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

Conrado Martínez Gonzalo Solera

November 21, 2018

Es muy buena idea que vayas escribiendo en este documento de trabajo u otro equivalente que constituya la base de tu documentación del TFG, las diferentes cosas que vayas haciendo, de la manera más organizada que puedas, aunque no esé muy pulido. Este documento te da un buen punto de partida sobre cómo organizar tu TFG (habrás de incluir algunas secciones "chorras", you know it) y también va estableciendo el marco de trabajo (p.e. esquema de nombres para los diferentes algoritmos, la plantilla "unificada" para todos ellos, las definiciones de las medidas de rendimiento, ...)

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 $x_1, \ldots, x_n$ 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 $\lfloor 1/c \rfloor$ heavy hitters in a data stream, and thus an algorithm that retrieves the top $k^* = \lfloor 1/c \rfloor$ 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

$f(z) = 1 + f_{\text{ini}}(z)$. As in the case where $z \in \mathcal{S}$, some additional bookkeeping might be necessary for each particular algorithm.

The algorithms that we will consider differ hence three main aspects:

1. When do we add to the sample an incoming item $z \notin \mathcal{S}$ ($\texttt{add?}(z, \mathcal{S})$? We will consider algorithms that make the decision by first generating a number $t \in [0, 1]$, a *token*, uniformly and independently at random. If $t \geq t^*$ then z will be added, where $t^* \in [0, 1]$ is a *threshold* value defined by each particular algorithm, and in general will depend on the "evolution" of the data stream—namely, on the state of the sample and all the tokens generated so far[1].

   The details about how t^* is computed—this includes whether tokens are generated for every instance in the data stream, or only for those for which we need to decide whether we add or not them to the sample—will lead to several different "subfamilies" of the generic algorithm.

2. Which element z^* do we evict from the sample when a new element z has to be added to the sample ($\texttt{element\_to\_evict}(\mathcal{S})$)? 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, $\texttt{add?}(z, \mathcal{S}) = \textbf{true}$. The element z^* selected for eviction is one with lowest estimated frequency

$$z^* = z_{\text{lef}} = \arg\min_{x \in \mathcal{S}} \{f(x)\}.$$

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

A first simple variation of SpaceSaving that we will call Randomized SpaceSaving (RSS) will behave as SS, but the addition of the current elements z, if not in $\mathcal{S}$, is decided a threshold $t^* > 0$. The simplest case is to have a fixed threshold, say $t^* = 0.95$, but more sophisticated schemes are also possible if the threshold t^* is dynamically computed, for example, t^* might be the mean value of the tokens providing "entrance" to the m current elements in the sample.

Generalizing the ideas above we move on a family of algorithms where the main distinctive feature is that for every element in the sample we keep a *ticket* $t(z)$ which is the maximum among the tokens generated for z. While in RSS variants we do not generate a token when the current element z in the data stream is already in the sample, in the algorithms that we will explore below tokens are generated for each element of the data stream, whether they already belong or not to the sample.

[1] The well-known SpaceSaving algorithm might be seen as a particular instance of this scheme with $t^* = 0$; therefore the generation of tokens is not necessary at all.

Let us know consider the first algorithm in this family, that we call 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 random nuber 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 $t(z) := t$; in other words, $t(z)$ is always the largest random number generated for z. If z is not in the sample, we compare t with the minimum ticket $t_{\min} = t(z_{\min})$ 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 $z_{\min}$, the one with the minimum ticket, and the frequency counter of z is initialized with

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

Notice that when z is added to the sample t must be the largest random number generated for z so far, hence, we also set $t(z) := t$. We will actually do that $(t(z) := t)$ whenever an element z enters the sample in all the variants that we consider below. However, in some of them, it is not true that $t(z)$ is the largest random number ever generated for z, and hence, it's not, rigorously speaking, the *ticket* of t. The property is true in those variants in which the minimum ticket $t_{\min}$ is always evicted when a new element z enters the sample, either because we evict $z_{\min}$ or becasue we swap tickets between $z\_textmin$ and z^* (*ticket stealing*).

LotterySaving-LowestEstimatedFrequency (LS-LEF) combines some of the ideas of the two algorithms above. When an item z is not in the sample we compare the random number t with the ticket $t_{\text{lef}} = t(z_{\text{lef}})$ of an element z_{lef} with smallest frequency counter (in case of ties, any of them is picked at random). Notice that t_{lef} might be or not the minimum ticket in the sample. If $t > t_{\text{lef}}$ then z is added and $z^* = z_{\text{lef}}$ is evicted, and $f_{\text{ini}}(z) = f(z^*)$.

