Algorithms for Database Systems.
Efficient Episode Mining of Dynamic Event Streams.

Álvaro Marco Añó
malvaro@student.ethz.ch

March 9, 2015
Contents

1 Introduction .......................................................... 3

2 Problem Statement .................................................. 3
  2.1 Example 1 .......................................................... 4

3 Persistence and top-
  k approximation ................................................. 5
  3.1 Maximum rate of change ........................................ 5
  3.2 Top-k separation of \((\phi, \epsilon)\) .................................. 5
  3.3 Theorem 1 and first algorithm .................................. 6
  3.4 Theorem 2 .......................................................... 6
  3.5 Theorem 3: Quality of approximation of top-
  k with \((v, k)\)-
  persistent episodes .............................................. 7

4 Algorithm ............................................................ 7
  4.1 Example 2 .......................................................... 10

5 Conclusion ............................................................ 12
1 Introduction

In the last years advances in technology allowed a huge increase in the amount of data generated by some applications. For instance in the contexts of neuroscience, telecommunications and recommender systems; applications can produce hundreds or thousands of gigabytes per day. This led to an increasing interest in fields like Data Mining, Big Data and Machine Learning.

In this document a streaming algorithm for mining dynamic events is introduced. A streaming algorithm is an algorithm that processes data streams: data arrives sequentially and has to be processed immediately. Typically only one pass over the data is required, the amount of memory used by the algorithm is very restricted and there are possibly multiple data streams.

2 Problem Statement

An episode is a collection of events that occur relatively close to each other in a given partial order. An $\ell$-node episode is a sequence of episodes of length $\ell$. The task in top-$k$ episode mining is to find the $k$ most frequent episodes.

The event stream is in the form of a potentially infinite sequence of events:

$$D = \langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_i, \tau_i), \ldots, (e_n, \tau_n), \ldots \rangle$$  \hspace{1cm} (1)

where $(e_i, \tau_i)$ represents the $i^{th}$ event; $e_i$ is drawn from a finite alphabet $\epsilon$ of symbols (called event-types) and $\tau_i$ denotes the time-stamp of the $i^{th}$ event.

Our goal is to find all episodes that were frequent in the recent past; for this we consider a sliding window model. In this model, the user wants to determine episodes that were frequent over a window of fixed-size. As new events arrive in the stream, the user’s window of interest shifts, and the data mining task is to report the frequent episodes in the new window of interest.

We consider the case where the window of interest is very large and cannot be stored and processed in-memory. The events in the stream are grouped into batches such that at any given time only the latest incoming batch is stored and processed in memory. This is illustrated by figure 1. The frequency of an episode $\alpha$ in a batch $B_s$ is referred to as its batch frequency $f^*(\alpha)$. The current window of interest, $W_s$, consists of $m$ consecutive batches ending in batch $B_s$:

$$W_s = \langle B_{s-m+1}, B_{s-m+2}, \ldots, B_s \rangle$$  \hspace{1cm} (2)
Figure 1: A sliding window model for episode mining over event streams: $B_s$ is the most recent batch of events that arrived in the stream and $W_s$ is the window of interest over which the user wants to determine the set of frequent episodes.

The frequency of an episode $\alpha$ over window $W_s$, referred to as its *window frequency* is defined as the sum of batch frequencies of $\alpha$ in $W_s$:

$$f_{W_s}(\alpha) = \sum_{B_j \in W_s} f_j(\alpha)$$ (3)

In summary, we are given an event stream $(\mathcal{D})$, a time-span for batches, the number of consecutive batches that constitute the current window of interest ($m$), the desired size of frequent episodes ($\ell$), the desired number of most frequent episodes ($k$) and the problem is to discover the top-$k$ episodes in the current window without actually having the entire window in memory.

### 2.1 Example 1

The following example will help to understand the problem. Let $W$ be a window of four batches, $B_1$ through $B_4$. The episodes in each batch with corresponding batch frequencies are listed in figure 2. The corresponding window frequencies (sum of each episodes’ batch frequencies) are listed in table 1.

