Résumer efficacement des flux de données massifs en fenêtre glissante

Nicoló Rivetti, Yann Busnel, Achour Mostefaoui

Résumer efficacement des flux de données massifs en fenêtre glissante

Nicoló Rivetti[1,2], Yann Busnel[3,4] et Achour Mostefaoui[1]

[1]*LINA / Université de Nantes, France*
[2]*Sapienza University of Rome, Italie*
[3]*Crest / Ensai, Rennes, France*
[4]*Inria Rennes – Bretagne Atlantique, France*

Estimer la fréquence de n'importe quel item dans des flux de données massifs est un des problèmes majeurs de la dernière décennie. Si plusieurs solutions élégantes ont été proposées récemment, leur approximation est calculée depuis le commencement du flux. Dans un contexte applicatif en ligne, il serait préférable de collecter l'information sur un passé récent, tant pour économiser des ressources que par pertinence de l'information la plus récente. Dans cet article, nous considérons le modèle dit de *fenêtre glissante* et proposons deux algorithmes en ligne qui estiment la fréquence de chaque item dans la fenêtre courante. Ces algorithmes sont des (ε,δ)-approximations absolues des valeurs de fréquences réelles, utilisant une faible quantité mémoire (respectivement $O(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log N+\log n))$ et $O(\frac{1}{\tau\varepsilon}\log\frac{1}{\delta}(\log N+\log n))$ bits, où N est la longueur de la fenêtre, n est le nombre d'items distincts du flux et τ est un paramètre permettant de limiter l'utilisation mémoire. Les expérimentations conduites, comparant nos solutions à celles de l'état de l'art, illustrent la validité et la robustesse de nos algorithmes.

Mots-clefs : Flux de données; Fenêtre glissante; Estimation de fréquence; Algorithme d'approximation probabiliste

Ces travaux ont été partiellement financés par le projet ANR SocioPlug (ANR-13-INFR-0003) et le projet DeSceNt du Labex CominLabs (ANR-10-LABX-07-01).

1 Introduction

In many systems it is most likely critical to gather efficiently various aggregates over massive data that may be generated at very high speed. A straightforward application is network monitoring, for instance keeping track of the frequencies of IP addresses in a subnet. This can be modelled by a node trying to continuously evaluate a given function over stream of items it can observe. The main goal is to evaluate such functions at the lowest cost in terms of the space used, as well as minimizing the update and query time. The solutions proposed so far are focused on computing functions or statistics using ε or (ε,δ)-approximations in poly-logarithmic space over the size m of the stream and the number n of its distinct items.

Datar *et al.* [DGIM02] introduced the sliding window concept in the data streaming model presenting the *exponential histogram* algorithm that provides an ε-approximation for basic counting. In this paper, we tackle the frequency estimation problem in the sliding window model. Using little memory (low space complexity) implies some kind of data aggregation. If the number of counters is less than the number of different items then necessarily each counter encodes the occurrences of more than one item. The problem is then how to slide the window to no more keep track of the items that exited the window and how to introduce new items. We extend the well-known algorithm for frequency estimation, namely the COUNT-MIN sketch [CM05], in a windowed version. We propose our approach in two steps, two first naive and straightforward algorithms called PERFECT and SIMPLE followed by two more sophisticated ones called PROPORTIONAL windowed and SPLITTER windowed algorithms. Then, we compare their respective performances together with the ECM-sketches solution, proposed in [PGD12] (the only work that tackles a similar problem, to our knowledge).

2 Data Streaming Model

We consider a massively long input stream σ, that is, a sequence of elements $\langle a_1, a_2, \ldots, a_m, \ldots \rangle$ called samples. Samples are drawn from a universe $[n] = \{1, 2, \ldots, n\}$ of items. The size of the universe (or number of distinct items) of the stream is n. This sequence can only be accessed in its given order (no random access). We rely on randomized algorithms that implement approximations of our goals. Namely, such an algorithm $\mathcal{A}$ evaluates the stream in a single pass (on-line) and continuously. It is said to be an (ε, δ)-additive-approximation of a function ϕ on a stream σ if, for any prefix of size m of items of the input stream σ, the output $\hat{\phi}$ of $\mathcal{A}$ is such that $\mathbb{P}\{|\ \hat{\phi} - \phi\ | > \varepsilon C\} < \delta$, where ε, $\delta > 0$ are given as precision and error parameters, while C is an arbitrary constant. On the other hand, the *sliding window* model, formalized by Datar *et al.* [DGIM02], models that when the function ϕ is evaluated, it will be only on the N more recent items among the m items already observed. In this model, samples arrive continuously and expire after exactly N steps. The additional problem brought by a sliding window resides in the fact that when a prefix of a stream is summarized we lose the temporal information related to the different items making the exclusion of the most ancient items non trivial with little memory.

