Mining Frequent Itemsets from Online Data Streams: Comparative Study

HebaTallah Mohamed Nabil
Faculty of Computers and Information Fayoum University
Fayoum, Egypt

Ahmed Sharaf Eldin
Faculty of Computers and Information Helwan University
Cairo, Egypt

Mohamed Abd El-Fattah Belal
Faculty of Computers and Information Helwan University
Cairo, Egypt

Abstract—**Online mining of data streams poses many new challenges more than mining static databases. In addition to the one-scan nature, the unbounded memory requirement, the high data arrival rate of data streams and the combinatorial explosion of itemsets exacerbate the mining task. The high complexity of the frequent itemsets mining problem hinders the application of the stream mining techniques. In this review, we present a comparative study among almost all, as we are acquainted, the algorithms for mining frequent itemsets from online data streams. All those techniques immolate with the accuracy of the results due to the relatively limited storage, leading, at all times, to approximated results.**

Keywords*—*Data mining; frequent itemsets; data stream; sliding window model; landmark model; fading model.

I. Introduction

Recently, the data generation rates in some data sources become faster than ever before. Examples include network traffic analysis, Web click stream mining, network intrusion detection, sensor networks, web logs, and on-line transaction analysis. This rapid generation of continuous streams of information has challenged our storage, computation and communication capabilities in computing systems. Systems, models and techniques have been proposed and developed over the past few years to address those challenges [Gaber M. et al., 2005].

In [Babcock B. et al., 2002] and [Lin C.H. et al., 2005], the Data Stream Model is characterized by that, some or all of the input data that are to be operated on are not available for random access from disk or memory, but rather arrive as one or more *continuous data streams*.

Data streams differ from the conventional stored relation model in several ways: *1) Continuity:* Data continuously arrive at a high rate. *2) Expiration:* Data can be read only once. *3) Infinity:* The total amount of data is unbounded. These characteristics lead respectively to the following challenges [Zhu Y. and Shasha D. 2002] in mining data streams: 1) Limited memory space. 2) Each item in a stream could be examined only once. 3) The mining result should be generated as fast as possible.

The infinite nature of these data sources is a serious obstacle to the use of most of the traditional methods since available computing resources are limited. One of the first effects is the need to process data as they arrive. The amount of previously happened events is usually overwhelming, so they can be either dropped after processing or archived separately in secondary storage. In the first case access to past data is obviously impossible whereas in the second case the cost for data retrieval is likely to be acceptable only for some ”ad hoc” queries, especially when several scan of past data are needed to obtain just one single result [Silvestri C., 2006]. In the process of mining frequent itemset, traditional methods for static data usually read the database more than once. However due to the consideration of performance and storage constraints, on-line data stream mining algorithms are restricted to make only one pass over the data. Thus, traditional methods cannot be directly applied to data stream mining [Pauray S. and Tsai M., 2009].

II. Background

According to [Li H. F. et al, 2006], data streams are further classified into: *1) offline data streams:* which characterized by discontinuity or regular bulk arrivals [Manku G. and Motwani R., 2002], such as a bulk addition of new transactions as in a data warehouse system, and *2) online data streams:* which characterized by real-time updated data that come one by one in time, such as a continuously generated transaction as in a network monitoring system.

A *transaction data stream* is a sequence of incoming transactions and an excerpt of the stream is called a *window*. A window, W, can be (1) either *time-based* or *count-based*, and (2) either a *landmark window* or a *sliding window*. W is time-based if W consists of a sequence of fixed-length time units, where a variable number of transactions may arrive within each time unit. W is count-based if W is composed of a sequence of batches, where each batch consists of an equal number of transactions. W is a landmark window if $W = [T1, T2, \ldots, T\tau]$; W is a sliding window if $W = [T\tau-w+1, \ldots, T\tau]$, where each Ti is a time unit or a batch, $T1$ and $T\tau$ are the *oldest* and the *current* time unit or batch, and w is the number of time units or batches in the sliding window, depending on whether W is time-based or count-based. Note that a count-based window can also be captured by a time-based window by assuming that a uniform number of transactions arrive within each time unit.

An itemset X is a Frequent Itemset (FI) in W, if $\sup(X) \geq \sigma$, where σ ($0 \leq \sigma \leq 1$) is a user-specified *minimum support threshold*. In the process of mining data streams, it is necessary to keep not only the FIs, but also the infrequent itemsets that are promising to be frequent later, since an infrequent itemset may become frequent later in the stream. Therefore, many of the existing approximate mining algorithms used a *relaxed* minimum support threshold (also called a *user-specified error parameter*), ϵ, where $0 \le \epsilon \le \sigma \le 1$, to obtain an extra set of itemsets that are potential to become frequent later.

There are many algorithms for mining frequent itemsets from data streams; according to [Pauray S. and Tsai M., 2009], all those algorithms are fallen into one of the following data stream mining models: *1) Landmark model, 2) Fading model and 3) Sliding window model.*

III. Approaches Of Mining FIs From Data Streams

A. Landmark model

Which considers all the data from a specified point of time (usually the time the system starts), to the current time. All the data considered are treated equally. In this model, knowledge discovery is performed based on the values between a specific timestamp called landmark and the present. See Figure 1.

In [Cormode. G., 2007], an algorithm called Lossy Counting is presented. It produces an approximate set of FIs over the entire history of a stream. The stream is divided into a sequence of buckets and each bucket consists of $B = \lceil 1/\epsilon \rceil$ transactions. It processes a batch of transactions arriving on the stream at a time, where each batch contains β buckets of transactions. The idea of maximum possible error is used to maintain all the possible frequent itemsets. Although the output is approximate, the error is guaranteed not to exceed a user-specified threshold.

According to [Cormode. G., 2007], this method attempts to use the available space as fully as possible. As, for each new transaction, it generates all the subsets, and stores them in a compact trie-based structure. When the space is full, it uses a pruning algorithm based on frequent items algorithms to delete the least frequent itemsets, and track the error in the estimated counts of each item.

