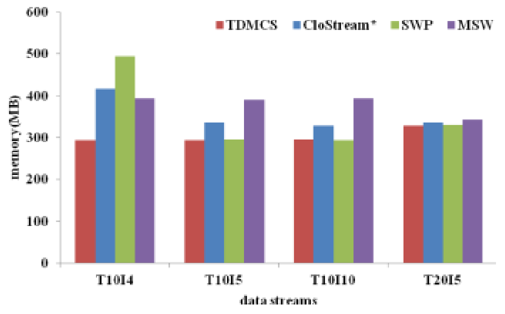
Figure 5. The performance study of TDMCS on data stream msnbc based on different sizes of windows and pruning steps.

Finally, we compare the performances of TDMCS and classical algorithms with different sliding window sizes. Parameters minimum support θ is 0.06, maximal support error ε is 0.1, pruning step P is 0.1M, sliding window size N is from 0.1M to 0.5M and decay factor f is from f_1 to f_4 as shown in Table 9.

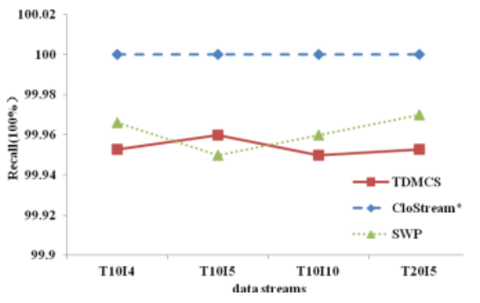
The performances of four algorithms processing synthetic data streams are shown in Figure 6, which are average values of recalls and precisions under different sliding window sizes. Four data streams with different lengths of transactions or patterns are used, including: T10I4, T10I5, T10I10 and T20I5. Figure 6-a shows the runtimes of four algorithms. Overall, the runtimes of TDMCS and CloStream* are lower than the other two algorithms. This is because algorithm CloStream* does not process data with decay operations and algorithm TDMCS uses closure operator. The memory usages of algorithms are shown in Figure 6-b. The memory usage of TDMCS is the lowest of all. But the differences between four algorithms are not too much. Figure 6-c and Figure 6-d show the recalls and precisions of algorithms. Both algorithms MSW and SWP set decay factor to lower bound, so we only compared with SWP. The recall of algorithm CloStream* is the highest of all for it does not use time decay model, but the precision is the lowest of all. The recall and precision of algorithm SWP are in the middle of these algorithms. The recall of TDMCS is about 1% lower than other two algorithms, but the precision of TDMCS is about 10% higher than CloStream* and about 4% higher than SWP. Therefore, algorithm TDMCS can get more balance recall and precision than the other three methods. From performances of four algorithms on data streams with different pattern lengths, such as T10I4, T10I5 and T10I10, or data streams with different transaction lengths, such as T10I5 and T20I5, it can be concluded that algorithm TDMCS is more suitable to process data streams with long transactions and long patterns.



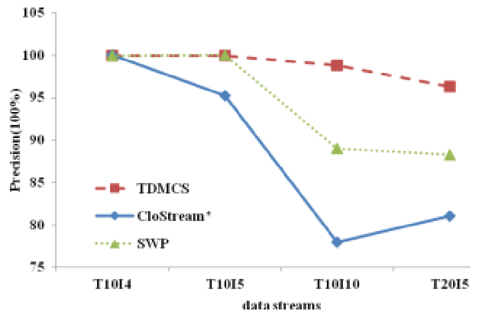
a) Runtimes.



b) Memory usage.



c) Recall.



d) Precision.

Figure 6. The performance studies of algorithms on synthetic data streams.

5. Summary

Data stream is a fluid, continuous, unbounded and time ordered sequence of data transactions generated at a rapid rate. Due to the knowledge contained in data stream may change over time, concept drift should be taken into account when mining frequent patterns.

Normally, recent transactions contain more important information than historical transactions, thus they should be treated differently. Considering the data stream characteristics, an efficient algorithm TDMCS is proposed in this paper. It is used to mining closed frequent patterns, and it based on time decay model and sliding window model. It uses closure operator to improve the efficiency of mining closed frequent itemsets. In order to balance the recall and precision, a novel manner by average the lower bound and higher bound is provided in this paper. It uses frame minimum support-maximum support error-decay factor to avoid concept drift and discover more reasonable and compact result set. The performance of the proposed algorithm was investigated using experiments. The results show that it is efficient and scalable, and it applies to mining high dense data stream and long patterns.

References

[1] Chang T., "Mining Frequent User Query Patterns From xml Query Streams," *The International Arab Journal of Information Technology*, vol. 11, no. 5, pp. 452-458, 2014.

[2] Chen H., "Mining Top-K Frequent Patterns over Data Streams Sliding Window," *Journal of Intelligence Information System*, vol. 42, no. 1, pp. 111-131, 2014.

[3] Chen H., Shu L., Xia J., and Deng Q., "Mining Frequent Patterns in a Varying-Size Sliding Window of Online Transactional Data Streams," *Information Sciences*, vol. 215, no. 12, pp. 15-36, 2012.

[4] Cheng J., Ke Y., and Ng W., "Maintaining Frequent Closed Itemsets over a Sliding Window," *Journal of Intelligent Information Systems*, vol. 31, no. 3, pp. 191-215, 2008.

