11 Heavy Hitters

A core mining problem is to find items that occur more than one would expect. These may be called outliers, anomalies, or other terms. Statistical models can be layered on top of or underneath these notions.

We begin with a very simple problem. There are m elements and they come from a domain $[n]$ (but both m and n might be very large, and we don't want to use $\Omega(m)$ or $\Omega(n)$ space). Some items in the domain occur more than once, and we want to find the items which occur the most frequently.

If we can keep a counter for each item in the domain, this is easy. But we will assume n is huge (like all possible IP addresses), and m is also huge, the number of packets passing through a router in a day.

11.1 Streaming

Streaming [5] is a model of computation that emphasizes *space* over all else. The goal is to compute something using as little storage space as possible. So much so that we cannot even store the input. Typically, you get to read the data once, you can then store something about the data, and then let it go forever! Or sometimes, less dramatically, you can make 2 or more passes on the data.

Formally, there is a stream $A = \langle a_1, a_2, \ldots, a_m \rangle$ of m items where (for this lecture) each $a_i \in [n]$. This means, the size of each a_i is about $\log n$ (to represent which element), and just to count how many items you have seen requires space $\log m$ (although if you allow approximations you can reduce this). Unless otherwise specified, $\log$ is used to represent $\log_2$ that is the base-2 logarithm. The goal is to compute a summary S_A using space that is only $\mathsf{poly}(\log n, \log m)$.

Let $f_j = |\{a_i \in A \mid a_i = j\}|$ represent the number of items in the stream that have value j. Let $F_1 = \sum_j f_j = m$ be the total number of elements seen. Let $F_2 = \sqrt{\sum_j f_j^2}$ be the sum of squares of elements counts, squarerooted. Let $F_0 = \sum_j f_j^0$ be the number of distinct elements.

11.2 Majority

One of the most basic streaming problems is as follows:

MAJORITY: if some $f_j > m/2$, output j. Otherwise, output anything.

How can we do this with $\log n + \log m$ space (one counter c, and one location ℓ)?

Answer: Maintaining that single label and counter, do the only thing feasible. If you see a new item with same label, increment the counter. If the label is different, decrement the counter. If the counter reaches zero and you see a new element, replace the label, and set the counter to 1. The pseudocode is in Algorithm 11.2.1.

Algorithm 11.2.1 Majority(A)

```
Set $c = 0$ and $\ell = \emptyset$
for $i = 1$ to $m$ do
  if $(a_i = \ell)$ then
    $c = c + 1$
  else
    $c = c - 1$
  if $(c < 0)$ then
    $c = 1, \ell = a_i$
return $\ell$
```

Why is Algorithm 11.2.1 correct? Consider the case where for some $j \in [n]$ we have $f_j > m/2$, the only relevant case. Since then $f_j > \sum_{j' \neq j} f_j$ we can match each stream element with $a_i \neq j$ (a "bad element") to another element $a_{i'} = j$ (a "good element"). If we chose the correct pairing (lets assume we did) then either the good element decremented the counter when the label was not j, or the bad element decremented the counter when the label was j. This results in a net 0 change in the counter for each pair. Its also possible that a bad element decremented the counter when the label was not equal to j, but this will only help. After this cancelation, there must still be unpaired good elements, and since then the label would need to be $\ell = j$ or the counter $c = 0$, they always end their turn with $\ell = j$ and the counter incremented. Thus after seeing all stream elements, we must terminate with $\ell = j$ and $c > 0$.

11.3 Misra-Gries Algorithm for Heavy Hitters

Now we generalize the MAJORITY problem to something much more useful.

k-FREQUENCY-ESTIMATION: Build a data structure S. For any $j \in [n]$ we can return $S(j) = \hat{f}_j$ such that

$$f_j - m/k \leq \hat{f}_j \leq f_j.$$

From another view, a ϕ-*heavy hitter* is an element $j \in [n]$ such that $f_j > \phi m$. We want to build a data structure for ε-approximate ϕ-heavy hitters so that it returns

- all f_j such that $f_j > \phi m$
- no f_j such that $f_j < \phi m - \varepsilon m$
- (any f_j such that $\phi m - \varepsilon m \leq f_j < \phi m$ can be returned, but might not be).

11.3.1 Misra-Gries Algorithm

[Misra+Gries 1982] Solves k-FREQUENCY-ESTIMATION in $k(\log m + \log n)$ space [4].

The trick is to run the MAJORITY algorithm, but with $(k-1)$ counters instead of 1. Let C be an array of $(k-1)$ counters $C[1], C[2], \ldots, C[k-1]$. Let L be an array of $(k-1)$ locations $L[1], L[2], \ldots, L[k-1]$.

- If we see a stream element that matches a label, we increment the associated counter.
- If not, and a counter is 0, we can reassign the associated label, and increment the counter.
- Finally, if all counters are non-zero, and no labels match, then we decrement all counters.