In [Yu J. X. et al, 2004], an algorithm called FDPM is derived from the Chernoff bound, to approximate a set of FIs over a landmark window. Suppose that there is a sequence of N observations and consider the first n (n << N) observations as independent coin flips such that Pr(head) = p and Pr(tail) = 1 − p. Let r be the number of heads. Then, the expectation of r is np. The Chernoff bound states, for any $\gamma > 0$:

$$\Pr(|r - np| \ge np\gamma) \le 2e\hat{}((-np\gamma^2)/2) \qquad (1)$$

After applying some substitutions and derivations:

$$\Pr\{ p - \epsilon \le \bar{r} \le p + \epsilon \} \ge (1-\delta) \qquad (2)$$

The underlying idea of the FDPM algorithm is explained as follows. First, a memory bound, $n_0 \approx (2 + 2\ln(2/\delta))/\sigma$, is also derived. Given a probability parameter, δ, and an integer, k. The batch size, B, is given as $k \cdot n_0$. Then, for each batch of B transactions, FDPM employs an existing non-streaming FI mining algorithm to compute all itemsets whose support in the current batch is no less than $(\sigma - \epsilon_B)$, where $\epsilon_B = \sqrt{(2\sigma\ln(2/\delta))/B}$. The set of itemsets computed are then merged with the set of itemsets obtained so far for the stream. If the total number of itemsets kept for the stream is larger than $c \cdot n_0$, where c is an empirically determined float number, then all itemsets whose support is less than $(\sigma - \epsilon_N)$ are pruned, where N is the number of transactions received so far in the stream and $\epsilon_N = \sqrt{(2\sigma\ln(2/\delta))/N}$. Finally, FDPM outputs those itemsets whose frequency is no less than σ N.

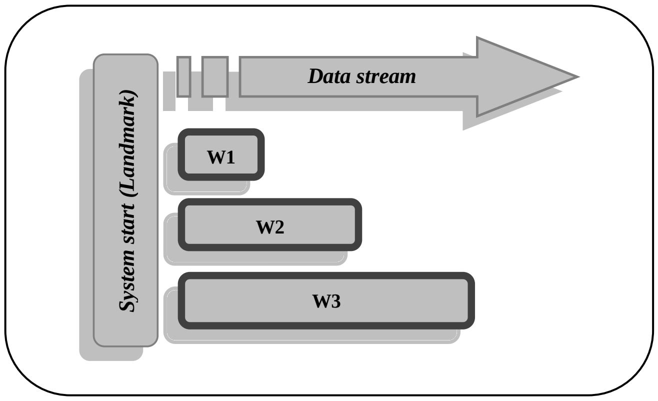

Fig.1. The landmark model

It solves the problem of huge number of sub-FIs problem, by first using a constant lowered minimum support threshold to compute a set of potential FIs and then using a gradually increasing lowered minimum support threshold to control the total number of sub-FIs kept in memory, but with a drawback of producing false negatives.

In [Jin R. and Agrawal. G., 2005], an algorithm called StreamMining is proposed. It is built on the idea of [Karp R.M. et al, 2003] to determine frequent items (or 1-itemsets). In [Karp R.M. et al, 2003] a two pass algorithm was presented for this purpose, which requires only (1/θ) memory, where θ is the desired support level. Their first pass computes a superset of frequent items, and the second pass eliminates any false positives. StreamMining algorithm addressed three major challenges in applying their idea for frequent itemset mining in a streaming environment. First, it developed a method for finding frequent k-itemsets, while still keeping the memory requirements limited. Second, it developed a way to have a bound on the superset computed after the first pass. Third, it developed a data structure and a number of other implementation optimizations to support efficient execution. This data structure called *TreeHash*, which implements a prefix tree using a hash table. It has the compactness of a prefix tree and allows easy deletions like a hash table. It also uses a relaxed minimum support threshold ϵ, like almost all the mining algorithms for data streams, so the memory requirements increase proportional to $1/\epsilon$. So, this algorithm should had to compute k-itemsets approximately after the first pass, without requiring any out-of-core or large summary structure, and ensure a provable bound on the accuracy of the results after the first pass on the dataset; because in streaming environments, second pass on the dataset is usually not feasible. Therefore, it is important that the set K computed above does not contain many false positives. It was different with [53]^{2002} in the space requirements. As, for finding frequent items, it takes O(1/θ) space, while [Manku G., and Motwani R., 2002] requires O((1/θ) log (θN)) space. As [Manku G., and Motwani R., 2002] requires an out-of-core data structure, while it used an in-core data structure. It also has deterministic bounds on the accuracy. One exception is datasets with the average length of an itemset is quite large. In such case, some additional knowledge of maximal frequent itemsets helps efficiency of our algorithms.

In [Liu X. et al, 2005], an algorithm called FP-DS is presented. It uses a Frequent Pattern structure similar to the FP-DS tree in [Han J. et al, 2000]. The user can obtain current frequent itemsets online continuously without pattern-delay. Compared with the existing related algorithms, the FP-DS algorithm is especially suitable for the mining of long frequent items. It is unnecessary to enumerate every subset on transactions, nor produce a lot of frequent candidate items. The FP-DS tree stores the potential frequent itemsets. It does not need to store all subsets of itemsets independently. It reduces the storage capacity of itemsets and moreover, the itemsets are put in the order of the descending sequence of support of global 1-itemset. The more frequently the items appear the closer to the root of the tree. Such a compression tree has a higher compression ratio.

In the Landmark model, all FIs are outputted, although they are approximated; in other words, the data stream from system start to the existing point is scanned for mining (considering historical data, not only recent data). The support count is computed from the entire data set between the landmark and the current time. But, it isn't aware of time (time unconscious) and therefore it cannot distinguish between new data and old ones. In other words, it losses the time information the itemsets mined.