![Batch frequencies for the example.](image)

The top-2 episodes in $B_1$ are (PQRS) and (WXYZ). Similarly (EFGH) and (IJKL) are the top-2 episodes in $B_2$, and so on. (ABCD) and (MNOP)
have the highest window frequencies but never appear in the top-2 of any batch – these episodes would go undetected if we considered only the top-2 episodes in every batch as candidates for the top-2 episodes over $W$. This example can be easily generalized to any number of batches and any $k$.

The example highlights the main challenge in the streaming top-$k$ mining problem. We can only store the most recent batch of events and the batchwise top-$k$ may not contain sufficient information to compute the top-$k$ over the entire window. It is obviously not possible to track all episodes since the pattern space is typically very large. This brings us to the question of which episodes to track in every batch.

3 Persistence and top-$k$ approximation

In this section some important properties of the underlying event stream are presented.

3.1 Maximum rate of change

The first property is the Maximum rate of change $\Delta(>0)$, it is defined as the maximum change in batch frequency of an episode $\alpha$ across any pair of consecutive batches $B_s$ and $B_{s+1}$. $\forall \alpha, s$ we have

$$|f_{s+1}^\alpha(\alpha) - f_s^\alpha(\alpha)| \leq \Delta \quad (4)$$

Intuitively, $\Delta$ controls the extent of change from one batch to the next. While it is trivially bounded by the number of events arriving per batch, it is often much smaller in practice.

Therefore, the batch frequencies of the $k^{th}$ most-frequent episodes in any pair of consecutive batches cannot differ by more than the maximum rate of change $\Delta$, i.e., for every $B_s$ we must have

$$|f_{s+1}^k(\alpha) - f_s^k(\alpha)| \leq \Delta \quad (5)$$

3.2 Top-k separation of $(\varphi, \epsilon)$

A batch $B_s$ has a top-$k$ separation of $(\varphi, \epsilon)$, $\varphi \geq 0$, $\epsilon \geq 0$, if it contains at most $(1 + \epsilon)k$ episodes with batch frequencies of $(f_k^s - \varphi \Delta)$ or more, where
\( f_s^k \) is the batch frequency of the \( k^{th} \) most-frequent episode in \( B_s \) and \( \Delta \) is the maximum rate of change.

This is a measure of how well-separated the frequencies of the top-\( k \) episodes are relative to the rest of episodes. The separation is considered to be high if lowering the threshold from \( f_s^k \) to \( (f_s^k - \varphi \Delta) \) only brings in very few additional episodes, i.e. \( \epsilon \) remains small as \( \varphi \) increases.

### 3.3 Theorem 1 and first algorithm

**Theorem 1 (Exact Top-\( k \) over \( W_s \))**:

An episode \( \alpha \), can be a top-\( k \) episode over window \( W_s \) only if its batch frequencies satisfy:

\[
f_s^\prime(\alpha) \geq (f_s^\prime k - 2(m - 1)\Delta) \quad \forall B_{s'} \in W_s \tag{6}
\]

Based on Theorem 1, the following algorithm obtains the top-\( k \) episodes over a window: use a level-wise approach to find all episodes with a batch frequency of at least \( (f_1^k - 2(m - 1)\Delta) \) in the first batch \( (B_1) \), accumulate their corresponding batch frequencies over all \( m \) batches of \( W_s \) and report the episodes with the \( k \) highest window frequencies over \( W_s \). This approach is guaranteed to return the exact top-\( k \) episodes over \( W_s \). To obtain the top-\( k \) episodes over \( W_{s+1} \) we need to consider all episodes with batch frequency of at least \( (f_2^k - 2(m - 1)\Delta) \) in the second batch and track them over all batches of \( W_{s+1} \), and so on. Thus, an exact solution to the problem would require running a level-wise episode mining in every batch \( B_s \) with a frequency threshold of \( (f_s^k - 2(m - 1)\Delta) \).

**Theorem 1** characterizes the minimum batchwise computation needed in order to obtain the exact top-\( k \) episodes over a sliding window. However, when the frequencies become low the number of episodes to track over the window becomes impractically high.