3 Windowed Count-Min

The problem we tackle in this paper is the *frequency estimation* problem. In a stream, each item appears a given number of times that allows to define its frequency. The function that defines this problem returns a frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$ where f_j represents the number of occurrences of item j in the portion of the input stream σ evaluated so far. The goal is to provide an estimate $\hat{f}_j$ of f_j for each item $j \in [n]$.

Cormode and Muthukrishnan have introduced in [CM05] the COUNT-MIN sketch that provides, for each item j an (ε, δ)-additive-approximation $\hat{f}_j$ of the frequency f_j. Briefly, the CM algorithm maintains a two-dimensional array $\hat{F}$ of $c_1 \times c_2$ counters with $c_1 = \lceil \log(1/\delta) \rceil$ and $c_2 = \lceil \exp(1)/\varepsilon \rceil$, and uses a set of 2-universal hash functions $h_1, \ldots, h_{c_1}$. Each time an item j is read from the input stream, this causes one counter per line to be incremented, *i.e.*, $\hat{F}[u][h_u(j)]$ is incremented for all $u \in \{1, \ldots, c_1\}$. When a query is issued to get an estimate $\hat{f}_j$, the returned value corresponds to the minimum among the c_1 values of $\hat{F}[u][h_u(j)]$, $1 \leq u \leq c_1$. Fed with a stream of m items, the space complexity of this algorithm is $O(\frac{1}{\varepsilon} \log \frac{1}{\delta} (\log m + \log n))$ bits, while update and query time complexities are $O(\log 1/\delta)$. Concerning its accuracy, the following bound holds : $\mathbb{P}\{|\ \hat{f}_j - f_j\ | \geq \varepsilon(m - f_j)\} \leq \delta$, while $f_j \leq \hat{f}_j$ is always true.

We propose two extensions in order to meet the sliding window model : PROPORTIONAL and SPLITTER. Nevertheless, we first introduce two naive algorithms that enjoy optimal bounds with respect to accuracy (algorithm PERFECT) and space complexity (algorithm SIMPLE). Note that in the following f_j is redefined as the frequency of item j in the last N samples among the m items of the portion of the stream evaluated so far. Due to space constraints, algorithm pseudo-codes and proofs are available in the companion paper [RBM15], which the interested reader is invited to consult.

Perfect Windowed Count-Min PERFECT provides the best accuracy by dropping the complexity space requirements : it trivially stores the whole active window. When it reads sample j, it enqueues j and increases all the $\hat{F}$ matrix cells associated with j. Once the queue reaches size N, it dequeues the expired sample j' and decreases all the cells associated with j'. The frequency estimation is retrieved as above.

Simple Windowed Count-Min SIMPLE is as straightforward as possible and achieves optimal space complexity with respect to the vanilla algorithm. It behaves as the COUNT-MIN, except that it resets the $\hat{F}$ matrix at the beginning of each new window. Obviously it provides a really rough estimation since it simply drops all information about any previous window once a new window starts.

Proportional Windowed Count-Min We now present the first extension algorithm, denoted PROPORTIONAL. The intuition behind this algorithm is as follows. At the end of each window, it stores separately a snapshot of the $\hat{F}$ matrix, which represents what happened during the previous window. Starting from the current $\hat{F}$ state, for each new sample, it increments the associated cells and decreases all the $\hat{F}$ matrix cells proportionally to the last snapshot. This smooths the impact of resetting the $\hat{F}$ matrix throughout the current window.

Theorem 3.1 PROPORTIONAL *space complexity is* $O(\frac{1}{\varepsilon}\log\frac{1}{\delta}(\log N+\log n))$ *bits. Update and query time complexities are* $O(\frac{1}{\varepsilon}\log 1/\delta)$ *and* $O(\log 1/\delta)$.