B. Fading model

That is called the ***Fading*** model in [Chang J. and Lee W., 2004], the ***Damped*** model according to [Zhu Y. and Shasha D., 2002] or ***Time-titled*** model according to [Chen, Y. et al, 2002] and [Pauray S. and Tsai M., 2009]; which is considered a variation of the *landmark* model. It also considers data from the start of streams up to the current moment, but the time period is divided into multiple time slots or assigned different weights to transaction such that new ones have higher weights than old ones. In other words time slots in recent time period are assigned at a fine granularity, while those in ancient time period are assigned at a coarse granularity. In this model, recent sliding windows are more important than previous ones. See Figure 2.

In [Chang J.H. and Lee WS., 2003], an algorithm called estDec is proposed. It uses a decay rate, d ($0 < d < 1$), to diminish the effect of old transactions on the mining result. As a new transaction comes in, the frequency of an old itemset is discounted by a factor of d. Thus, the set of FIs returned is called recent FIs. estDec algorithm adopts the mechanism in [Hidber C., 1999] to estimate the frequency of the itemsets. For example, let the decay rate and the support count of itemset X be d and v, respectively. As a new transaction containing X arrives, the new support count of X is equal to $v \times d+1$. Obviously, when d equals 1, the time-fading model becomes the landmark model. Assume that the stream has received τ transactions, $\langle Y1, Y2, \ldots, Y\tau \rangle$. The decayed total number of transactions, $N\tau$, and the decayed frequency of an itemset, freqτ (X), are defined as follows:

$$N_\tau = d^{\tau-1} + d^{\tau-2} + \cdots + d^1 + 1 = \frac{1-d^\tau}{1-d} \qquad (3)$$

$$freq_\tau(X) = d^{\tau-1} \times w_1(X) + d^{\tau-2} \times w_2(X) + \cdots + d^1 \times w_{\tau-1}(X) + 1 \times w_\tau(X) \quad (4)$$

$$\text{where } w_i(X) = \begin{cases} 1 \text{ if } X \subseteq Y_i, \\ 0 \text{ otherwise} \end{cases}$$

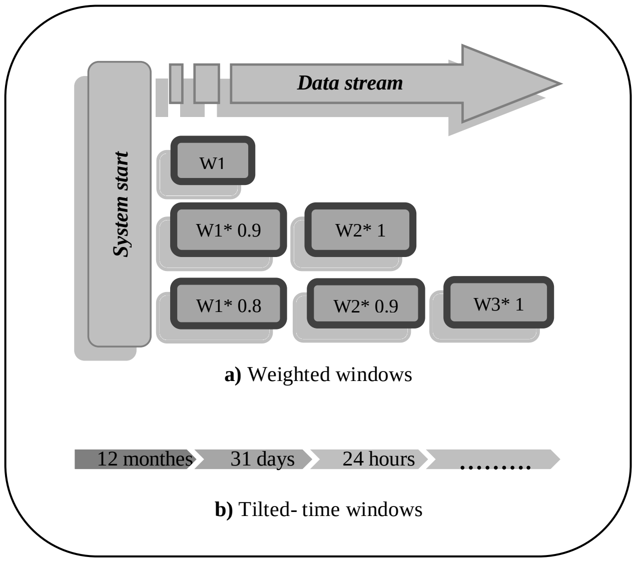

Fig.2. a) Fading model, b) Tilted time window

In [Giannella, J. et al, 2003], an algorithm called FP-Streaming is proposed. It proposed an FP-stream structure [Han J. et al, 2000] based algorithm, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique of [Chen, Y. et al, 2002]. Frequent patterns are maintained under a tilted-time window framework in order to answer time-sensitive queries. The frequency of an itemset is kept at a finer granularity for more recent time frames and at a coarser granularity for older time frames. For example, we may keep the frequency of an FI in the last hour, the last 2 h, the last 4 h, and so on.

The count of each itemset is asymmetrically distributed into multiple time slots such that the recent time period is assigned more time slots than the past. It is suitable for people to mine the recent data at a fine granularity while mining the long-term data at a coarse granularity. It computes a set of sub-FIs at the relaxed minimum support threshold, ϵ, over each batch of incoming transactions by using the FI mining algorithm, FP-growth [Han J. et al, 2000]. Two parameters, the minimum support count σ and the maximum support error ε where $\sigma \geq \varepsilon$, are used to classify all the itemsets into three categories: 1) Frequent: Support count is greater than and equal to σ. 2) Sub-frequent: Support count falls in $[\varepsilon, \sigma]$. 3) Infrequent: Support count is smaller than ε. Next, only frequent and sub-frequent itemsets are stored and organized in the FP-stream.

[Cohen E. and Strauss M., 2003] and [Chang J. and Lee W., 2004] have also provided variations of decay functions, like in [56]^{2003}, under the time-fading model.

The fading model was proposed to overcome the limitation of time unconscious in the landmark model. It diminishes the effect of the old and obsolete information of a data stream on the mining result. In other words, it considers the effect of old transactions in some way. It can assign different weights to transactions such that new ones have higher weights than old ones, these weights are decreasing as time passes by.