LotterySaving-LeastRecentlyObserved (LS-LRO) is similar to LS-LEF, but the ticket t of a candidate element z to enter the sample is compared with the ticket $t_{\text{lro}} = t(z_{\text{lro}})$ of the item z_{lro} in the sample that was observed least recently (i.e., z_{lro} is the item with frequency counter updated the longest ago). Like in LS-LEF, if $t > t_{\text{lro}}$ then z is added and $z^* = z_{\text{lro}}$ is evicted. The counter of z is initialized using $f_{\text{ini}}(z) = f(z_{\text{lro}})$.

In LotterySaving-SmallestTicket (LS-ST) the element to be evicted is $z_{\min}$; but instead of the rule for initialization of f of LS it uses the estimated frequency f of z^*, like LS-LEF and LS-LRO. That is, when an item z is not in the sample we compare its ticket t with the smallest ticket $t_{\min} = t(z_{\min})$ of the element $z_{\min}$ with smallest ticket (w.l.o.g. we can safely assume that no two tickets ever are identical). If $t > t_{\min}$ then z is added and $z^* = z_{\min}$ is evicted, like in LS. But, unlike LS, we have $f_{\text{ini}}(z) = f(z^*)$.

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 $z^* = z_{\text{lef}}$ with minimum frequency counter and z inherits the frequency counter of z^* as the initial estimation: $f_{\text{ini}}(z) = f(z^*)$.

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 z_{lef}, one with smallest frequency counter, and z inherits $f(z^*)$ like in LS-LEF 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 algorithm uses $t_{(\lceil \alpha \cdot m \rceil)}$ to decide whether to include or not a new item in the sample, where $t_{(r)}$ denotes the r-th smallest ticket in the sample, $1 \leq r \leq m$.

The basic variants of LS-LEF and LS-LRO can be modified to incorporate the ideas behind LS-AM and LS-MED. Thus LS-LEF-AM and LS-LRO-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-LEF-MED LS-LRO-MED, or more generally, LS-LEF-QUANT-α and LS-LRO-QUANT-α.

Notice that it makes no sense to "combine" LS-AM and LS-QUANT with LS-ST since $t_{\min} < \bar{t}$ and $t_{\min} \leq t_{(r)}$ for all r.

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. We can derive corresponding variants of all the others discussed above were the items retain the ticket t with which they entered the sample, and never update it once in the sample, for instance.

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 $t_{\min}$. Recall that we can safely assume that all tickets in $\mathcal{S}$ are distinct.

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)
```

Algorithm	t^*	z^*	$f_ini(z)$
SS	0	z_{lef}	$f(z^*)$
LS	$t_{\min} = t(z^*)$	$z_{\min}$	$\lfloor \frac{1}{1-t^*} \rfloor$
LS-LEF	$t(z^*)$	z_{lef}	$f(z^*)$
LS-LRO	$t(z^*)$	z_{lro}	$f(z^*)$
LS-ST	$t(z^*)$	$z_{\min}$	$f(z^*)$
LS-THR	$\theta \cdot t_{(m)} = \theta \cdot t_{\max}$	z_{lef}	$f(z^*)$
LS-AM	$\bar{t} = \sum_{z \in \mathcal{S}} t(z)/m$	z_{lef}	$f(z^*)$
LS-QUANT	$t_{\lceil \alpha m \rceil}$	z_{lef}	$f(z^*)$
LS-LEF-AM	$\min\{\bar{t}, t(z^*)\}$	z_{lef}	$f(z^*)$
LS-LEF-MED	$\min\{t_{\lfloor (m+1)/2 \rfloor}, t(z^*)\}$	z_{lef}	$f(z^*)$

```
      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^* ($z_{\min}$ = the element with minimum t, z_{lef} = an element with minimum f, and z_{lro} = the least recently observed element); if we contemplate C different ways to define t^* ($C = 9$ in our table), we could consider then up to $6C$ different possible combinations—some make no sense. 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.

The three strategies LS-ST, LS-LEF and LS-LRO are almost identical, they only differ in the definition of the element z^* to be evicted. These three strategies will be called *canonical*, as all the other algorithms that we have discussed here can be seen as variantions of one of the three; maybe in the way that tickets are updated, maybe in the way that t^* is defined (e.g., $t^* \neq t(z^*)$), or the way that f is initialized (e.g., $f_ini(z) \neq f(z^*)$).

It is clear that the combinations that we can make "playing" around with the criteria for addition (`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-LEF 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.

Gonzalo: agrega a esta seccion otros algoritmos que estes probando. En el documento final es posible que hayas de hacer una "purga", eliminando las opciones que no han dado buenos resultados. Sin embargo, SpaceSaving, Lottery Sampling y las tres variantes canonicas LS-ST, LS-LEF y LS-LRO deben estar siempre, como referencia. Para nuevas variantes inventa nombres y acronimos. Tambien habriamos de pensar algun modo facil de indicar si una estrategia es con o sin update de tickets, en vez de inventar nuevos nombres. Por ejemplo, LS*-LEF es la variante de LS-LEF sin hacer update de los tickets.
Por cierto que nn una primera impresion yo diria que LS*-LEF es casi lo mismo que SpaceSaving pero lanzando una moneda e incluyendo al item z nuevo en el sample con probabilidad $m/(m+1)$ (con probabilidad $1/(m+1)$ el item nuevo no reemplaza a z_{lef}).

