Mining frequent items in a stream using flexible windows

Mining Frequent Items in a Stream Using Flexible Windows (Extended Abstract)

Toon Calders, Nele Dexters and Bart Goethals

University of Antwerp, Belgium
firstname.lastname@ua.ac.be

Abstract. In this paper, we study the problem of finding frequent items in a continuous stream of items. A new frequency measure is introduced, based on a flexible window length. For a given item, its current frequency in the stream is defined as the maximal frequency over all windows from any point in the past until the current state. We study the properties of the new measure, and propose an incremental algorithm that allows to produce the current frequency of an item immediately at any time. It is shown experimentally that the memory requirements of the algorithm are extremely small for many different realistic data distributions.

1 Introduction

Mining frequent items over a stream of items presents interesting new challenges over traditional mining in static databases. It is assumed that the stream can only be scanned once, and hence if an item is passed, it can not be revisited, unless it is stored in main memory. Storing large parts of the stream, however, is not possible because the amount of data passing by is typically huge.

Most previous work on mining frequently occurring items from a stream either focusses on (1) the whole stream, (2) on only the most recent items in a window of fixed length [1, 4, 6], or (3) where a time-decaying factor fades out the history of the stream [5]. In many applications, however, it is not possible to fix a window length or a decay factor that is most appropriate for every item at every timepoint in an evolving stream. For example, consider a large retail chain of which sales can be considered as a stream. Then, in order to do market basket analysis, it is very difficult to choose in which period of the collected data you are particularly interested. For many products, the amount of them sold depends highly on the period of the year. In summer time, e.g., sales of ice cream increase. During the world cup, sales of beer increase. Such seasonal behavior of a specific item can only be discovered when choosing the correct window size for that item, but this size can then also hide a similar behavior of other items. Therefore, we propose to consider for each item the window in which it has the highest frequency. More specifically, we define the current frequency of an item as the maximum over all windows from the past until the current state. The disadvantage of having a frequency of 100%, when the stream ends with the particular item, can be resolved by setting a minimum window length all windows have to obey. Hence, when the stream evolves, the length of the window containing the highest frequency for a given item can grow and shrink continuously. We show some important properties on how the length of the maximal window can evolve.

In our approach, on every timestamp, a new item arrives in the stream. We present an incremental algorithm that maintains a small summary of relevant information of the history of the stream that allows to produce the current frequency of an item immediately at any time. That is, when a new item arrives, the summary is updated, and when at a certain point in time, the current frequency is required, the result can be obtained instantly from the summary. The structure of the summary is based on some critical observations about the windows with the maximal frequency. In short, many points in the stream can never become the starting point of a maximal window, no matter what the continuation of the stream will be. The summary will thus consist of some statistics about the few points in the stream that are still candidate starting points of a maximal window. These important points in the stream will be called the *borders*.

Critical for the usefulness of the technique are the memory requirements of the summary that needs to be maintained in memory. We show experimentally that, even though in worst case the summary depends on the length of the stream, for realistic data distributionsits size is extremely small. Obviously, this property is highly desirable as it allows for an efficient and effective computation of our new measure. Also note that our approach allows exact information as compared to many approximations considered in other works.

The organization of the paper is as follows. In Section 2, the new measure is defined and the problem is formally introduced. Section 3 gives the theoretical results for the basic theorem, on which the incremental algorithm in Section 4 is based. Section 5 contains experiments that show that the memory requirements for the algorithm are extremely small for many real-life data distributions.

2 New Frequency Measure in Stream Mining

In this section, we define our new frequency measure for streams and we formally introduce the problem. Throughout the paper, we assume that the items in the stream come from a finite set of items $\mathcal{I}$, unless explicitly mentioned otherwise.

2.1 Streams and Max-Frequency

A *stream* $\mathbb{S}$ is a sequence $\langle i_1, i_2, \ldots, i_n \rangle$ of items, where n is the *length* of the stream, denoted $|\mathbb{S}|$. The number of occurrences of an item i in the stream $\mathbb{S}$ will be denoted *count*$(i, \mathbb{S})$. The *frequency* of i in $\mathbb{S}$, is defined as

$$\mathit{freq}(i, \mathbb{S}) := \frac{\mathit{count}(i, \mathbb{S})}{|\mathbb{S}|} \quad .$$