C. Sliding window model

In [Chang J.H and Lee WS., 2003], an algorithm called estWin is presented. At which the itemsets generated are maintained in a prefix tree structure, D. An itemset, X, in D has the following three fields: $\overline{\text{freq}}(X)$, err(X) and tid(X), where $\overline{\text{freq}}(X)$ is assigned as the frequency of X in the current window since X is inserted into D, err(X) is assigned as an upper bound for the frequency of X in the current window before X is inserted into D, and tid(X) is the ID of the transaction being processed, for X is inserted into D. For each incoming transaction Y with an ID tid_τ, estWin increments the computed frequency of each subset of Y in D. We prune an itemset X and all X's supersets if (1) $\text{tid}(X) \leq \text{tid}_1$ and $\overline{\text{freq}}(X) < \lceil \epsilon N \rceil$, or (2) $\text{tid}(X) > \text{tid}_1$ and $\overline{\text{freq}}(X) < \lceil \epsilon(N - (\text{tid}(X) - \text{tid}_1)) \rceil$. For each expiring transaction of the sliding window, those itemsets in D that are subsets of the transaction are traversed. For each itemset, X, being visited, if $\text{tid}(X) \leq \text{tid}_1$, $\overline{\text{freq}}(X)$ is decreased by 1; otherwise, no change is made since the itemset is inserted by a transaction that comes later than the expiring transaction. Then, pruning is performed on X as described before. Finally, for each itemset, X, in D, estWin outputs X as an FI if (1) $\text{tid}(X) \leq \text{tid}_1$ and $\overline{\text{freq}}(X) \geq \sigma.N$, or (2) $\text{tid}(X) > \text{tid}_1$ and $(\overline{\text{freq}}(X) + \text{err}(X)) \geq \sigma N$.

In [Chang JH. and Lee WS., 2004], an algorithm called estWin Lossy-counting-based is also presented as a similar method to [Chang J.H and Lee WS., 2003] based on the estimation mechanism of [Manku G. and Motwani R., 2002].

In [Lin C.H. et al, 2005], the first *time-sensitive sliding-window* was proposed, which regards a fixed time period as the basic unit for mining. At which the transaction data stream $TDS = T_1, T_2, \ldots, T_N$ is a continuous sequence of transactions, where N is the transaction identifier of latest incoming transaction T_N.

A transaction $T = (TU_{id}, T_{id}, \text{itemset})$, where TU_{id} is the identifier of the time unit, and T_{id} is the identifier of the transaction. A time-sensitive sliding window (TimeSW) in the transaction data stream is a window that slides forward for every time unit (TU). Each time unit TU_i consists of a variable number, $|TU_i|$, of transactions, and $|TU_i|$ is also called the size of the time unit. Hence, the current time-sensitive sliding window with w time units is $\text{TimeSW}_{N\_w+1} = [TU_{N\_w+1}, TU_{N\_w+2}, \ldots, TU_N]$, where N_{N_w+1} is the id of time unit of current TimeSW, and N is the TU_{id} of latest time unit TU_N. The window at each slide has a fixed number, w, of time units. The value $w = |TU_{N\_w+1}| + |TU_{N\_w+2}| + \ldots\ldots + |TU_N|$ is called the size of the time-sensitive sliding window and denoted as |TimeSW|.

It doesn't use a relaxed minimum support threshold, like almost all data stream mining algorithms; instead it maintained a data structure named the *discounting table* (DT) to retain the frequent itemsets with their support counts in the individual basic TUs of the current window.

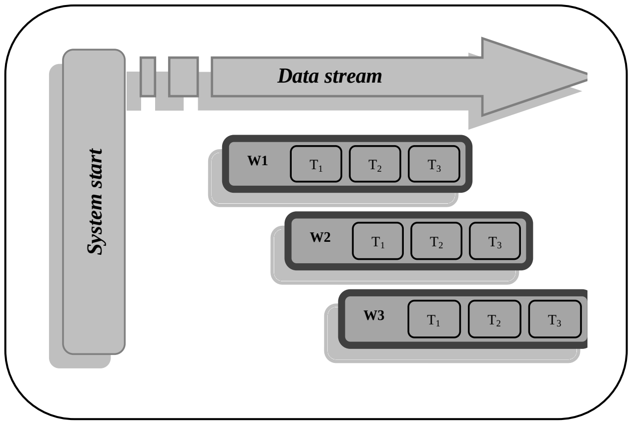

Fig.3. The sliding window model

Moreover, a data structure named the *Potentially Frequent-itemset Pool* (PFP) is used to keep the frequent itemsets in W_i and the frequent ones in TU_i. it includes the itemsets that are frequent in TU_i but not frequent in W_{i-1} in PFP because they are possibly frequent in W_i. Only the summary information derived from W_{i-1} is provided for mining frequent itemsets in W_i. It provides two alternatives to determine the frequent itemsets for output, having trade-off between accuracy and efficiency: *1) Recall-oriented and 2) Precision-oriented.*

In [Li H. F. et al, 2006] and [Li H. F. and Lee S. Y., 2009], an algorithm called MFI-TransSW is presented. It used an effective bit-sequence representation of items to reduce the time and memory needed to slide the windows. For each item X in the current SW, a bit-sequence with w bits, denoted as Bit(X), is constructed. If an item X is in the i-th transaction of current SW, the i-th bit of Bit(X) is set to be 1; otherwise, it is set to be 0. It consists of three phases: *1) window initialization phase:* when the number of transactions generated so far in a transaction data stream is less than or equal to a user-predefined sliding window size w, each item in the new incoming transaction is transformed into its bit-sequence representation; *2) window sliding phase:* after the sliding window TransSW becomes full, a new incoming transaction is appended to the sliding window, and the oldest transaction is removed from the window. The bitwise left shift operation is used to remove the aged transaction from the set of items in the current sliding window. After sliding the window, an item X in the current transaction-sensitive sliding window is dropped if and only if sup(X)TransSW = 0.; *3) frequent itemsets generation phase:* It is performed only when the up-to-date set of frequent itemsets is requested. a level-wise method is used to generate the set of candidate itemsets CI_k (candidate itemsets with k items) from the pre-known frequent itemsets FI_{k-1} (frequent itemsets with k-1 items) according to the Apriori property [Agrawal R. and Srikant R., 1994]. Then, the bitwise AND operation is used to compute the support (the number of bit 1) of these candidates in order to find the frequent k-itemsets FI_k.