Splitter Windowed Count-Min PROPORTIONAL removes the *average* frequency distribution of the previous window from the current window. Consequently, PROPORTIONAL does not capture sudden changes in the stream distribution. To cope with this flaw, one could track these critical changes through multiple snapshots. However, each row of the $\hat{F}$ matrix is associated with a specific 2-universal hash function, thus changes in the stream distribution will not affect equally each rows.

Therefore, SPLITTER proposes a finer grained approach analyzing the update rate of each cell in $\hat{F}$. To record changes in the cell update rate, we add a (fifo) queue of sub-cells to each cell. When SPLITTER detects a relevant variation in the cell update rate, it creates and enqueues a new sub-cell. This new sub-cell then tracks the current update rate, while the former one stores the previous rate.

Each sub-cell has a frequency *counter* and 2 timestamps : *init*, that stores the (logical) time where the sub-cell started to be active, and *last*, that tracks the time of the last update. After a short bootstrap, any cell contains at least two sub-cells : the current one that depicts what happened in the very recent history, and a predecessor representing what happened in the past. Figure 1 illustrates a possible state of the data structure of SPLITTER, after reading a prefix of 101 items of σ, which is introduced in the top part of the figure with all the parameters of SPLITTER.

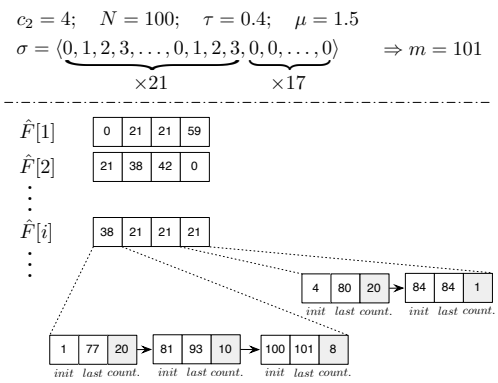


FIGURE 1: State of the data structure of SPLITTER after a prefix of 101 items of σ.

SPLITTER spawns additional sub-cells to capture distribution changes. The decision whether to create a new sub-cell is tuned by two parameters, τ and μ, and an error function : ERROR. Informally, the function ERROR evaluates the potential amount of information lost by merging two consecutive sub-cells, while μ represents the amount of affordable information loss. Performing this check at each sample arrival may lead to erratic behaviors. To avoid this, we introduced τ, such that $0 < \tau \leq 1$, that sets the minimal length ratio of a sub-cell before taking this sub-cell into account in the decision process. More details is available in the companion paper [RBM15].

Lemma 3.2 *[Number of Splits Upper-bound] Given* $0 < \tau \leq 1$*, the maximum number* $\bar{s}$ *of splits (number of sub-cells spawned to track distribution changes) is* $O(\frac{1}{\varepsilon\tau}\log\frac{1}{\delta})$.

Theorem 3.3 SPLITTER *space complexity is* $O(\frac{1}{\tau\varepsilon}\log\frac{1}{\delta}(\log N+\log n))$ *bits, while update and query time complexities are* $O(\log 1/\delta)$.

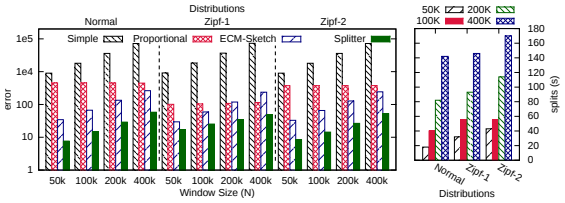
For comparizon, the closest related work, that is ECM-SKETCH [PGD12], owns a space complexity of $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\left(\log^2\varepsilon N+\log n\right)\right)$ bits and its update and query times are $O(\log 1/\delta)$. Finally, distributed and time-based versions of our algorithms are also available in the companion paper [RBM15].