### 3.4 Theorem 2

To address the mentioned problem let us introduce a new class of episodes. A \((v,k)-\text{Persistent Episode}\) is an episode over window \( W_s \) which is a top-\( k \) episode in at least \( v \) batches of \( W_s \).

**Theorem 2**: An episode \( \alpha \) can be \((v,k)\)-persistent over the window \( W_s \) only if its batch frequencies satisfy:

\[
f_s^\prime(\alpha) \geq (f_s^\prime k - 2(m - v)\Delta) \quad \forall B_{s'} \in W_s \tag{7}
\]

For discovering \((v,k)\)-persistent episodes we can take profit of Theorem 2 and reuse the algorithm described for extracting the exact top-\( k \) episodes only applying a higher batchwise threshold \( (f_s^\prime k - 2(m - v)\Delta) \). More details on the algorithm are provided in section 4.
3.5 Theorem 3: Quality of approximation of top-k with \((v, k)\)-persistent episodes

The idea here is that, under a maximum rate of change \(\Delta\) and a top-k separation of \((\phi, \epsilon)\), there cannot be too many distinct episodes which are not \((v, k)\)-persistent while still having sufficiently high window frequencies.

A lower-bound \((f_L)\) on the window frequencies of \((v, k)\)-persistent episodes is defined as:

\[
f^{W_s}(\alpha) \geq \sum_{B_{s'}} f^s_{k'} - (m - v)(m - v + 1)\Delta \quad \text{def} = f_L \tag{8}
\]

And an upper-bound \((f_U)\) on the window frequencies of episodes that are not \((v, k)\)-persistent is defined as:

\[
f^{W_s}(\beta) < \sum_{B_{s'}} f^s_{k'} + v(v + 1)\Delta \quad \text{def} = f_U \tag{9}
\]

If episode \(\beta\) is not \((v, k)\)-persistent over a window \(W_s\), then its window frequency \(f^{W_s}(\beta)\) must satisfy \(f_U\).

It turns out that \(f_U > f_L \forall v, 1 \leq v \leq m\), and hence there is always a possibility for some episodes which are not \((v, k)\)-persistent to end up with higher window frequencies than a \((v, k)\)-persistent episode. We observed a specific instance of this kind of “mixing” in Example 1. This brings us to the top-k separation property. Intuitively, if there is sufficient separation of the top-k episodes from the rest of the episodes in every batch, then we would expect to see very little mixing. This separation need not to occur exactly at \(k^{th}\) most-frequent episode in every batch, somewhere close to it is sufficient to achieve a good top-k approximation.

**Theorem 3 (Quality of Top-k Approximation):** Let every batch \(B_{s'} \in W_s\) have a top-k separation of \((\phi, \epsilon)\) with \(\frac{\phi}{2} > max\{1, (1 - \frac{v}{m})(m - v + 1)\}\). Let \(\mathcal{P}\) denote the set of all \((v, k)\)-persistent episodes over \(W_s\). If \(|\mathcal{P}| \geq k\), then the top-k episodes over \(W_s\) can be determined from \(\mathcal{P}\) with an error of no more than \((\frac{w_{km}}{m})\) episodes, where \(\mu = min\{m - v + 1, \frac{v}{2}, \frac{1}{2}(\sqrt{1 + 2m\phi} - 1)\}\).

This last theorem brings out the tension between the persistence parameter \(v\) and the quality of approximation: at smaller values of \(v\) at smaller values of \(v\) the algorithm mines “deeper” within each batch and so we expect fewer errors with respect to the true top-k episodes. On the other hand, deeper mining within batches is computationally more intensive, with the required effort approaching that of exact top-k mining as \(v\) approaches 1.

4 Algorithm

In this section we present an efficient algorithm for incrementally mining episodes with frequency at least \((f^s_k - \theta)\) in batch \(B_s\), for each batch in the
stream.