In [Cheng J. et al, 2006], an algorithm called MineSW Algorithm is proposed. It is progressively increasing minimum support function. See preliminaries of time-sensitive sliding window model in the description of [Lin C.H. et al, 2005]. By contrast with other algorithms which uses an error parameter, ϵ , to control the mining accuracy, which leads to a dilemma. It tackles this problem by considering ϵ=r.σ as a relaxed MST , where r ($0 \leq r \leq 1$) is the *relaxation rate*, to mine the set of FIs over each time unit t in the sliding window, allowing to increase ϵ at the expense of slightly degraded accuracy, but significantly improves the mining efficiency and saves memory usage. When an itemset is retained in the window longer, its minimum support required to approach the minimum support of an FI. Thus, the number of potential FIs to be maintained is greatly reduced.

In [Kun Li. Et al, 2008], an algorithm called BFI Steam is presented. It is a Bounded FIs algorithm, which maintains all accurate frequent itemsets from sliding windows by monitoring the boundary between frequent itemsets and infrequent itemsets; it restricts the update process on a small part of the tree. Mining all frequent itemsets with accurate frequencies is just to traverse the tree. It has no candidate generation and it uses a prefix-tree based structure, called BFI-tree, to maintain all frequent itemsets in the sliding window. The BFI-tree is a prefix-tree based data structure and is derived from CET structure in the Moment algorithm [Chi Y. et al, 2004]. It uses a count-based sliding window with fixed size of N, which always contains recent N transactions. BFI-tree monitors the boundary movements to efficiently maintain the selected part of infrequent itemsets. If a node status changes, either from infrequent to frequent or vice versa, it must come through the boundary and result in boundary movements. Boundary movements may cause recursive updates, which will be restricted in a small sub-tree. It may also cause creating new nodes, which need an additional scan on all transactions in the sliding window to compute their frequencies. In order to return accurate frequent itemsets, all transactions in the sliding window must be maintained in a highly compact structure. However, the boundary is stable at most time, which means the update cost is very small. BFI-tree uses the Apriori property [Agrawal R. and Srikant R., 1994] in construction and updates to prune infrequent nodes. So, 1) all the children of an infrequent node should be pruned and consequently all infrequent nodes are leaves in BFI-tree, 2) some children of a frequent node may be infrequent and should be pruned.

In [Ren J. D. and Li K., 2008], an algorithm called MRFI-SW is presented to mine Recent FIs with SW, which uses a transaction-sensitive sliding window. The transactions are denoted with a special representation, which can denote the number and the order of items that are contained in the transactions. Using Apriori property, frequent itemsets can be mined through processing the representation's information. In this representation, for each itemset X which is contained in transactions in current sliding window is constructed as a sequence. The length of the sequence is w, where w is the number of transactions in transaction-sensitive sliding window. Each entry is the form of (bit, order), denoted as R(x). If item X is in the i-th transaction in current sliding window, the i-th entry of R(X)_bit is set to be 1 and the order of items in a transaction can get from R(X)_order, otherwise the R(X) is set to be 0 (R(X)_bit=R(X)_order=0). The process of creating this sequence of items for transaction in current sliding window is called bit-order transform. For example, let <T1, (acd)>, <T2, (bce)>, <T3, (abce)>, and <T4, (be)>, and the size of sliding window be 3, hence, SW1=[T1, T2, T3] and SW2=[T2, T3, T4]. In SW1, because item **a** appears in T1 and T3 and is the first item in both transactions, so, R(**a**) is <(1, 1), 0, (1, 1)>. Similarly, R(**c**)=<(1, 2), (1, 2), (1, 3)>, R(**d**)=<(1, 3), 0, 0>, R(**b**)=<0, (1, 1), (1, 2)>, and R(**e**)=<0, (1, 3), (1, 4)>.

In [Naganth E.R. and Dhanaseelan F. R., 2008] a *graph structure* is proposed to capture the contents of transactions in a sliding window. The graph structure captures the contents of transactions in each batch of streaming data. Transaction items are arranged according to some canonical order, which can be specified by the user prior to the graph construction or the mining process. Whenever a new batch of transactions flows in, it appends to this list at each node its frequency count in the current batch. In other words, the last entry of the list at node X shows the frequency count of X in the current batch. When the next batch of transactions comes in, the list is shifted forward. The last entry shifts and becomes the second – last entry; this leaves room for the newest batch. It uses a pointer to indicate the last update at each node. If the pointer points to the previous entry in the list of frequency counts at a node X, then this indicates that X has just been visited at the update of the last batch. On the other hand, if the pointer points to a much earlier entry in the list at a node Y, then this indicates that Y has not been visited since then and that the frequent counts of Y for the entries in between should be 0s. Since this graph structure is constructed independent of the minimum support threshold, every transaction in the current window is captured. Once such a tree is constructed, we can mine frequent itemsets from it in a fashion similar to FP-growth [Han J. et al, 2000] using the user supplied σ. Since items are consistently arranged according to some canonical order, one can guarantee the inclusion of all frequent items using just upward traversals, leading to exact mining results. There is also no worry about possible omission ordoubly counting of items during the mining process. So, mining is delayed until it is needed to avoid lots of unnecessary computation.

In [Pauray S. and Tsai M., 2009], an algorithm called WSW Algorithm is proposed. It is a weighted SW framework is proposed. See preliminaries of time-sensitive sliding window model in the description of [Lin C.H. et al, 2005]. In the traditional sliding window model, only one window is considered for mining at each time point. WSW proposed a flexible framework for continuous query processing in data streams. The time interval for periodical queries is defined to be the size of a window. In traditional sliding window model, the size of a window is usually defined to be a given number of transactions, say T. Although only the latest T transactions are considered, the time to cover these T transactions may be long. If we ignore the significance of data at different time intervals, the effectiveness of the mining result may decrease. So the WSW model has the following two new features: (1) window size is defined by time, not the number of transactions, the purpose is to avoid the case where intervals that cover T transactions at different time points may vary dramatically. (2) number of windows considered for mining is specified by the user. Moreover, the user can assign different weights to different windows according to the importance of data in each section. For example, the data near to the current moment may be more influential in the mining, and could be given a higher weight. The weights of windows affect the determination of frequent items. Even if the total support count of an item is large, if its support count in the window with a high weight is very low, it may not become a frequent item. Thus the consideration of weights for windows is reasonable and significant. So, the mining result would be closer to user's requirements.