2.1 Implementing the Algorithms

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. Likewise, when the task at hand is to report heavy hitters, i.e., elements with relative frequency above a given threshold c, we might not report some heavy hitters (or even report non-heavy hitters as if they were; this might happen for algorithms working with overestimations of the real frequencies).

In order to measure the quality of our algorithms and other contenders, we will introduce several metrics. The first two metrics which we will introduce are inspired by the very well known *recall* and *precision* from Information Retrieval (IR), and we will use the same names. These two metrics measure the quality of the answer to a query to an IR system. Some of the documents retrieved by the system will not be "relevant" to the user, whereas some of the relevant documents will not be retrieved. Then

$$\texttt{recall} = \frac{\#\text{ of retrieved relevant documents}}{\#\text{ of relevant documents}}$$

$$\texttt{precision} = \frac{\#\text{ of retrieved relevant documents}}{\#\text{ of retrieved documents}}$$

In what follows we discuss the metrics that we have used to measure the quality of the results of the algorithms introduced here and compare it with the quality of competitors such as SpaceSaving. The following random variables will be essential for our definitions below:

1. The indicator random variables Y_i, $1 \leq i \leq n$, telling us whether x_i is returned as a result or not, that is, $Y_i = 1$ if x_i is returned as a result of the algorithm (as a potential top-k most frequent or a heavy hitter), and $y_i = 0$ otherwise.

2. For top-k queries, R_i will denote the *rank* of the element that the algorithm reports as i-th most frequent. For any element x, rank(x) is defined as 1 plus the number of elements in the stream with frequency strictly larger than x. Thus, if all frequencies were distinct, rank(x_i) $= i$, for all i, $1 \leq i \leq n$. We also introduce ρ_i and F_i, the actual relative and absolute frequency, respetively, of the element reported as the i-th most frequent by the algorithm. Notice that $\rho_i = p_{R_i}$, and $F_i = f_{R_i}$.

3. For any element x_i, we denote f_i' the estimated absolute frequency of x_i (by convention, we set $f_i' = 0$ if x_i is not in the sample). Likewise, $p_i' = f_i'/N$ will denote the estimated relative frequency of x_i.

3.1 Recall & Precision

In the case of top-k queries, since the number of returned elements and the number of relevant elements are both equal to k, recall and precision are identical. We will hence only work with recall (or precision, whatever name serves):

$$R_u = \frac{Y_1 + Y_2 + \cdots + Y_k}{k}.$$

Indeed, the top-k most frequent (relevant) elements are x_1, x_2, ..., x_k and $Y_1 + Y_2 + \cdots + Y_k$ is the number of returned elements which are among the top-k most frequent.

One drawback of this metric is that it penalizes in exactly the same amount the miss of x_i, $1 \leq i \leq k$; however, it is reasonable that we consider as a better algorithm one that misses x_k (the k-th most frequent element) than one that missed x_1 (the most frequent element). Hence we introduce weighted recall:

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

our first metric R_u will be called *unweighted recall*.

In heavy hitter queries with threshold c, $0 < c < 1$, the number of returned elements and the number of relevant elements are different. The first is given by

$$Y_1 + \cdots + Y_n,$$

whereas the second is k^*, the largest index such that $p_{k^*} \geq c$. Hence the recall metrics (unweighted and weighted) are

$$R_u = \frac{Y_1 + \cdots + Y_{k^*}}{k^*}$$
$$R = \frac{p_1 Y_1 + \cdots + p_{k^*} Y_{k^*}}{p_1 + \cdots + p_{k^*}}$$

The (unweighted) precision is

$$P_u = \frac{Y_1 + \cdots + Y_{k^*}}{Y_1 + \cdots + Y_n},$$

and a weighted precision can also be introduced as

$$P = \frac{p_1 Y_1 + \cdots + p_k Y_{k^*}}{p_1 Y_1 + \cdots + p_n Y_n}.$$

Notice that in all cases the metrics adopt values in $[0, 1]$; lower values indicate poorer quality of the returned results, with relevant elements missing in the result and/or irrelevant elements (not among the top-k most frequent, not heavy hitters) reported as "relevant". Values near 1 correspond to very good approximations to the exact answer. Take for instance P. Since