The *concatenation* of m streams $\mathbb{S}_1, \ldots, \mathbb{S}_m$ is denoted $\mathbb{S}_1 \cdot \mathbb{S}_2 \cdot \ldots \cdot \mathbb{S}_m$. Every stream is glued at the end of the previous stream. Let $\mathbb{S} = \langle i_1, i_2, \ldots, i_n \rangle$. Then, $\mathbb{S}[s,t]$ denotes the *sub-stream* or *window* $\langle i_s, i_{s+1}, \ldots, i_t \rangle$. The sub-stream of $\mathbb{S}$ consisting of the last k items of $\mathbb{S}$, denoted $\mathit{last}(k, \mathbb{S})$, is

$$\mathit{last}(k, \mathbb{S}) = \mathbb{S}\big[|\mathbb{S}| - k + 1, |\mathbb{S}|\big] \ .$$

We are now ready to define our new frequency measure:

Definition 1. *The* max-frequency $\mathit{mfreq}(i, \mathbb{S})$ *of an item i in a stream $\mathbb{S}$ is defined as the maximum of the frequency of i over all windows extending from the end of the stream; that is:*

$$\mathit{mfreq}(i, \mathbb{S}) \ := \ \max_{k=1\ldots|\mathbb{S}|} \left(\mathit{freq}(i, \mathit{last}(k, \mathbb{S}))\right) \ .$$

This $\mathit{mfreq}(i, \mathbb{S})$ is used as a new frequency measure for stream mining. For a given item, its current frequency in the stream is defined as the maximal frequency over all evolving windows from the end to the beginning of the stream.

The longest window in which the max-support is reached, is called the maximal window for i in $\mathbb{S}$*, and its starting point is denoted $\mathit{maxwin}(i, \mathbb{S})$. That is, $\mathit{maxwin}(i, \mathbb{S})$ is the smallest index such that*

$$\mathit{mfreq}(i, \mathbb{S}) \ = \ \mathit{freq}(i, \mathbb{S}\big[\mathit{maxwin}(i, \mathbb{S}), |\mathbb{S}|\big]) \ . \qquad \square$$

Note that, by definition, the max-frequency of an item a in a stream that ends with an a, is always 100%, independently of the overall frequency of a. Hence, even in a stream where a is extremely rare, at some points, the max-frequency will be maximal! This disadvantage of max-frequency, however, can easily be resolved by either considering streams of blocks of items instead of single items, or by setting a minimal length all windows must obey. We did not include these solutions in the paper, as they are not fundamental for the theory developed.

Example 1. We focus on target item a.

$$\begin{aligned}
\mathit{mfreq}(a, abaaab) &= \max_{k=1\ldots6} \left(\mathit{freq}(a, \mathit{last}(k, abaaab))\right) \\
&= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = \frac{3}{4} \ . \\
\mathit{mfreq}(a, bcdabcda) &= \max\left(\frac{1}{1}, \cdots\right) = 1 \ . \\
\mathit{mfreq}(a, xaaxaax) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = \frac{2}{3} \ .
\end{aligned}$$

Notice that our definition of frequency is quite different from the usual approaches, where the frequency of an item i in a stream $\mathbb{S}$ is either defined as $\mathit{freq}(i, \mathit{last}(wl, \mathbb{S}))$ for a fixed window length wl, or with a time-decaying factor; e.g., $\mathit{freq}(i, \mathbb{S}) = \sum_{j=1}^{|\mathbb{S}|/wl} d^j \mathit{freq}(i, \mathbb{S}[(j-1)wl + 1, j\,wl])$, with $d < 1$.

2.2 Problem Statement

Notice that we defined a stream as a statical object. In reality, however, a stream is an evolving object. At every timepoint, a new item might be inserted at the end of the stream. As such, evolving streams are essentially unbounded, and when processing them, it is to be assumed that only a small partcan be kept in memory.

In our examples, new entrieswill be added on the right side of the stream. This means that the older items are on the left side of the stream. For simplicity we assume that the first item arrived at timestamp 1, and since then, at every timestamp a new item was inserted. $\mathbb{S}_t$ denotes the stream up to timestamp t.