By data characteristics, an improved algorithm, WSW-Imp, is explored to further reduce the time of deciding whether a candidate itemset is frequent or not. Experimental results show that the performance of WSW-Imp significantly outperforms that of WSW over weighted sliding windows.

In [Li H. F. and Lee S. Y., 2009], an algorithm called MFI-Time-SW is proposed. It works in a SW environment with a time-sensitive SW. See preliminaries of time-sensitive sliding window model in the description of [Lin C.H. et al, 2005]. The major differences between MFI-TransSW []Li H. F. et al, 2006] and [Li H. F. and Lee S. Y., 2009]and MFI-TimeSW are the following: 1) *Unit of data processing:* each time unit contains variable number of transactions. 2) *Bit-sequence transformation of a time unit:* For each item X in the current time-sensitive stream sliding window $TimeSW_{N\_w+1}$, a bitsequence with $|TimeSW_{N\_w+1}|$ bits, denoted as $Bit\ (X)_{N-w+1}^{TimeSW}$, is constructed. Similarly, if the item X is in the i-th transaction of $TimeSW_{N\_w+1}$, the ith bit of $Bit\ (X)_{N-w+1}^{TimeSW}$, is set to be 1. Otherwise, it is set to be 0. 3) *Number of sliding transactions:* In the window sliding phase of MFI-TimeSW algorithm, after the oldest time unit TU_{N_w+1} is removed from the current sliding window, a new time unit TU_{N+1} is appended to the window. If the aged time unit TU_{N_w+1} contains d transactions, MFI-TimeSW performs d times of bitwise left shift operation on the current sliding window. After that, MFI-TimeSW uses that same pruning method Item-Prune to improve the memory usage in mining process. 4) *Dynamic frequent threshold of itemsets:* the value of frequent threshold is s * |TimeSW| is a dynamic value, where $|TimeSW| = |TU_{N\_w+1}| + |TU_{N\_w+2}| + \ldots + |TU_N|$.

The sliding window model captures recent pattern changes and trends, by utilizing only the latest transactions for mining. As in certain applications, users can only be interested in the data recently arriving within a fixed time period. Sliding window model ignores the important fact of that Itemsets are changing their frequencies according to certain time intervals. In other words the sliding window model ignores the history of the itemsets' frequencies.

IV. Storage, Time And Accuracy Tradeoff

TABLE 1 presents the important comparative parameters to distinguish among the state of the art algorithms which are as the following: *1) The mining model, 2) The accuracy of the results, 3) The processing strategy, 4) The units of processing and 5) The data to be processed.*

According to Table 1, the first and last parameters are closely related, at which the landmark and fading models process the whole data stream from the system start. On the other hand, the sliding window model processes only a recent portion of the data stream.

All the algorithms of the landmark and fading models using the count based windows, except the algorithm of [Giannella, J. et al, 2003], which uses the time based window. In the Sliding window model, all the algorithms using the count base window, except the algorithms of [Lin C.H. et al, 2005], [Cheng J. et al, 2006], [Pauray S. and Tsai M., 2009] and part of [Li H. F. and Lee S. Y., 2009], which considered a fixed time period as the basic unit for processing (i.e time based window).

The count based windows are easy for programmers to deal with and not easy for people to specify. By contrast, in time based windows, it is natural for people to specify a time period as the basic unit, but it is more difficult to deal with windows with variable sizes in terms of bytes. In the performance perspective, no difference between using the time based or the count based windows [Arasu A. and Widom J., 2003].

The algorithms of [Manku G., and Motwani R., 2002], [Jin R. and Agrawal. G., 2005], [Chang J.H. and Lee WS., 2003], [Chang J.H and Lee WS., 2003], [Chang JH. and Lee WS., 2004], [Kun Li. et al, 2008], [Ren J. D. and Li K., 2008] and [Pauray S. and Tsai M., 2009] are using a tuple processing mechanism, at which processing is done transaction by transaction; the rest of the algorithms are

TABLE I. COMPARATIVE SUMMARY AMONG THE SATE-OF-THE-ART ALGORITHMS

Data Stream Mining Model	Data Stream Mining Algorithm	Approximate/Exact Results	Tuple/ Batch Processing	Time/ Count base (transactions)	All / Recent Transactions
Landmark Model	Lossy Counting [Manku G.,Motwani R., 2002]	Approximate/ False Positives (FP)	Tuple	Count	All
	FDPM [Yu J. X. et al,2004]	Approx./False Negatives (FN)	Batch	Count	All
	StreamMining [Jin R. ,Agrawal. G.,2005]	Approx. / FP	Tuple	Count	All
	FP-DS [Liu X. et al,2005]	Approx. / FP	Batch	Count	All
Fading Model	estDec [Chang J.H.,Lee WS.,2003]	Approx./ FP	Tuple	Count	All (recent is more important)
	FP-streaming [Giannella, J. et al,2003]	Approx. / FP	Batch	Time	All (recent is more important)
Sliding Window Model (SW)	estWin [Chang J.H,Lee WS.,2003]	Approx. / FP	Tuple	Count	Recent
	estWin_Lossy_Counting based [Chang JH.,Lee WS.,2004]	Approx. / FP	Tuple	Count	Recent
	[Lin C.H. et al,2005]	Approx. / FP or FN	Batch	Time	Recent
	MFI-TransSW [Li H. F.et al,2006],[Li H. F.,Lee S. Y.,2009]	Approx. / FP	Batch	Count	Recent
	Mine-SW [Cheng J et al,2006]	Approx. / FN	Batch	Time	Recent
	BFI Steam [Kun Li. Et al,2008]	Exact	Tuple	Count	Recent
	MRFI-SW [Ren J. D.,Li K.,2008]	Exact	Tuple	Count	Recent
	Graph Structure _SW [Naganth E.R.,Dhanaseelan F. R.,2008]	Exact	Batch	Count	Recent
	WSW [Pauray S.,Tsai M.,2009]	Approx. / FP	Tuple	Time	Recent
	MFI-Time-SW [Li H. F.,Lee S. Y.,2009]	Approx. / FP	Batch	Time	Recent

Using batch processing. Processing each transaction against the entire stream in most cases is less efficient than processing a batch of transactions against the entire stream. In general, batch processing is more suitable for high speed data streams [Cheng J. et al, 2008]. See Figure 4.