Psuedocode is provided in Algorithm 11.3.1.

Algorithm 11.3.1 Misra-Gries(A)

```
Set all $C[i] = 0$ and all $L[i] = \emptyset$
for $i = 1$ to $m$ do
  if ($a_i = L[j]$) then
    $C[j] = C[j] + 1$
  else
    if (some $C[j] = 0$) then
      Set $L[j] = a_i$ & $C[j] = 1$
    else
      for $j \in [k-1]$ do $C[j] = C[j] - 1$
return $C$, $L$
```

Then on a query $q \in [n]$ to C, L, if $q \in L$ (specifically $L[j] = q$), then return $\hat{f}_q = C[j]$. Otherwise return $\hat{f}_q = 0$.

Analysis: Why is Algorithm 11.3.1 correct?

- A counter $C[j]$ representing $L[j] = q$ is only incremented if $a_i = q$, so we always have
$$\hat{f}_q \leq f_q.$$
- If a counter $C[j]$ representing $L[j] = q$ is decremented, then $k-2$ other counters are also decremented, and the current item's count is not recorded. This happens at most m/k times: since each decrement destroys the record of k objects, and since there are m objects total. Thus a counter $C[j]$ representing $L[j] = q$ is decremented at most m/k times. Thus
$$f_q - m/k \leq \hat{f}_q.$$

We can now apply this to get an additive ε-approximate FREQUENCY-ESTIMATION by setting $k = 1/\varepsilon$. We return $\hat{f}_q$ such that
$$|f_q - \hat{f}_q| \leq \varepsilon m.$$
Or we can set $k = 2/\varepsilon$ and return $C[j] + (m/k)/2$ to make error on both sides.

Space is $(1/\varepsilon)(\log m + \log n)$, since there are $(1/\varepsilon)$ counters and locations.

11.4 Quantiles

Another important but simple problem for streaming data is the *quantiles* problem. For this we consider the ordering of the elements in $[n]$ as important. In fact, its typically easier to think of each element being a real value $a_i \in \mathbb{R}$ so that they are continuously valued and we have a comparison operator $<$. Think of a_i as the number of milliseconds someone spent on a visit to a website before clicking a link. Or a_i could be the amount of money spent on a transaction.

Now instead of searching for frequently occurring items (since we may never see the same item twice) it is better to treat these as draws from a continuous distribution over $\mathbb{R}$. In this case, two very similar (but perhaps not identical) values are essentially equivalent. The simplest well-defined interaction with such a distribution is through the associated cumulative density function. That is, given any value v, we can ask what fraction of items have value less than or equal to v. We can define the *rank* of v over a stream A as
$$\mathsf{rank}_A(v) = |\{a_i \in A \mid a_i \leq v\}|.$$
Now an *ε-approximate quantiles data structure* Q_A returns a value $Q_A(v)$ for all v such that
$$|Q_A(v) - \mathsf{rank}_A(v)/m| \leq \varepsilon.$$

If we are not concerned about streaming, we can easily construct a data structure of size $1/\varepsilon$. We simply sort all values in A, and then select a subset B of size $1/\varepsilon$ elements, evenly spaced in that sorted order. Then $Q_A(v) = \mathsf{rank}_B(v)/|B|$. This is the smallest possible in general. Streaming algorithms are known of size $O((1/\varepsilon)\log\log(1/\varepsilon))$ [2] (which is the smallest possible size) and efficient variants of size $O((1/\varepsilon)\log^{1.5}(1/\varepsilon))$ [6, 7]. By combining two such queries, we can also ask what fraction of data falls between two values v_1 and v_2 as $Q_A(v_2) - Q_A(v_1)$.

Additionally, such a summary also encodes properties like the approximate median. This is the value for which $\mathsf{rank}_A(v)/m = 0.5$ (naively one may have to find this by binary search, if the structure is a set B and $Q_A(v) = \mathsf{rank}_B(v)/|B|$, then we can also maintain this directly. In addition to a basic quantiles sketch, we will describe a simpler "frugal" variant [3] which can maintain values like the approximate median (or any other quantile) approximately without maintaining all quantiles. Going backwards from the Misra-Gries to Majority, is akin to the idea of maintain just a Frugal sketch of the median as opposed to all quantiles.

11.4.1 Merging Quantiles

The key idea in efficient quantiles sketches are being able to merge two sketches without increasing the error or the size [1]. For quantiles, this works be a simple procedure describe below, where each of the Q_A data structures is simply a set B which returns $\textsf{rank}_B(v)/|B|$.