The problem we study in the paper is the following: *For an evolving stream* $\mathbb{S}$ *and a fixed item* a*, maintain a small summary of the stream in time, such that, at any timepoint* t*,* $\mathit{mfreq}(a, \mathbb{S}_t)$ *can be produced instantly from the summary.*

More formally, we will introduce a summary of a stream $\mathit{summary}(\mathbb{S})$, an update procedure *Update*, and a procedure *Get_mfreq*, such that, if we assume that on timestamp $t+1$ item i is added to the stream, $\mathit{Update}(\mathit{summary}(\mathbb{S}_t), i)$ equals $\mathit{summary}(\mathbb{S}_t \cdot \langle i \rangle)$ equals $\mathit{summary}(\mathbb{S}_{t+1})$, and $\mathit{Get\_mfreq}(\mathit{summary}(\mathbb{S}_{t'}))$ equals $\mathit{mfreq}(a, \mathbb{S}_{t'})$. Because *Update* has to be executed every time a new item arrives, it has to be extremely efficient, in order to be finished before the next item arrives. Similarly, because the stream continuously grows, the summary must be independent of the number of items seen so far, or, at least grow very slowly as the stream evolves. The method we develop will indeed meet these criteria, as the experiments will show.

<table>
<tr><th>stream</th><th>time stamp</th><th>$\mathit{mfreq}(a, \mathbb{S}_t)$</th></tr>
<tr><td>|a</td><td>1</td><td>$\max(\frac{1}{1}) = 1$</td></tr>
<tr><td>|aa</td><td>2</td><td>$\max(\frac{1}{1}, \frac{2}{2}) = \frac{2}{2} = 1$</td></tr>
<tr><td>|aaa</td><td>3</td><td>$\max(\frac{1}{1}, \frac{2}{2}, \frac{3}{3}) = \frac{3}{3} = 1$</td></tr>
<tr><td>|aaab</td><td>4</td><td>$\max(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}) = \frac{3}{4}$</td></tr>
<tr><td>|aaabb</td><td>5</td><td>$\max(\frac{0}{1}, \frac{0}{2}, \frac{1}{3}, \frac{2}{4}, \frac{3}{5}) = \frac{3}{5}$</td></tr>
<tr><td>|aaabbb</td><td>6</td><td>$\max(\frac{0}{1}, \frac{0}{2}, \frac{0}{3}, \frac{1}{4}, \frac{2}{5}, \frac{3}{6}) = \frac{3}{6}$</td></tr>
<tr><td>aaabbb|a</td><td>7</td><td>$\max(\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}, \frac{3}{6}, \frac{4}{7}) = 1$</td></tr>
<tr><td>aaabbb|aa</td><td>8</td><td>$\max(\frac{1}{1}, \frac{2}{2}, \frac{2}{3}, \frac{2}{4}, \frac{2}{5}, \frac{3}{6}, \frac{4}{7}, \frac{5}{8}) = \frac{2}{2}$</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
</table>

Fig. 1. Max-frequency of a stream at every timepoint.

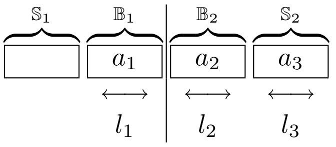
In Fig. 1, the max-frequency has been given for an example evolving stream. The starting point $\mathit{maxwin}(a, \mathbb{S})$ of each maximal window is marked with $|$.

3 Properties of Max-Frequency

In this section we show some properties of max-frequency that are crucial for the incremental algorithm that maintains the summary of the stream. Obviously, checking all possible windows to find the maximal one is infeasible algorithmically, given the constraints of stream problems. Fortunately, not every point in the stream needs to be checked. The theoretical results from this section show exactly which points need to be inspected. These points will be called the *borders* in the stream. The summary of the stream will consist exactly of the recording of these borders, and the frequency of the target item up to the most recent timepoint.

Theorem 1. *Consider a stream* $\mathbb{S}_1 \cdot \mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{S}_2$*. If* $\mathbb{B}_2 \cdot \mathbb{S}_2$ *is the maximal window for* a *in* $\mathbb{S}$*, then* $\mathit{freq}(a, \mathbb{B}_1) < \mathit{freq}(a, \mathbb{B}_2)$

Proof. If $\mathbb{B}_2 \cdot \mathbb{S}_2$ is the maximal window for a in $\mathbb{S}$, then this implies that the frequency of a in $\mathbb{B}_2 \cdot \mathbb{S}_2$ is strictly higher than in $\mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{S}_2$ and at least as high as in $\mathbb{S}_2$ (remember that in the case of multiple windows with maximal frequency, the largest one is selected). Let now $l_1 = |\mathbb{B}_1|$, $l_2 = |\mathbb{B}_2|$, and $l_3 = |\mathbb{S}_2|$, and let $a_1 = \mathit{count}(a, \mathbb{B}_1)$, $a_2 = \mathit{count}(a, \mathbb{B}_2)$, and $a_3 = \mathit{count}(a, \mathbb{S}_2)$, as depicted in:



.

