Lottery Sampling, Lottery Saving and other Algorithms for Top k Frequent Elements

Conrado Martínez Gonzalo Solera

September 3, 2018

1 Preliminaries

Consider a (large) data stream $\mathcal{Z} = z_1, \ldots, z_N$, where each z_j is drawn from some domain or *universe* $\mathcal{U}$, and let $n \leq N$ be the number of distinct elements in $\mathcal{Z}$. We may thus look at the multiset X underlying $\mathcal{Z}$

$$X = \{x_1^{f_1}, \ldots, x_n^{f_n}\}$$

where x_1, …, x_n are the n distinct elements that occur in $\mathcal{Z}$ and f_i denotes the number of occurrences (absolute frequency) of x_i in $\mathcal{Z}$. We will assume, w.l.o.g., that we index the elements in X in non-increasing order of frequency, thus $f_1 \geq f_2 \geq \cdots \geq f_{n-1} \geq f_n > 0$. We will use $p_i = f_i/N$ to denote the relative frequency of x_i. For simplicity, we will assume that $f_1 > f_2 > \cdots > f_n$ in the definitions below—they can be more or less easily adapted to cope with elements of identical frequency.

The two problems that we want to study here are:

1. Top k most frequent elements. Given $\mathcal{Z}$ and a value $k \leq n$, we want to find $\{x_1, \ldots, x_k\}$ (or any subset of k distinct elements with maximal frequencies).

2. Heavy hitters. Given $\mathcal{Z}$ and a value c, $0 < c < 1$, we want to find (or count) the number of distinct elements in $\mathcal{Z}$ with relative frequency $p_i \geq c$. Those elements are called *heavy hitters.* Given the data stream and the value c, we want to obtain $\{x_1, \ldots, x_{k^*}\}$, where k^* is the largest value k such that $p_k \geq c$. The value k^* is the number of heavy hitters.

Moreover, in both problems, we might want that the algorithm returns the frequency f_i of the returned elements.

None of these two problems can be solved exactly unless we can keep $\Theta(n)$ elements in memory; thus under the tight memory constraints of the data stream model, we must aim at approximate good solutions. Hence, the algorithms that we describe next might return elements which are not among the most frequent elements, or that are not heavy hitters, and rather than the frequencies f_i of the returned elements, we will have to content ourselves with estimations f'_i of the real frequencies.

We will concentrate in algorithms for top k most frequent elements. Notice that there can be at most $\lceil 1/c \rceil$ heavy hitters in a data stream, and thus an algorithm that retrieves the top $k^* = \lfloor /c \rfloor$ most frequent elements will obtain all the heavy hitters.

2 The Algorithms

The algorithms that we shall consider next fail under the following scheme. They all keep a *sample* $\mathcal{S}$ of up to m distinct elements, for some value $m \geq k$ fixed in advance. Each element x in the sample is eqquiped with a frequency counter

$$f'(x) = f_{\text{obs}}(x) + f_{\text{ini}}(x),$$

which is the sum of two components: $f_{\text{obs}}(x)$ is the number of times that x has been observed in the data stream since it has been included in $\mathcal{S}$; $f_{\text{ini}}(x)$ is an estimation of the number of occurrences of x prior to the occurrence in which x has been added to the sample.

In the initial phase, the algorithms populate the sample with the initial distinct elements in the data stream, recording their frequencies. In particular, $f_{\text{ini}}(x) = 0$ for these elements, and $f'(x)$ will be the real frequenciy of x in the "prefix" of the data stream examined so far. This first phase ends when m distinct elements have already been collected and an $(m+1)$-th distinct element occurs in the data stream, or when the data stream is exhausted—because $m \geq n$. In the latter case, the sample has complete information about the full data stream and all required information can be obtained exactly. In the interesting case, when $m < n$ (typically, $m \ll n$), the algorithms loop through the remaining items in the data stream $\mathcal{Z}$. For every incoming item z, if z is already in $\mathcal{S}$, the algorithms update $f'(z)$ and perhaps perform some additional bookkeeping (depending on the particular algorithm under consideration), but not much more has to be done. On the other hand, if $z \notin \mathcal{S}$ then we apply some criterium `add?`$(z, \mathcal{S})$; if `add?`$(z, \mathcal{S})$ is not satisfied, then z is discarded. Otherwise, some element $z' \in \mathcal{S}$ is choosen and evicted from the sample ($z' =$ `element_to_evict`$(\mathcal{S})$), and z is added to $\mathcal{S}$; the frequency counter is initialized $f'(z) = 1 + f_{\text{ini}}(z)$. The algorithms that we will consider differ hence in three main aspects:

1. When do we add to the sample an incoming item $z \notin \mathcal{S}$ (`add?`$(z, \mathcal{S})$? In several algorithms that we will consider later, this is a randomized choice.