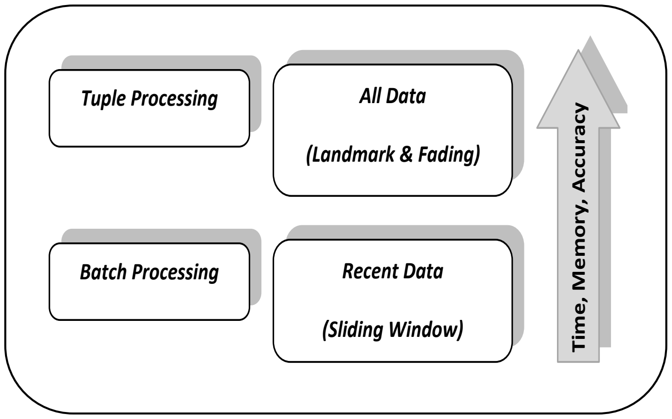

Fig.4. Time, memory and accuracy tradeoff

[Chang J.H. and Lee WS., 2003] favors recent itemsets by diminishing exponentially the effect of old transactions; but estimating itemsets' frequencies from their subsets leading to a propagated error. And [Chang J.H. and Lee WS., 2003] partitions the window according to a logarithmic scale with the recent frequency of an itemset recorded at a finer granularity using a tilted-time window, which answers time-sensitive queries at the expense of more than one record kept for each itemset leading to a very large structure over time which in turn degrade the mining process. All the variations of the landmark and fading models have the limitation of providing approximate answers for long-term data and adjust their storage requirement based on the available space.

All the algorithms are producing approximate results, except [Kun Li. et al, 2008], [Ren J. D. and Li K., 2008] and [Naganth E.R. and Dhanaseelan F. R., 2008], which produce exact results but for only a recent portion of the data stream (i.e they are under the sliding window model). All the approximate algorithms adopt false-positive approaches, except [Yu J. X. et al, 2004] and an option in [Lin C.H. et al, 2005], which adopts a false-negative approach. The false-positive approach uses a relaxed minimum support threshold, ϵ, to reduce the number of false-positives, so obtaining a more accurate result. However, to obtain a more accurate result, a smaller value of ϵ is to be set, leading to a larger set of sub-FIs to be maintained, consuming large amount of memory. The false-negative approach also uses a relaxed minimum support threshold, ϵ; however, its' use lowers the probability of discarding an infrequent itemset that may become frequent later. The error bound in the computed support and the possible false mining results of *most* of the false-positive approaches are implied by the following equation according the derivation of [Cheng J. et al, 2008]:

Support error bound = err $(X)/N_\tau$,

False results = $\{X \mid \text{freq}(X) < \sigma N \leq (\overline{\text{freq}}\,(X) + \text{err}(X))\}$

Almost all the online mining algorithms do the mining process during the entering of the stream, by constructing, filling and extracting FIs in parallel to the data stream entering; except [Li H. F. et al, 2006] and [Li H. F. and Lee S. Y., 2009] and [Naganth E.R. and Dhanaseelan F. R., 2008], which actually extract FIs from the filled data structure (i.e done during the data stream entering) only when it is requested by the user; which is a more effective strategy.

Any data stream mining algorithm aims to enforce correct and accurate results, minimized consumption of memory, fully utilized CPU and minimized time for processing. Correctness here refers to mining only true frequent itemsets (i.e no false positives, nor false negatives), and accuracy refers to mining exact or approximated frequencies for the itemsets (leading to true or nearest to true frequent itemsets).

Exact mining requires keeping track of all itemsets in the window and their actual frequency, because any infrequent itemset may become frequent later in the stream. However, the number of all itemsets is $O(2^{|I|})$ (that is, given a set of items I, the possible number of itemsets can be up to $2^{|I|} - 1$) making exact mining computationally difficult, in terms of both CPU and memory.

Also, using a relaxed minimum support threshold,ϵ, to control the quality of the approximation of the mining result leads to a dilemma. The smaller the value of ϵ, the more accurate is the approximation but the greater is the number of sub-FIs generated, which requires both more memory space and more CPU processing power. However, if ϵ approaches σ, more false-positive answers will be included in the result, since all sub-FIs whose computed frequency is at least $(\sigma\text{-}\epsilon)N \approx 0$ are outputted while the computed frequency of the sub-FIs can be less than their actual frequency by as much as $\sigma.N$.

Almost all the approximation algorithms produce false-positive results, at which, the set of sub-FIs kept is often too large in order to obtain a highly accurate answer. Thus, throughput is decreased and memory consumption is increased due to the processing of a large number of sub-FIs. But for the approximation algorithms which produce false negative results [Yu J. X. et al, 2004], it infringes the correctness. Accordingly, there is a direct proportion between the accuracy and memory consumption; more accurate results needs more memory usage leading to increased processing time.

V. Conclusion

According to the continues high flow of data streams and the relatively limited resources of CPU and storage, the process of mining frequent itemsets is chained to mining approximated frequencies, even it has guarantee on error bounds. The accuracy of the resulting FIs directly proportional with the memory usage; high accuracy needs high memory usage.