Then, the conditions on the frequency translate into:

$$\frac{a_2 + a_3}{l_2 + l_3} > \frac{a_1 + a_2 + a_3}{l_1 + l_2 + l_3} \quad \text{and} \quad \frac{a_2 + a_3}{l_2 + l_3} \geq \frac{a_3}{l_3}.$$

From these conditions, it can be derived that

$$\mathit{freq}(a, \mathbb{B}_1) = \frac{a_1}{l_1} < \frac{a_2}{l_2} = \mathit{freq}(a, \mathbb{B}_2). \qquad \square$$

Based on this theorem, it is possible to give an exact characterization of which points in $\mathbb{S}_t$ can potentially become the starting point of the maximal window at a future point in time, after new items have been added. The next corollary gives this characterization.

We now formally define the important notion of a *border*. Intuitively, a border is a point in the stream that can still become the starting point of the maximal window.

Definition 2. *The position* q *in* $\mathbb{S}$ *is called a* border for a in $\mathbb{S}$ *if there exists another stream* $\mathbb{B}$ *such that* $q = \mathit{maxwin}(a, \mathbb{S} \cdot \mathbb{B})$.

Corollary 1. *Let* $\mathbb{S}$ *be a stream, and let* $q = 1$*. Position* q *is a border for item* a *if and only if the first item in the stream equals the target item* a*.*

Let $\mathbb{S}$ *be a stream, and let* $2 \leq q \leq |\mathbb{S}|$*. Position* q *is a border for item* a *in* $\mathbb{S}$ *if and only if for all indices* j, k *with* $1 \leq j < q$ *and* $q \leq k \leq |\mathbb{S}|$*, it holds that* $\mathit{freq}(a, \mathbb{S}[j, q-1]) < \mathit{freq}(a, \mathbb{S}[q, k])$*.*

Proof. **Only if:** Suppose that there exist indices j and k, and a stream $\mathbb{B}$ such that $\mathit{freq}(a, \mathbb{S}[j, q-1]) \geq \mathit{freq}(a, \mathbb{S}[q, k])$, and $q = \mathit{maxwin}(a, \mathbb{S} \cdot \mathbb{B})$. This situation is in contradiction with Theorem 1: split the stream $\mathbb{S} \cdot \mathbb{B}$ as $(\mathbb{S}[1, j-1]) \cdot (\mathbb{S}[j, q-1]) \cdot (\mathbb{S}[q, k]) \cdot (\mathbb{S}[k+1, |\mathbb{S}|] \cdot \mathbb{B})$. In this stream, $(\mathbb{S}[q, |\mathbb{S}|]) \cdot \mathbb{B}$ is the maximal window, while $\mathit{freq}(a, \mathbb{S}[j, q-1]) \geq \mathit{freq}(a, \mathbb{S}[q, k])$.

If: It can be shown that the item at timepoint q must be the target item a. If enough non-target items are added to the stream $\mathbb{S}$, eventually q will become the starting point of the maximal window. The full proof of this part, however, is omitted due to space restrictions. □

Example 2. Assume we have the following stream $\mathbb{S}_{27}$:

$$\begin{array}{c|c c c|c c}
4/9 & 4/10 & & 2/3 & 1/2 & \\
\boxed{\text{aaabbbabb}} & \boxed{\text{ababababbb}} & \text{b} & \boxed{\text{aab}} & \boxed{\text{ab}} & \text{ba}
\end{array}$$

In the stream, two positions have been marked with $|$. Both these points do not meet the criteria given in Corollary 1 to be a border. Indeed, for both positions, a block before and after it is indicated such that the frequency in the before-block is higher than in the after-block. In this stream, the only positions that meet the requirement are indicated in $aaabbbabbababababbbb|aababb|a$. □

The following simple facts play an important role in the algorithm that will be developed in the next section.

Corollary 2. *Every border always ends on a target item. If p is not a border in $\mathbb{S}$, then neither it can ever be a border in any extension $\mathbb{S} \cdot \mathbb{B}$.*

4 Algorithm

Based on the theorems of Section 3, we now present an incremental algorithm to maintain a summary.