2. Which element z' do we evict from the sample when a new element z has to be added to the sample (`element_to_evict`$(\mathcal{S})$)? Again, this choice might be probabilistic.

3. How do we estimate the frequency $f_{\text{ini}}(z)$, that is, the number of prior occurrences of a new element z that is to be added to the sample?

Take for instance the well-known SpaceSaving (SS) [**?**]. It always adds incoming items $z \notin \mathcal{S}$, that is, `add?`$(z, \mathcal{S}) = $ **true**. The element z' selected for eviction is one with minimal estimated frequency

$$z' = \arg\min_{x \in \mathcal{S}} \{f'(x)\}.$$

And the newly added item z inherits the frequency of the evicted item: $f_{\mathrm{ini}}(z) = f'(z')$.

Let us know consider our first new algorithm LotterySampling (LS). Each element x in the sample has, besides its frequency counter, a ticket $t(x) \in (0, 1)$. When a new item z in the data stream is retrieved LS generates, uniformly at random in $(0, 1)$, a ticket t for z. If z was already in the sample its frequency counter $f'(z)$ is updated (as usual), but also its ticket: if $t > t(z)$ then $t(z) := t$. If z is not in the sample, we compare t with the minimum ticket $t_{\min} = t(z')$ in the sample. If $t > t_{\min}$ then z is added to the sample, otherwise z is discarded. The element z' to be evicted is the one with the minimum ticket, and the frequency counter of z is initialized with

$$f_{\mathrm{ini}}(z) = \left\lfloor \frac{1}{1 - t_{\min}} \right\rfloor .$$

LotterySaving-LFU (LS-LFU) combines some of the ideas of the two algorithms above. When an item z is not in the sample we compare its ticket t with the ticket $t' = t(z')$ of the element z' with smallest frequency counter. Notice that t' might be or not the minimum ticket in the sample. If $t > t'$ then z is added and z' is evicted, and $f_{\mathrm{ini}}(z) = f'(z')$.

LotterySaving-LRU (LS-LRU) is similar to LS-LFU, but the ticket t a candidate element z to enter the sample is compared with the ticket $t' = t(z')$ of the item z' in the sample that was observed least recently (i.e., z' is the item with frequency counter updated the longest ago). Like in LS-LFU, if $t > t'$ then z is added and z' is evicted, and $f_{\mathrm{ini}}(z) = f'(z')$.

LotterySaving-Threshold (LS-THR-θ) has a parameter $\theta \in [0, 1]$. We omit the parameter θ in the name, thus write LS-THR in generic discussions about this algorithm. A new item $z \notin \mathcal{S}$ is added to the sample if and only if the ticket t generated for z is larger that $\theta \cdot t_{\max}$, where $t_{\max}$ is the largest ticket among the elements in the sample (LS-THR-0 is equivalent to SS).

If z is added to $\mathcal{S}$, LS-THR evicts the element z' with minimum frequency counter and z inherits the frequency counter of z' as the initial estimation: $f_{\mathrm{ini}}(z) = f'(z')$.

LotterySaving-AboveMean (LS-AM) adds a new item to the sample if its ticket t is larger that the mean value $\bar{t}$ of all the tickets in the sample. When $t > \bar{t}$, the evicted item z' is the one with smallest frequency counter and z inherits $f'(z')$ like in LS-LFU and LS-THR.

Another variation is LotterySaving-AboveMedian (LS-MED) which adds a new item to the sample if its ticket t is above the median t_{med} of the tickets in the sample. The remaining choices for the algorithm are as in LS-AM. LS-MED can be generalized to consider the α-quantile (LS-QUANT-α) of the tickets in the sample. Notice that LS-MED is LS-QUANT-0.5.

The basic variants of LS-LFU and LS-LS-LRU can be modified to incorporate the ideas behind LS-AM and LS-MED. Thus LS-LFU-AM and LS-LRU-AM will add an element $z \notin \mathcal{S}$ to the sample if $t > t(z')$ **or** t is above the mean value $\bar{t}$ of the tickets in the sample. The evicted item z' and the initialization of the frequency counter of z is as in the corresponding basic algorithms.