Therefore the algorithms of mining FIs from data streams which are using the Landmark model, Fading model (at which all the data stream is mined) or the Sliding Window model (at which only recent data is mined); leading to approximated results certainly.

VI. Future Work

We can focus in future to get exact mining results with regard to the available storage. Traditional data mining algorithms do not produce any results that show the change of the results over time. Dynamics of data streams using changes in the knowledge structures generated would benefit many temporal-based analysis applications.

References

[1] Agrawal R. and Srikant R. (1994). Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499.

[2] Arasu A. and Widom J. (2003). Resource Sharing in Continuous Sliding-Window Aggregates. ACM SIGMOD Record, Volume 32 Issue 2.

[3] Babcock B., Babu S., Datar M., Motwani R., and Widom J. (June 2002) Models and issues in data stream systems. In PODS'02, Madison, WI.

[4] Cormode. G. (2007). Fundamentals of Analyzing and Mining Data Streams. WORKSHOP ON DATA STREAM ANALYSIS – San Leucio, Italy - March, 15-16.

[5] Chang J. and Lee W.(2004). Decaying Obsolete Information in Finding Recent Frequent Itemsets over Data Stream. IEICE Transaction on Information and Systems, Vol. E87-D, No. 6.

[6] Chen, Y., Dong, G.; Han, J.; Wah, B.W.; and Wang, J.(2002). Multidimensional regression analysis of time-series data streams. In Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), 323.334.

[7] Chang J.H. and Lee WS. (2003). Finding recent frequent itemsets adaptively over online data streams. In:Getoor L, Senator T, Domingos P, Faloutsos C (eds) Proceedings of the Ninth ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, pp 487–492.

[8] Cohen E. and Strauss M. (2003). Maintaining Time Decaying Stream Aggregates. Proc. of ACM PODS Symp.

[9] Chang J.H and Lee WS. (2003). estWin: adaptively monitoring the recent change of frequent itemsets over online data streams. In: Proceedings of the 2003 ACM CIKM international conference on information and knowledge management, New Orleans, Louisiana, USA, November 2003, pp 536–539.

[10] Chang JH. and Lee WS. (2004). A sliding window method for finding recently frequent itemsets over online data streams. J Inf Sci Eng 20(4):753–762.

[11] Cheng J, Ke Y. and Ng W. (2006). Maintaining frequent itemsets over high-speed data streams. In: Ng WK, Kitsuregawa M, Li J, Chang K (eds) Proceedings of the 10th Pacific-asia Conference on knowledge discovery and data mining, Singapore, April pp 462–467.

[12] Chi Y, Wang H, Yu P, Muntz R (2004). Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proceedings of the 4th IEEE international conference on data mining, Brighton, UK, , pp 59–66.

[13] Cheng J, Ke Y and Ng W. (2008). A survey on algorithms for mining frequent itemsets over data streams. Knowl Inf Syst, 16:1–27 DOI 10.1007/s10115-007-0092-4.

[14] Gaber M., Zaslavsky A. and Krishnaswamy Sh. (June 2005). Mining Data Streams: A Review. SIGMOD Record, Vol. 34, No. 2.

[15] Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. (2003). Mining Frequent Patterns in Data Streams at Multiple Time Granularities. in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT.

[16] Han J., Pei J. and Yin Y. (2000). Mining frequent patterns without candidate generation. In Proceedings of ACM SIGMOD, 1–12.

[17] Hidber C. (1999). Online association rule mining. In: Delis A, Faloutsos C, Ghandeharizadeh S (eds) Proceedings of the ACM SIGMOD international conference on management of data, Philadelphia, Pennsylvania, pp 145–156.

[18] Jin R. and Agrawal. G. (2005). An algorithm for in-core frequent itemset mining on streaming data. To appear in ICDM'05.

[19] [19] Karp R.M., Shenker S., and Papadimitriou C.H. (2003). A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS), 28(1):51–55.

[20] Kun Li., Yong-yan Wang, Manzoor Ellahi, Hong-an Wang. (2008). Mining Recent Frequent Itemsets in Data Streams. IEEE 2008.

[21] Li H. F., Chin-Chuan Ho, Man-Kwan Shan, and Suh-Yin Lee (2006). Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window. In IEEE SMC.

[22] Li H. F. and Lee S. Y. (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. Sceince Direct, Expert Systems with Applications 36 (2009) 1466–1477.

[23] Lin C.H., Chiu D.Y., Wu Y.H. and Chen A.L.P. (2005). Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In Proceedings of 2005 SIAM International Conference on Data Mining.

[24] Liu X., Xu H., Dong Y., Wang Y. and Qian J.(2005). Dynamically Mining Frequent Patterns over Online Data Streams. Springer-Verlag Berlin Heidelberg. Y. Pan et al. (Eds.): ISPA 2005, LNCS 3758, pp. 645 – 654.

[25] Manku G., and Motwani R(2002). Approximate frequency counts over data streams. In Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), 346.357.

[26] Naganth E.R. and Dhanaseelan F. R.(2008). Efficient Graph Structure for the Mining of Frequent Itemsets from data Streams. IJCSES International Journal of Computer Sciences and Engineering Systems, Vol.1, No.4.

[27] Pauray S. and Tsai M. (2009). Mining frequent itemsets in data streams using the weighted sliding window model. Elsevier, Expert Systems with Applications.

[28] Ren J. D. and Li K. (2008). Online data Stream mining of recent frequent Itemsets Based On Sliding Window Model. Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming, 12-15.

[29] Silvestri C. (2006). Distributed and Stream Data Mining Algorithms for Frequent Pattern Discovery. Universit`a Ca' Foscari di Venezia, Dipartimento di Informatica, Dottorato di Ricerca in Informatica. Ph.D. Thesis: TD-2006-4.

[30] Yu J. X., Chong Z., Lu H., and Zhou A.(2004). False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In Proceedings of the 30th International Conference on Very Large Data Bases, pp. 204-215.Zhu Y. and Shasha D. (2002). StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 358-369.