A trivial (brute-force) algorithm to find all episodes with frequency greater than \( (f_k^s - \Theta) \) in \( B_s \) is as follows: Apply an Apriori-based episode mining algorithm on batch \( B_s \). If the number of episodes of size-\( \ell \) is less than \( k \), the support threshold is decreased and the mining repeated until at least \( k\ell \)-size episodes are found. At this point \( f_k^s \) is known. The mining process is then repeated once more with the frequency threshold \( (f_k^s - \Theta) \). Doing this entire procedure for every new batch is expensive and wasteful. After seeing the first batch of the data, whenever a new batch arrives we have information about the episodes that were frequent in the previous batch. This can be exploited to incrementally and efficiently update the set of frequent episodes in the new batch. The intuition is that the frequencies of most episodes do not change much from one batch to the next. As a result only a small number of previously frequent episodes fall below the new support threshold in the new batch; similarly, some new episodes may become frequent. This is illustrated in figure 3.

![Diagram of frequent episodes](image)

**Figure 3:** The set of frequent episodes can be incrementally updated as new batches arrive.

Frequent episodes are discovered level-wise, in ascending order of episode-size. An Apriori procedure alternates between counting and candidate generation. First a set \( C^i \) of candidate \( i \)-size episodes is determined by combining frequent \((i - 1)\)-size episodes from the previous level. Then the data is scanned for determining frequencies of the candidates, from which the frequent \( i \)-size episodes are obtained. We note that all candidate episodes that are not frequent constitute the negative border of the frequent lattice. This is because, a candidate is generated only when all its subepisodes are frequent.

The pseudocode for incrementally mining frequent episodes in batches is listed in *Algorithm 1*. For the first batch \( B_1 \) (lines 1–3) the top-\( k \) episodes
Algorithm 1: Persistent episode miner: Mine top-\(k\) \(v\)-persistent episodes.

**Input**: Number \(k\) of top episodes; New batch \(B_s\) of events; Current lattice of frequent & border episodes \((F^{s-1}_s, B^{s-1}_s)\); Threshold parameter \(\theta\) set to \(2(m - v)\Delta\) for \((v,k)\)-persistence (see Theorem 2), \(2(m - 1)\Delta\) for exact top-\(k\) (see Theorem 1)

**Output**: Lattice of frequent and border episodes \((F^*_s, B^*_s)\) after \(B_s\)

1. if \(s = 1\) then
   2. Determine \(f_k^1\) (brute force)
   3. Compute \((F^1_s, B^1_s) \leftarrow \text{Apriori} (B_1, \ell, f_k^1 - \theta)\)
4. else
   5. CountFrequency \((F^{\ell-1}_s, B_s)\)
   6. Determine \(f_k^s\) (based on episodes in \(F^{\ell-1}_s\))
   7. Initialize \(C^1 \leftarrow \phi\) (new candidates of size 1)
   8. for \(i \leftarrow 1, \ldots, \ell\) do
      9. Initialize \(F^i_s \leftarrow \phi\), \(B^i_s \leftarrow \phi\)
     10. Initialize \(F^i_{\text{new}} \leftarrow \phi\) (new frequent episodes of size \(i\))
     11. CountFrequency \((F^{i-1}_s \cup B^{i-1}_s \cup C^i, B_s)\)
     12. for \(\alpha \in F^{i-1}_s\) do
         13. if \(f_s^i(\alpha) \geq f_k^s - \theta\) then
             14. Update \(F^i_s \leftarrow F^i_s \cup \{\alpha\}\)
         15. else
             16. Update \(B^i_s \leftarrow B^i_s \cup \{\alpha\}\)
             17. Delete super-episodes of \(\alpha\) from \((F^{i-1}_s, B^{i-1}_s)\)
         end
     end
     19. for \(\alpha \in B^{i-1}_s \cup C^i\) do
         20. if \(f_s^i(\alpha) \geq f_k^s - \theta\) then
             21. Update \(F^i_s \leftarrow F^i_s \cup \alpha\)
             22. Update \(F^i_{\text{new}} \leftarrow F^i_{\text{new}} \cup \alpha\)
         23. else
             24. Update \(B^i_s \leftarrow B^i_s \cup \alpha\)
         end
     end
     27. \(C^{i+1} \leftarrow \text{GenerateCandidate} (F^i_{\text{new}}, F^i_s)\)
   end
30. return \((F^*_s, B^*_s)\)
are found by a brute-force method. For subsequent batches, we do not need a brute-force method to determine $f_k^s$. If $\theta \geq 2\Delta$, $F_{s-1}^\ell$ from batch $B_{s-1}$ contains every potential top-$k$ episode of the next batch $B_s$. Therefore simply updating the counts of all episodes in $F_{s-1}^\ell$ in the new batch $B_s$ and picking the $k^{th}$ highest frequency gives $f_k^s$ (lines 4–6). To update the lattice of frequent and border episodes (lines 7–30) the procedure starts from the bottom (size-1 episodes). The data is scanned to determine the frequency of new candidates together with the frequent and border episodes from the lattice (line 11). In the first level (episodes of size 1), the candidate set is empty. After counting, the episodes from $F_{s-1}^\ell$ that continue to be frequent in the new batch are added to $F_{s-1}^\ell$ (lines 13–14). But if an episode is no longer frequent it is marked as a border set and all its super episodes are deleted (lines 20–23). This ensures that only border episodes are retained in the lattice. In the border and new candidate sets, any episode that is found to be frequent is added to both $F_s^i$ and $F_{new}^i$ (lines 18–21). The remaining infrequent episodes belong to the border because, otherwise, they would have at least one infrequent subepisode and would have been deleted at a previous level; hence, these infrequent episodes are added to $B_s^\ell$ (lines 24–26).