[5] Chi Y., Wang H., Yu P., and Muntz R., "Catch the Moment: Maintaining Closed Frequent Itemsets over a Data Stream Sliding Window," *Knowledge and Information Systems*, vol. 10, no. 3, pp. 265-294, 2006.

[6] Farzanyar Z., Kangavari M., and Cercone N., "Max-FISM: Mining (Recently) Maximal Frequent Itemsets over Data Streams Using the Sliding Window Model," *Computers and Mathematics with Applications*, vol. 64, no. 6, pp. 1706-1718, 2012.

[7] Frank A. and Asuncion A., http://archive.ics.uci.edu/ml, Last Visited 2010.

[8] HewaNadungodage C., Xia Y., Lee J., and Tu Y., "Hyper-Structure Mining of Frequent Patterns in Uncertain Data Streams," *Knowledge and Information Systems*, vol. 37, no. 1, pp. 219-244, 2013.

[9] Jiang N. and Gruenwald L., "CFI-Stream: Mining Closed Frequent Itemsets in Data Streams," *in Proceeding of ACM SIGKDD Internal Conference on Knowledge Discovering and Data Mining*, New York, pp. 592-597, 2006.

[10] Lee G., Yun U., and Ryu K., "Sliding Window Based Weighted Maximal Frequent Pattern Mining over Data Streams," *Expert Systems with Applications*, vol. 41, no. 2, pp. 694-708, 2014.

[11] Li H., Ho C., and Lee S., "Incremental Updates of Closed Frequent Itemsets over Continuous Data Streams," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2451-2458, 2009.

[12] Li G. and Chen H., "Mining the Frequent Patterns in an Arbitrary Sliding Window over Online Data Streams," *Journal of Software*, vol. 19, no. 10, pp. 2585-2596, 2008.

[13] Li H., Zhang N., Zhu J., and Cao H., "Frequent Itemset Mining over Time-Sensitive Streams," *Chinese Journal of Computers*, vol. 35, no. 11, pp. 2283-2293, 2012.

[14] Li H., Ho C., Chen H., and Lee S., "A Single-Scan Algorithm for Mining Sequential Patterns from Data Streams," *International Journal of Innovative Computing, Information and Control*, vol. 8, no. 3A, pp. 1799-1820, 2012.

[15] Manku Q. and Motwani., "Approximate Frequency Counts over Streaming Data," *in Proceeding of the 28th International Conference on Very Large Data Bases*, Hong Kong, pp. 346-357, 2002.

[16] Nabil H., Eldin A., and Belal M., "Mining Frequent Itemsets from Online Data Streams: Comparative Study," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 7, pp. 117-125, 2013.

[17] Nori F., Deypir M., and Sadreddini M., "A Sliding Window based Algorithm for Frequent Closed Itemset Mining over Data Streams," *Journal of Systems and Software*, vol. 86, no. 3, pp. 615-623, 2013.

[18] Patnaik D., Laxman S., Chandramouli B., and Ramakrishnan N., "A General Streaming Algorithm for Pattern Discovery," *Knowledge and Information Systems*, vol. 37, no. 3, pp. 585-610, 2013.

[19] Shie B., Yu P., and Tseng V., "Efficient Algorithms for Mining Maximal High Utility Itemsets from Data Streams with Different Models," *Expert Systems with Applications*, vol. 39, no. 17, pp. 12947-12960, 2012.

[20] Tang K., Dai C., and Chen L., "A Novel Strategy for Mining Frequent Closed Itemsets in Data Streams," *Journal of Computers*, vol. 7, no. 7, pp. 1564-1572, 2012.

[21] Tsai P., "Mining top-K Frequent Closed Itemsets over Data Streams Using the Sliding Window Model," *Expert Systems with Applications*, vol. 37, no. 10, pp. 6968-6973, 2010.

[22] Wong R. and Fu A., "Mining Top-K Frequnt Itemsets form Data Streams," *Data Mining and Knowledge Discovery*, vol. 13, no. 2, pp. 193-217, 2006.
[23] Yang B. and Huang H., "TOPSIL-Miner: an Efficient Algorthm for Mining Top-K Significant Itemsets over Data Streams," *Knowledge and Information Systems*, vol. 23, no. 2, pp. 225-242, 2010.
[24] Yen S., Lee Y., Wu C., and Lin C., "An Efficient Algorithm for Maintaining Frequent Closed Itemsets over Data Stream," *Next-Generation Applied Intelligence*, vol. 5579, no. 1, pp. 767-776, 2009.
[25] Yen S., Wu C., and Lee Y., "A Fast Algorithm for Mining Frequent Closed Itemsets over Stream Sliding Window," *in Proceeding of IEEE International Conference on Fuzzy Systems*, Taipei, pp. 996-1002, 2011.
[26] Yu J., Chong Z., Lu H., and Zhou A., "False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams," *in Proceeding of the 30th International Conference on Very Large Data Bases*, Toronto, pp. 204-215, 2004.

Han Meng, born in 1982, Ph.D. candidate, associate professor. Her research interests include data mining and machine learning.

Jian Ding, born in 1977, M.S., associate professor. His research interests include machine learning and data mining.

Juan Li, born in 1975, M.S., associate professor. Her research interests include information security and cloud computing.