The summary. The summary for an item a in the stream $\mathbb{S}_t$ is the array of that contains a pair $(p, x/y)$ for every border on position p, with x the number of occurrences of a since p, i.e., $\mathit{count}(a, \mathbb{S}_t[p, t])$, and y the length of the block from p until the end of the stream $\mathbb{S}_t$, i.e., $t-p+1$. The output of the algorithm in the case of r borders is written as an array of the form $[(p_1, x_1/y_1), \cdots, (p_r, x_r/y_r)]$, visualized by

$$T_t = \begin{array}{|c|c|c|}\hline p_1 & \cdots & p_r \\ \hline x_1/y_1 & \cdots & x_r/y_r \\ \hline\end{array} \ .$$

This array is in fact $\mathit{summary}(\mathbb{S}_t)$, and is abbreviated by T_t. In this array, the border positions are ordered from old to most recent, reflecting in $p_1 < \cdots < p_r$. The corresponding frequencies must follow the same ordering $x_1/y_1 < \cdots < x_r/y_r$; indeed, consider two consecutive borders p_i and p_{i+1}. Suppose for the sake of contradiction, that $x_i/y_i \geq x_{i+1}/y_{i+1}$. From this, it follows that

$$\mathit{freq}(a, \mathbb{S}[p_i, p_{i+1}-1]) = \frac{x_i - x_{i+1}}{y_i - y_{i+1}} \geq \frac{x_i}{y_i} = \mathit{freq}(a, \mathbb{S}[p_{i+1}, |\mathbb{S}|]) \ .$$

According to Theorem 1 this implies that p_{i+1} cannot be a border because the frequency of a in a before-block is at least the frequency of a in an after-block for p_{i+1}. *Notice that this implies that the current frequency can always be obtained immediately from the summary; the most recent entry in the summary is always the largest and thus holds the current max-support.*

In every next step, the algorithm adjusts the stored values in the array based on the newly entered item in the stream. Hence, at every step, we need to test for every border of $\mathbb{S}_t$ if it is still a border in $\mathbb{S}_{t+1}$. Hence, we need to check if still the frequency in all before blocks is smaller than the frequency in all after blocks. *However, adding a new item to the stream does not introduce a new before-block, and only one after-block!* Hence, only one test has to be performed for every border of $\mathbb{S}_t$: the before-block with the highest frequency of a has to be compared to the new after-block. The frequency of the new after-block for border p_i can easily be obtained from the pair $(p_i, x_i/y_i)$ in the summary of $\mathbb{S}_t$: if the new item is a non-target item, the frequency of a in the new after-block is $x_i/(y_i+1)$. *Notice that, as the before block never changes when a new item enters the stream, and the insertion of the target item a only results in an increased frequency in the new after block, the addition of a target-item will never result in the removal of borders.*

Based on the observation that in any summary the borders must be in order w.r.t. the frequency of a, it is not too hard to see that the before-block with the maximal frequency is exactly the block $\mathbb{S}_t[p_{i-1}, p_i - 1]$. Via a similar reasoning as above, it follows that p_i is still a border for $\mathbb{S}_{t+1}$ if and only if the updated frequency $x_i/(y_i + 1)$ is still larger than the updated frequency $x_{i-1}/(y_i + 1)$ of p_{i-1}. To summarize, we have the following properties:

- The frequencies in the summary are always increasing.
- When the target item is added to the stream, all borders remain borders. The frequencies of the borders can be updated by incrementing all nominators and denominators by 1. A new entry with the current timestamp and frequency 1/1 can be added, unless the last entry has also 100% frequency.
- If a non-target item enters the stream, the frequencies of the borders can be updated by adding 1 to the denominators of the frequencies. All former borders for which after the update, the frequency no longer is larger than in the previous entry, are no borders anymore.

Algorithm 1 is based on these observations.

The Algorithm. Before the first target item enters the stream, the array will remain empty. The pseudo-code of the algorithm to create T_{t+1}, based on T_t and the item t that enters the stream at time $t + 1$ is given in Algorithm 1. In short, when a new item i enters the stream, the frequencies are updated by increasing the nominators if i equals the target item, and always increasing the denominators. If the item i is the target item, a new border will be added only if the frequency of the last block in T_t was not equal to 1. Furthermore, we have to take into account that some of the borders of $\mathbb{S}_t$ might no longer be borders in $\mathbb{S}_{t+1}$. This can only happen if the item that enters the stream is a non-target