$$p_1 Y_1 + \cdots + p_k Y_{k^*} \leq p_1 Y_1 + \cdots + p_n Y_n,$$

it is clear that its value will be always $P \leq 1$. If no heavy hitter were reported then $p_1 Y_1 + \cdots + p_k Y_{k^*} = 0$ and $P = 0$. Suppose that some algorithm A reports k' elements, of which only $k'' \leq k^*$ are heavy hitters, say $x_{i_1}, \ldots, x_{i_{k''}}$, with $1 \leq i_1 \leq i_2 \leq i_{k''} \leq k^*$. Then if

$$v = p_{i_|} + \cdots + p_{i_{k''}}$$

we will have

$$P = \frac{v}{v + w},$$

where $w \geq 0$ is the sum of the relative frequencies of the other $k' - k''$ returned elements which are not heavy hitters. If $k' = k''$ then $w = 0$ and the precision is 1: all reported elements are heavy hitters. However, the recall R might be less than 1 as some heavy hitters might have not been reported, i.e., $k'' \neq k^*$.

We also remark here that in heavy hitter queries, since all relevant elements have relative frequency $\geq c$ by definition, there will be many applications where all elements will be considered equally relevant. Hence, the weighted recall and weighted precision metrics might be of lesser interest in those cases.

3.2 Precision-like Metrics

La métrica que has venido denominando "discrete precision" es el *unweighted recall* explicado en el apartado anterior? En caso contrario, qué otras métricas has usado en los estudios de top-k most frequent? Completar esta subsección o eliminarla, según proceda.

3.3 Order

In the top-k most frequent problem, it might be often the case that the order in which elements are reported matters. So we will have metrics to measure how different is the real rank R_i of the i-th reported element from the pretended rank (i). Notice that the errors in the ordering typically come from bad estimates of the actual frequencies of the elements. Most algorithms top-k will keep a "sample" of $m \leq k$ elemnts from the stream, gather statistics for the sampled elements, and report the first k elements of the sample when sorted in non-increasing order of **estimated** frequencies. But it makes sense to consider metrics which take into account only the order induced by the estimated frequencies and not the estimated frequencies themselves (see the subsection below). For instance an algorithm might systematically and grossly overestimate the real frequencies of the sample elements, but these estimates might "preserve" well enough the real relative ordering of the elements.

Ninguna de estas medidas esta normalizada. Habra que estudiar cuales son "relevantes" (tienden a 0 en algoritmos malos, tienden a 1 en algoritmos buenos, discriminan entre algoritmos diferentes), y cuales no. Hay que agregar tambien la variante que propones. A continuacion, las metricas, a palo seco.

$$e = \sum_{1 \leq i \leq k} p_i R_i e' \quad = \sum_{1 \leq i \leq k} p_i \left(1 - p_{R_i}\right) IC_u = \sum_{1 \leq i \leq k} (R_i - i)^2 IC \quad = \sum_{1 \leq i \leq k} p_i (R_i - i)^2 \tag{1}$$

3.4 Estimated frequencies

Finally we consider metrics which measure the quality of the estimated frequencies. We discuss first these metrics in the context of top-k most frequent queries.

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

$$\epsilon_I = \frac{\sum_{1 \leq i \leq k} (f_i - f'_i)^2}{\sum_{1 \leq i \leq k} f_i^2}.$$

For reasons that will be discussed later we might also take $f'_i := \min\{f'_i, 2f_{\text{obs}}(x_i)\}$. Since $f_{\text{obs}}(x_i) \leq f_i$ 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 are reported, or how good or bad are the estimates of their corresponding 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 elements 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 result:

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

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).

Revisa las definiciones para ver si son correctas y se adecuan con las que usas en los experimentos. Agrega nuevas medidas que estes usando y se te ocurran. Ya habra ocasion de "purgar" aquellas que resulten poco relevantes, redundantes o demasiado ad-hoc (como comentamos tiene que haber un fundamento racional que justifique la definicion de la medida, no vale que de los resultados que nos gustan :-))
R_u y P_u las versiones no ponderadas de *recall* y *precision* no son las mas utiles, pero han de estar y aparecer los valores correspondientes en la parte experimental. En el paper de SS, son las medidas que se usan y estara bien hacer la comparativa en base a estas dos medidas tambien.

4 Analysis of the Algorithms

5 Experimental Results

Esta sección (que ocupará una parte importante de este documento y también en tu TFG) se dividira en no menos de tres partes:

- Descripción de los conjuntos de datos reales y sintéticos usados en los experimentos
- Descripción del *setup* experimental: cuantas veces se repetia el experimento, entorno de trabajo, etc.
- Presentación (tablas, gráficas) y análisis de los resultados experimentales sobre calidad de los algoritmos (recall, precision, errores de estimacion de frecuencias, ...)
- Presentación (tablas, gráficas) y análisis de los resultados experimentales sobre rendimiento (tiempo de proceso, memoria,...)