Finally, the candidate generation step (line 28) is required to fill out the missing parts of the frequent lattice. We want to avoid a full blown candidate generation. Note that if an episode is frequent in $B_{s-1}$ and $B_s$, then all its subepisodes are also frequent in both $B_s$ and $B_{s-1}$. Any new episode ($\notin F_{s-1}^\ell \cup B_{s-1}^\ell$) that turns frequent in $B_s$, therefore, must have at least one subepisode that was not frequent in $B_{s-1}$ but is frequent in $B_s$. All such episodes are listed in $F_{new}^i$. Hence the candidate generation step (line 24) for the next level generates only candidates with at least one subepisode $\in F_{new}^i$. This reduces the number of candidates generated at each level without compromising completeness of the results.

For a window $W_s$ ending in the batch $B_s$, the set of output episodes can be obtained by picking the top-$k$ most frequent episodes from the set $F_s^\ell$.

### 4.1 Example 2

In this example we illustrate the procedure for incrementally updating the frequent episodes lattice as a new batch $B_s$ is processed (see figure 4).

Figure 4(A) shows the lattice of frequent and border episodes found in the batch $B_{s-1}$. $ABCD$ is a 4-size frequent episode in the lattice. In the new batch $B_s$, the episode $ABCD$ is no longer frequent. The episode $CDXY$ appears as a new frequent episode. The episode lattice in $B_s$ is shown in Figure 4(B).

In the new batch $B_s$, $AB$ falls out of the frequent set. $AB$ now becomes the new border and all its super-episodes namely $ABC$, $BCD$ and $ABCD$ are deleted from the lattice. At level 2, the border episode $XY$ turns frequent.
Figure 4: Incremental lattice update for the next batch $B_s$ given the lattice of frequent and border episodes in $B_{s-1}$. 
in \( B_s \). This allows us to generate \( DXY \) as a new 3-size candidate. At level 3, \( DXY \) is also found to be frequent and is combined with \( CDX \) which is also frequent in \( B_s \) to generate \( CDXY \) as a 4-size candidate. Finally at level 4, \( CDXY \) is found to be frequent. This shows that border sets can be used to fill out the parts of the episode lattice that become frequent in the new data.

5 Conclusion

The algorithm has been tested on synthetic and real data streams from the domains of experimental neuroscience and telecommunication networks. The telecommunication test is based on one of the real-life situations that the algorithm is aimed to. It analyzes network traffic to detect attacks or other malicious activity.

In all these experiments the quality of the results obtained was the best in comparison to other methods performing the same task. Moreover, both the runtime and the memory used by the introduced algorithm was always better or in the same order of magnitude as in the compared methods.