4 Performance Evaluation

We have conducted a series of experiments on different types of streams and parameter settings, to compare the respective efficiency of both proposed algorithms and also to compare them to the only similar work in the related works. The wave-based version of ECM-SKETCH [PGD12] that we have implemented replaces each counter of the $\hat{F}$ matrix with a *wave* data structure. Each wave is a set of lists, the number and the size of such lists is set by the parameter $\varepsilon_{wave} = \varepsilon$. To verify the robustness of our algorithms, we have fed them with synthetic traces and real-world datasets. The latter give a representation of some existing monitoring applications, while synthetic traces allow to capture phenomena that may be difficult to obtain otherwise. Each run has been executed a hundred times, and we provide the mean over the repeated runs,

after removing the 1st and 10th deciles to avoid outliers. Finally, the accuracy metric used in our evaluation is the mean absolute error of the frequency estimation of all n items returned by the algorithms with respect to PERFECT, that is $\left(\sum_{j\in[n]} \left|\hat{f}_j^{\text{PERFECT}} - \hat{f}_j^{\text{TESTEDALGORITHM}}\right|\right)/n$. We refer to this metric as *estimation error*.

Window sizes Figure 2(a) presents the estimation error of the algorithms with a stream of $m = 3 \times N$ samples and $n = 1,000$ distinct items, while considering three distributions : Normal, Zipf with $\alpha = 1$ and Zipf with $\alpha = 2$. SIMPLE is always the worst (with an error equals to 3395 in average), followed by PROPORTIONAL (451 in average), ECM-SKETCH (262 in average) and SPLITTER (57 in average). In average, SPLITTER error is 4 times smaller than ECM-SKETCH, with 4 times less memory requirement. Figure 2(b) gives the number of splits spawned by SPLITTER in average to keep up with the distribution changes. The number of splits grows in average with a factor 1.7 for each 2-fold increase of N. In fact, as τ is fixed, the minimal size of each sub-cell grows with N, and so does the error.



(a) Average estimation error (b) Average splits

FIGURE 2: Results for different window sizes (N)

Multiple distributions according to time Figure 3 presents the estimation error evolution as the stream unfolds. Here, the stream distribution is shifted every 15,000 samples and swapped each 60,000 samples in the order proposed on top of the plot. Note that, in order to avoid side effect, the distribution shift and swap periods are not synchronised with the window size ($N = 50,000$). SPLITTER error does not exceed 23 (and is equal to 13 in average). ECM-SKETCH maximum error is 65 (29 in average), as PROPORTIONAL goes up to 740 (207 in average) and SIMPLE reaches 1877 (1035 in average). Since at the beginning of each window SIMPLE resets its $\hat{F}$ matrix, there is a periodic behavior : the error burst when a window starts and shrinks towards the end. In the 1-st window period (0 to 50,000) and in the 6-th windows (250,000 to 300,000) the distribution does not change over time (shifting Uniform has no effect). This means that SPLITTER does not capture more information than PROPORTIONAL, thus they provide the same estimations in the 2-nd and the 7-th windows (respectively between 50,000 and 100,000 samples then between 300,000 and 350,000 samples).

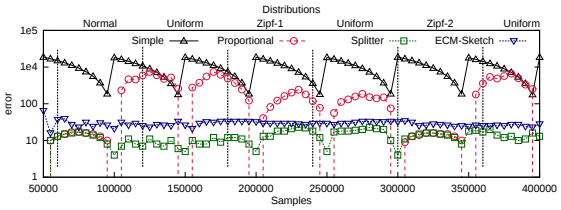


FIGURE 3: Estimation error with multiple distributions

The complete study [RBM15] shows the accuracy of both algorithms and that they outperform the only existing solution with real world traces and also with specifically tailored adversarial synthetic traces. Last but not least, these results reach better estimation with respect to the state of the art proposal and required 4 times less memory usage. We have also studied the impact of the two additional parameters of the SPLITTER algorithm (τ and μ).

Références

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary : The count-min sketch and its applications. *J. of Algorithms*, 55, 2005.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. on Computing*, 31, 2002.

[PGD12] Odysseas Papapetrou, Minos N. Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proc. of the VLDB Endowment*, 2012.

[RBM15] N. Rivetti, Y. Busnel, and A. Mostefaoui. Efficiently summarizing data streams over sliding windows. In *Proc. of the 14th IEEE International Symposium on Network Computing and Applications (NCA'15)*, Boston, USA, *Best Student Paper Award*, September 2015.