Algorithm 1 $Update(T_t, i)$ for target item a on time $t+1$

Require: $T_t = summary(\mathbb{S}_t) = [(p_1, x_1/y_1), \cdots, (p_r, x_r/y_r)]$
Ensure: $T_{t+1} = summary(\mathbb{S}_{t+1}) = summary(\mathbb{S}_t \cdot \langle i \rangle)$

```
 1: Set $T_{t+1} := [\ ]$
 2: if ($T_t$ is empty) then
 3:    if ($i$ = target item $a$) then
 4:       $T_{t+1} := [(t+1, 1/1)]$
 5: else
 6:    if ($i$ = target item $a$) then
 7:       for $1 \leq j \leq r$ do
 8:          $T_{t+1} := T_{t+1} + \big(p_j, (x_j+1)/(y_j+1)\big)$
 9:       if $x_r \neq y_r$ then
10:          $T_{t+1} := T_{t+1} + \big(t+1, 1/1\big)$
11:    else
12:       $high := 0$
13:       for all $j := 1 \dots r$ do
14:          if $(x_j)/(y_j+1) > high$ then
15:             $T_{t+1} := T_{t+1} + \big(p_j, (x_j)/(y_j+1)\big)$
16:             $high := (x_j)/(y_j+1)$
```

item, and is tested in lines 12-16: the frequencies have to increase for increasing positions of the borders.

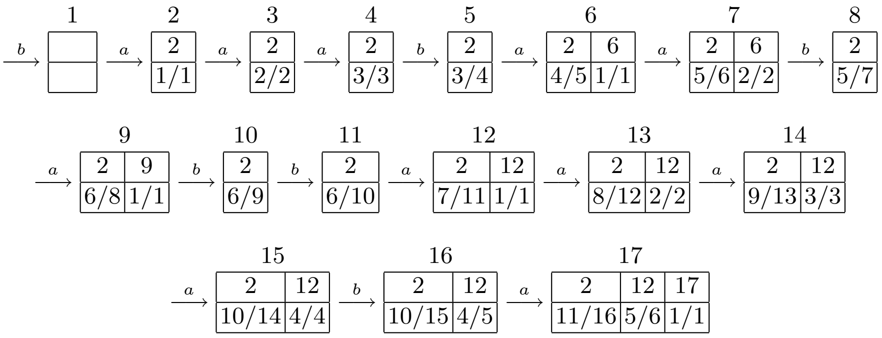
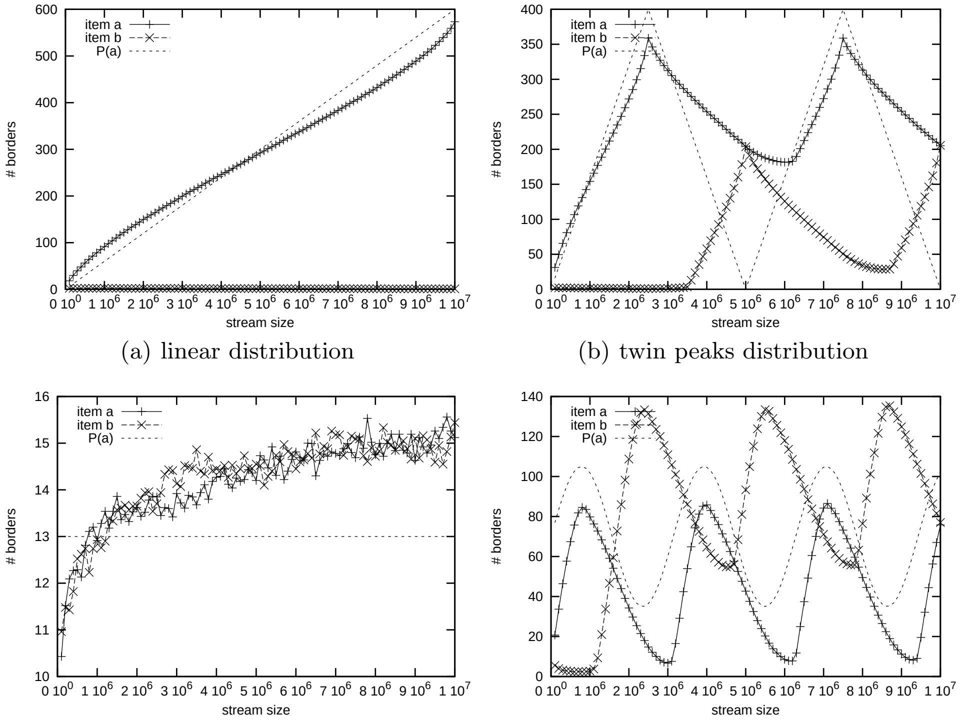


Fig. 2. Example for stream *baaabaababbaaaaba*.

