12 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.

12.1 Streaming

Streaming [3] 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 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 function $g(A)$ 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.

12.2 Majority and Heavy Hitters

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)?

Algorithm 12.2.1 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 ($c \leq 0$) then
    $c = 1, \ell = a_i$
return $\ell$
```

Why is Algorithm 12.2.1 correct? If $f_j > m/2$, then

- if ($\ell \neq j$) then c decremented at most $< m/2$ times, but $c > m/2$
- if ($\ell = j$) can be decremented $< m/2$ times, but incremented $> m/2$ times.

On the other hand, if $f_j < m/2$ for all j, then any answer is ok.

12.2.1 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).

12.2.2 Misra-Gries Algorithm

[Misra+Gries 1982] Solves k-FREQUENCY-ESTIMATION in $k(\log m + \log n)$ space [2].

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 $L[1], L[2], \ldots, L[k-1]$.

Algorithm 12.2.2 Misra-Gries(A)

```
Set all $C[i] = 0$ and all $L = \emptyset$
for $i = 1$ to $m$ do
  if ($a_i \in L$) (at index $j$) then
    $C[j] = C[j] + 1$
  else
    if ($|L| < k - 1$) then
      For some $j$ with $L[j] = \emptyset$: $C[j] = 1$ & $L[j] = a_i$
    else
      for $j \in [k-1]$ do $C[j] = C[j] - 1$
  for $j \in [k-1]$ do
    if ($C[j] \leq 0$) do $L[j] = \emptyset$
  if ($|L| < k - 1$ & $a_i \notin L$) then
    For some $j$ where $C[j] = 0$: $C[j] = 1$ & $L[j] = a_i$
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 12.2.2 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, this process only happens at most m/k times. 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.

12.2.3 Count-Min Sketch

We now describe a completely different way to solve the HEAVY-HITTER problem, called the Count-Min Sketch [Cormode + Muthukrishnan 2005] [1].

Start with t independent (random) hash functions $\{h_1, \ldots, h_t\}$ where each $h_i : [n] \to [k]$.

Now we store an 2d array of counters for $t = \log(1/\delta)$ and $k = 2/\varepsilon$:

h_1	$C_{1,1}$	$C_{1,2}$	$\ldots$	$C_{1,k}$
h_2	$C_{2,1}$	$C_{2,2}$	$\ldots$	$C_{2,k}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
h_t	$C_{t,1}$	$C_{t,2}$	$\ldots$	$C_{t,k}$

Algorithm 12.2.3 Count-Min(A)

```
Set all $C_{i,j} = 0$
for $i = 1$ to $m$ do
  for $j = 1$ to $t$ do
    $C_{j,h_j(a_i)} = C_{j,h_j(a_i)} + 1$
```

After running Algorithm 12.2.3 on a stream A, then on a query $q \in [n]$ we can return

$$\hat{f}_q = \min_{j \in [t]} C_{j,h_j(q)}.$$

This is why it is called a *count-min sketch.*

Analysis: Clearly $f_q \leq \hat{f}_q$ since each counter has everything for q, but may also have other stuff (on hash collisions).

Next we claim that $\hat{f}_q \leq f_q + W$ for some over count value W. So how large is W?

Consider just one hash function h_i. It adds to W when there is a collision $h_i(q) = h_i(j)$. This happens with probability $1/k$.

So we can create a random variable $Y_{i,j}$ that represents the overcount caused on h_i for q because of element $j \in [n]$. That is, for each instance of j, it increments W by 1 with probability $1/k$, and 0 otherwise. Each instance of j has the same value $h_i(j)$, so we need to sum up all these counts. Thus

- $Y_{i,j} = \begin{cases} f_j & \text{with probability } 1/k \\ 0 & \text{otherwise.} \end{cases}$
- $\mathbf{E}[Y_{i,j}] = f_j/k.$

Then let X_i be another random variable defined

- $X_i = \sum_{j \in [n], j \neq q} Y_{i,j}$, and
- $\mathbf{E}[X_i] = \mathbf{E}[\sum_{j \neq q} Y_{i,j}] = \sum_{j \neq q} f_j/k = F_1/k = \varepsilon F_1/2.$

Now we recall the Markov Inequality. For a random variable X and a value $\alpha > 0$, then $\textbf{Pr}[|X| \geq \alpha] \leq \textbf{E}[|X|]/\alpha$. Since $X_i > 0$, then $|X_i| = X_i$, and set $\alpha = \varepsilon F_1$. And note $\textbf{E}[|X|]/\alpha = (\varepsilon F_1/2)/(\varepsilon F_1) = 1/2$. It follows that

$$\textbf{Pr}[X_i \geq \varepsilon F_1] \leq 1/2.$$

But this was for just one hash function h_i. Now we extend this to t *independent* hash functions:

$$\begin{aligned}\textbf{Pr}[\hat{f}_q - f_q \geq \varepsilon F_1] &= \textbf{Pr}[\min_i X_i \geq \varepsilon F_1] = \textbf{Pr}[\forall_{i\in[t]}(X_i \geq \varepsilon F_1)] \\ &= \prod_{i\in[t]} \textbf{Pr}[X_i \geq \varepsilon F_1] \leq 1/2^t = \delta,\end{aligned}$$

since $t = \log(1/\delta)$.

So that gives us a PAC bound. The Count-Min Sketch for any q has

$$f_q \leq \hat{f}_q \leq f_q + \varepsilon F_1$$

where the first inequality always holds, and the second holds with probability at least $1 - \delta$.

Space. Since there are kt counters, and each require $\log m$ space, then the total counter space is $kt \log m$. But we also need to store t hash functions, these can be made to take $\log n$ space each. Then since $t = \log(1/\delta)$ and $k = 2/\varepsilon$ it follows the overall total space is $t(k \log m + \log n) = ((2/\varepsilon) \log m + \log n) \log(1/\delta)$.

Turnstile Model: There is a variation of streaming algorithms where each element $a_i \in A$ can either add one or subtract one from corpus (like a turnstile at the entrance of a football game), but each count must remain positive. This Count-Min has the same guarantees in the turnstile model, but Misra-Gries does not.

Bibliography

[1] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55:58–75, 2006.

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

[3] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 2003.