Instead of the mean value $\bar{t}$ of the tickets in the sample, we might consider the median of the tickets, or more generlly, the α-quantile of the tickets, getting the variants LS-LFU-MED LS-LRU-MED, or more generally, LS-LFU-QUANT-α and LS-LRU-QUANT-α.

Another source of variations stems from the way tickets are updated when an item z already in the sample is the incoming element from the stream. In all cases above we have assumed that tickets are updated via $t(z) := \max(t, t(z))$, where t is the ticket generated for the current instance of z, whereas $t(z)$ is the ticket associated to z in the sample.

All algorithms using tickets can be also characterized as defining a *thresold value* t^* and a candidate item z^* for eviction. The criterium for addition of a element $z \notin \mathcal{S}$ is always whether $t > t^*$ or not, where t is the ticket that was generated for the current instance. For example, lottery sampling (LS) takes $t^* \equiv t_{\min} = \min\{t(z) \mid z \in \mathcal{S}\}$, and z^* an element with the minimum ticket $t_{\min}$. Observe that we can safely assume that all tickets in $\mathcal{S}$ are distinct, and we shall use $t_{(r)}$ to denote the r-th smallest ticket in the sample, $1 \leq r \leq m$.

With these conventions the template for all the algorithms discussed above is as follows:

```
Populate S with the first m distinct elements in Z,
         updating tickets and frequencies
while Z is not exhausted do
      z:= current item in Z
      t:= Random(0,1)
      if z in S then
           Update the ticket t(z)
           f'(z):= f'(z) + 1
      else if t > t* then
           S:= S - {z*} + {z}
           t(z):= t
           f'(z):= fini(z) + 1
      else // do nothing
      Update t* and z*
endwhile
Report the k elements in S with largest f'
```

Table **??** summarizes the characteristic values of t^* and z^*. Observe that we have considered two options for the initialization of $f_{\text{ini}}(z)$ and three options for z^* (the element with minimum t, the element with minimum f', and the least recently accessed element); if we contemplate C different ways to define t^*

Algorithm	t^*	z^*	$f_ini(z)$
SS	0	the element of minimum f'	$f'(z^*)$
LS	$t_{(1)} = t(z^*)$	the element of minimum t	$\lfloor \frac{1}{1-t^*} \rfloor$
LS-LFU	$t(z^*)$	the element of minimum f'	$f'(z^*)$
LS-LRU	$t(z^*)$	the element accessed longest ago	$f'(z^*)$
LS-THR	$\theta \cdot t_{(m)} = \theta \cdot t_{\max}$	the element of minimum f'	$f'(z^*)$
LS-AM	$\bar{t} = \sum_{z\in\mathcal{S}} t(z)/m$	the element of minimum f'	$f'(z^*)$
LS-QUANT	$t_{\lceil \alpha m \rceil}$	the element of minimum f'	$f'(z^*)$
LS-LFU-AM	$\min\{\bar{t}, t(z^*)\}$	the element of minimum f'	$f'(z^*)$
LS-LFU-MED	$\min\{t_{\lfloor (m+1)/2 \rfloor}, t(z^*)\}$	the element of minimum f'	$f'(z^*)$

($C = 9$ in our table), we could consider up to $6C$ different possible combinations. If we add on top of that the choice to update or not tickets of sampled elements we raise the count to $12C$. Moreover, LS-THR and LS-QUANT depend on an additional real parameter in $[0, 1]$ giving us an additional source of variability.

Other strategies?

It is clear that the combinations that we can make "playing" around with the criteria for addition ($\texttt{add?}(z, \mathcal{S})$), the criteria for eviction and the initialiation of the frequency counter are almost endless, but here we will restrict ourselves to the those that we have been able to analyze and for which the experiments give the best results. Another important issue in our choice has been the efficiency with which the different operations can be supported. Thus for instance, to implement LS-LFU it is very convenient to maintain the elements of $\mathcal{S}$ sorted by frequency counter; locating the least frequent element becomes trivial, and maintaining the order of the sample after each frequency update is also very easy and efficient, as they increment one by one.

2.1 Implementing the Algoithms

3 Measures of Quality

Of course, for a top k frequent elements algorithm we would like that it reports the true k most frequent elements and their respective frequencies, but we know that this will not be possible unless we had a linear amount of memory ($Theta(n)$)—this includes the trivial situation when $m \geq n$. Hence we will assume in waht follows that $m \ll n$, and we would like that our algorithms return good approximations: elements that are the most frequent or close to that, and good estimations of their respective frequencies. To measure the quality of the different algorithm we introduce several measures, in which two random variables will be essential:

1. The indicator variable Y_i' tells us whether x_i has been sampled or not: $Y_i' = 1$ if $x_i \in \mathcal{S}$ and $Y_i' = 0$, otherwise. Notice that the Y_i''s are not independent and $Y_1' + \cdots + Y_n' = m$.