We explain the working of the algorithm for the stream *baaabaababbaaaaba*. For each timestamp, the output of the algorithm is given in Figure 2.

In this example, some interesting things happen. First of all, the stream starts with a junk item b. Therefore, $Update(T_0, b) = Update([\ ], b)$ on timestamp 1 remains empty, i.e., $T_1 = [\ ]$. The algorithm in fact really starts at timestamp 2. At this moment, $Update([\ ], a)$ results in $T_2 = [(2, 1/1)]$, corresponding to the stream $b|a$ with a border at position 2. On timestamp 8, another interesting fact



(a) linear distribution

(b) twin peaks distribution

(c) random distribution

(d) sinus distribution

Fig. 3. Size of the summaries for two items a and b

happens. $T_7 = [(2, 5/6), (6, (2/2))]$, corresponding with the stream $b|aaab|aa$. $Update(T_7, b)$ will yield $T_8 = [(2, 5/7)]$, and not $[(2, 5/7), (6, 2/3)]$, because the frequency decreases from the border at position 2 to the border at position 6, and hence, we can conclude that position 6 is no longer a border.

5 Experiments

From the description of the algorithm it is clear that the update procedure is very efficient, given the summaries remain small. Producing the current support of the target item is obviously very efficient, as it amounts to simply a lookup in the most recent entry. Hence, the complete approach will be feasible if and only if the summaries remain small. Therefore, for different streams, we have recorded the size of the summary. The results are reported in Figure 3. The streams we consider are over the items a and b, and have length 10^7. After every 10 000 items, the size of the summary for the items a and b are reported. The streams are randomly generated. The probability of having item a in the stream is given by the line $P(a)$. Thus, in the random graph, the probability of having a is $1/2$ in the whole stream, independent of the moment. The probability of b is 1 minus the probability of a. The graphs report the average over 100 streams, generated with the indicated distributions. In general, we can conclude that the size of the summary is extremely small w.r.t. the size of the stream. If the probability of the target item increases, also the size of the summary will increase, when the probability decreases, the summary will shrink. This is easily explained by the entries in the summary that need to have increasing frequency.

6 Conclusion and Future Work

We presented a new frequency measure for items in streams that does not rely on a fixed window length or a time-decaying factor. Based on the properties of the measure, an algorithm to compute it was shown. An experimental evaluation supported the claim that the new measure can be computed from a summary with extremely small memory requirements, that can be maintained and updated efficiently.

In the future, we will look at the same topic, but try to mine for frequent itemsets instead of items, based on this new frequency measure.

References

1. Ruoming J. and Agrawal G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. in Proc. 5th IEEE Int. Conf. on Data Mining (ICDM'05), pp 210–217.
2. Agrawal R., Imielinski T. and Swami A.: Mining Association Rules between Sets of Items in Large Databases. in Proc. ACM SIGMOD Int. Conf. on Management of Data (1993), pp 207–216.
3. Cormode G. and Muthukrishnan S.: What's Hot and What's Not: Tracking Most Frequent Items Dynamically. in Proc. PODS (2003).
4. Demaine E.D., Lopez-Ortiz A. and Munro, J.I.: Frequency Estimation of Internet Packet Streams with Limited Space. in Proc. of the 10th Annual European Symposium on Algorithms (2002), pp 348–360.
5. Giannella C., Han J., Robertson E. and Liu C.: Mining Frequent Itemsets Over Arbitrary Time Intervals in Data Streams. Technical Report TR587 at Indiana University, Bloomington, (Nov 2003), 37 pages.
6. Karp, R. M., Papadimitriou, C. H. and Shenker, S.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. in ACM Trans. on Database Systems (2003), 28, pp 51–55.
7. Lee, L.K. and Ting, H.F.: A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. in ACM PODS (2006).
8. Misra J. and Gries, D.: Finding Repeated Elements. in Science of Computer Programming (1982), 2(2), pp 143–152.
9. Chi-Wing Wong R. and Wai-Chee Fu A.: Mining Top-K Frequent itemsets from Data Streams. in Data Mining and Knowledge Discovery. to appear 2006