Now given two such sets B_1 and B_2, both of size s, representing sets A_1 and A_2 both of size t. To merge the summaries, we let $B = B_1 \cup B_2$. Its not hard to see that if

$$|\textsf{rank}_{A_1}(v)/t - \textsf{rank}_{B_1}(v)/s| \leq \varepsilon \quad \text{and} \quad |\textsf{rank}_{A_2}(v)/t - \textsf{rank}_{B_2}(v)/s| \leq \varepsilon,$$

then by examining $\textsf{rank}_A(v)$ and $(t/s)\textsf{rank}_B(v)$,

$$|\textsf{rank}_A(v)/t - \textsf{rank}_B(v)/s| \leq \varepsilon.$$

However, $|B| = 2s$, so the size has doubled. To reduce the size we do the following simple step. We sort B, and let B_e be all points in B with even indices, and let B_o be all points in B with odd indices. Then we let the new sketch B' be either B_e or B_o, chosen at random.

With this sketch, we don't expect (in the expected value sense) to over- or under-count any rank query. But the process still seems like it should add some error. It turns out not too much is added, since the previous levels induce far less additive error than the current ones. Moreover, if we increase the sketch size s from $1/\varepsilon$ to $k_\varepsilon = O((1/\varepsilon)\sqrt{\log(1/\varepsilon)\log(\delta)})$, then with probability at least $1 - \delta$, the error is never more than ε after any number of merges.

However, this requires that we only merge summaries B_1 and B_2 that represent exactly the same size sets A_1 and A_2. To deal with this issue, each summary will actually store up to $g_\varepsilon = O(\log(m\varepsilon))$ sets, where the jth set $B(j)$ (if it exists) represents a set of size $m/2^j$ for some $j \in [0, g_\varepsilon]$. Then on a merge, starting at the large-index layers, we merge pairwise (if there is more than one of some type), and push the merged sketches on up the representation, potentially increasing the height of the structure g_ε by 1. Or in the streaming setting, we can just add a single point to a buffer of size k_ε, then merge with the bottom ($j = g_\varepsilon$) layer.

This takes overall space $O((1/\varepsilon)\log(\varepsilon m)\sqrt{\log 1/\varepsilon}\log(1/\delta))$ to guarantee on a data set of size m, that the normalized rank has at most ε error, with probability at least $1 - \delta$. With some care to how the hierarchy is managed, the size can be reduced to $O((1/\varepsilon)\log\log(1/\varepsilon)\log(1/\delta))$ [1, 2].

11.4.2 Frugal Median

The Frugal estimate of the median can be maintained easily as followed over an ordered set of integers. The simplest version just maintains a single label $\ell \in [n]$. Initially set $\ell = 0$ (or any value). Then if $a_i > \ell$, then increment ℓ. If $a_i < \ell$, then decrement ℓ. Psuedocode is in Algorithm 11.4.1.

Algorithm 11.4.1 Frugal Median(A)

```
Set $\ell = 0$.
for $i = 1$ to $m$ do
  if ($a_i > \ell$) then
    $\ell \leftarrow \ell + 1$.
  if ($a_i < \ell$) then
    $\ell \leftarrow \ell - 1$.
return $\ell$.
```

This can be generalized to any quantile, say trying to find just the value v such that $\textsf{rank}_A(v)/m = 0.75$. Then we use a bit of randomization.

Algorithm 11.4.2 Frugal Quantile(A, ϕ) e.g. $\phi = 0.75$

```
Set $\ell = 0$.
for $i = 1$ to $m$ do
    $r = \mathsf{Unif}(0, 1)$ (at random)
    if ($a_i > \ell$ and $r > 1 - \phi$) then
        $\ell \leftarrow \ell + 1$.
    if ($a_i < \ell$ and $r > \phi$) then
        $\ell \leftarrow \ell - 1$.
return $\ell$.
```

The bounds for this algorithm are not as absolutely strong as for the Misra-Gries algorithm, but it uses far less space. For instance, for the median version let M be the integer value of the true median, and say we are happy with any value v such that $\mathsf{rank}_A(v)/m \in [1/2 - \varepsilon, 1/2 + \varepsilon]$ for some small value $\varepsilon \in (0, 1/2)$. The with probability at least $1 - \delta$ after $\frac{M \log(1/\delta)}{\varepsilon}$ steps, our estimate will be within the desired range.

The bounds are better if we start our estimate at a value closer to v^* than 0. Also, if we are using an extra small counter, then we can adaptively change the amount we increment or decrement the label, and decrease the number of steps we need.

Bibliography

[1] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(26), 2013.

[2] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *FOCS*, 2016.

[3] Qiang Ma, S. Muthukrishnan, and Mark Sandler. Frugal streaming for estimating quantiles. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume LNCS 8066, pages 77–96, 2013.

[4] J. Misra and D. Gries. Finding repeated elements. *Sc. Comp. Prog.*, 2:143–152, 1982.

[5] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[6] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. Quantiles over data streams: An experimental study. In *ACM SIGMOD International Conference on Management of Data*, 2013.

[7] Yahoo. Data sketches library, 2016.