2. Since $m \geq k$, the algorithm has to choose k out of the m elements to report them as the top k most frequent elements. Quite naturally, in all our algorithms the k elements with largest frequency counter will be the ones to be reported and their estimated frequencies will be those collected by the algorithm. The indicator variable $Y_i = 1$ if x_i is reported, $Y_i = 0$ otherwise (either because $x_i \notin \mathcal{S}$ or because $f'(x_i)$ is not among the largest k values of f' in $\mathcal{S}$). Like the Y_i''s, the Y_i's are not independent; and $Y_1 + \cdots + Y_n = k$.

Our first measure, *(weighted) recall*, measures the proportion of top frequent elements actually found by the algorithm:

$$R = \frac{p_1 Y_1 + \cdots + p_k Y_k}{p_1 + \cdots + p_k}$$

A proxy for R is given by

$$R' = \frac{p_1 Y_1' + \cdots + p_k Y_k'}{p_1 + \cdots + p_k},$$

which is always an upper boud for R, since $Y_i \leq Y_i'$ for all i, $1 \leq i \leq n$. Notice that, in all cases, $0 \leq R \leq 1$: $R = 0$ if no true frequent element is reported, and $R = 1$ if exactly the top k most frequent elements are reported. The weights p_i will penalize more algorithms that miss the most frequent elements among the top frequent, and will lessen the impact in R if the algorithm missed less frequent elements (albeit among the most frequent). The same is true for R'.

Another measure is *(weighted) precision*, which gives us an indication of the quality of the reported elements:

$$P = \frac{p_1 Y_1 + \cdots + p_n Y_n}{p_1 + \cdots + p_k}$$

The numerator sums the weights of all reported elements; since exactly k elements are reported $0 < P \leq 1$. The proxy P' for P needs a different "normalizing" denominator:

$$P' = \frac{p_1 Y_1' + \cdots + p_n Y_n'}{p_1 + \cdots + p_m}$$

We can also use the "ordinary" measures of unweighted recall R_u and unweighted precision P_u used in most of the literatire (e.g. in the experimental study of SS []). In terms of our indicator random iables we have

$$R_u = \frac{Y_1 + \ldots + Y_k}{k}$$
$$P_u = \frac{Y_1 + \ldots + Y_k}{m},$$

and the approximations (actually, upper bounds)

$$R_u' = \frac{Y_1' + \ldots + Y_k'}{k}$$
$$P_u' = \frac{Y_1' + \ldots + Y_k'}{m}.$$

Besides the measures of “recall” and “precision” we also consider measures for the quality of the estimated frequencies.

Error type I ϵ_1 measures the deviation of the estimated frequencies for the frequent elements:

$$\epsilon_I = \frac{\sum_{1\le i\le k}(f_i - \hat{f}_i)^2}{\sum_{1\le i\le k} f_i^2},$$

where $\hat{f}_i = f'(x_i)$ if $x_i \in \mathcal{S}$ and $\hat{f}_i = 0$, otherwise. For reasons that will be discussed later we might also take $\hat{f}_i = \min\{f'(x_i), 2f_{\rm obs}(x_i)\}$. Since $f_{\rm obs}(x_i) \le f_i$ for all i, the numerator in ϵ_I cannot exceed the denominator and thus ϵ_I will range between 0 (perfect estimation and all frequent elements in the sample) and 1 (all frequent elements missing and/or grossly overestimated). This meaure does not take into account what are the non-frequent elements that get sampel or how good or bad are the estimates of their frequencies. For that we have a second measure, error type II. In this case the absolute errors in the estimates are not that useful; we should penalize more severely the errors in the estimation of elements of very low frequency since that might imply reporting them as frequent element and they are not even close. On the other hand, no penalty should be added if the element is not sample. Hence for the definition of error type II we will be doing better using relative errors, and weighting respect their presence or not in the sample:

$$\epsilon_{II} = \frac{1}{m}\sum_{i\ge k} Y_i' \left(\frac{\hat{f}_i - f_i}{f_i}\right)^2 .$$

Since each term in the sum above is at most 1 and there are at most m non-null terms, ϵ_{II} also ranges between 0 (no mistakes in the frequency estimation) and 1 (all sampled items are infrequent and we make a gross overestimation of their frequencies).

Other measures of quality

4 Analysis of the Algorithms

5 Experimental Results