CS 395T: Sublinear Algorithms, Fall 2020 September 22nd, 2020

Lecture 7: Heavy Hitters

Prof. Eric Price *Scribe: Aditya Parulekar, Advait Parulekar*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the last lecture we talked about quantile estimation with two methods, sampling and merge-and-reduce.

In this lecture we will do streaming algorithms to determine the elements that appear the most often in a stream - the Heavy Hitters problem.

2 Heavy Hitters

Given a stream of URLs $u_1, u_2, \ldots, u_N \in U$. Let $x_u = \sum_{i=1}^{N} \mathbb{1}_{u_i=u}$, that is, the number of occurrences of u in the stream. The goal is to store the elements in the universe that occur the most often. For instance, the universe could be the set of URLs, and we might be interested in the most common ones. In many domains, (for example, words in book, # of friends, etc), the ith most common element occurs with frequency decaying as a power law, for instance proportional to $1/i^{0.7}$. This motivates the interest in the elements that occur the most often, as sometimes they can occur considerably more often than the other elements.

We could solve our problem by keeping track of how often each element occurs, however, by a reduction from the indexing problem we can see that solving this exactly would require $\Omega(n)$ space. Instead we focus on the following:

1. **Estimation version:** output $\tilde{x}_u$ such that $|\tilde{x}_u - x_u|$ is small with high probability.

$$|\hat{x}_u - x_u| \leq \frac{N}{k} = \frac{\|x\|_1}{k}$$

2. **Heavy Hitters version:** Keep track of just the most common elements, without caring about exactly how often they occur. That is, find $S \subseteq [n]$ such that

   (a) $u \in S$ for all u such that $x_u \geq \frac{2N}{k}$

   (b) $u \notin S$ for all u such that $x_u < \frac{N}{k}$

   Note that this means $|S| \leq k$.

2.1 Frequent Elements and Misra-Gries

Here we look at a deterministic algorithm to get estimates $\tilde{x}_u$ in the insertion only model.

Lets first look at the problem of identifying a single majority element, if it exists. If a majority does not exist, we may output any element.

To solve this, we keep track of a candidate element, and a counter. We initialize the counter to 1, and the candidate element to be the first element in the stream. Whenever the next element in the stream is the candidate element, we increment the counter by 1. Whenever the next element is different, we decrement the counter by 1. If the counter ever hits 0, we replace the candidate element by the most recent element, and set the counter to 1 again. We output the candidate element.

For example, suppose the stream was "A", "B", "B", "C", "A", "B", "B". Then after each element was seen our state would be ("A", 1), ("B", 1), ("B", 2), ("B", 1), ("A", 1), ("B", 1), ("B", 2). We output "B", which is the correct majority.

We can extend this to keep track of more elements. Suppose we kept track of the number of times each element occurs. That is, we store a map from elements to their count, and increment the count for an element whenever we it. If there are more than k elements, we remove an element. We cannot just remove the element with the smallest count (see what happens when we are limiting ourselves to keeping track of 3 elements and we see the stream ("A", "B", "C", "D", "D") - we will not get D). Instead, if we overflow, we decrement each counter currently in our map by 1. If any counter reaches 0, we delete that element and insert our new element with count 1.

Algorithm 1: Misra-Gries

```
   Input  : k
   Output: Table
 1 Table ← {};
 2 for element in stream do
 3 |   if element in Table then
 4 |   |   Table [element ]+ = 1;
 5 |   else
 6 |   |   if |Table | ≤ k then
 7 |   |   |   Table [element ] = 1
 8 |   |   else
 9 |   |   |   for i ∈ [k] do
10 |   |   |   |   Table [i]− = 1;                    // decrement step
11 |   |   |   |   if Table [i] = 0 then
12 |   |   |   |   |   Table \ i;
```

We use the counts in **Table** once all of the elements have been seen as our estimates $\tilde{x}_u$.

Claim 1. *Upon termination of Algorithm (1), for each element u, we have*

$$x_u - \frac{N}{k} \leq \tilde{x}_u \leq x_u$$

Furthermore, if the stream restricted to only its a most frequent elements is denoted $H_a(x)$, we have

$$x_u - \frac{2||x - H_{\frac{k}{2}}(x)||_1}{k} \leq \tilde{x}_u \leq x_u$$

Proof. The counts $\tilde{x}_u$ for each element u are always underestimates of the true frequency x_u, since the element may have been in the **Table** when **decrement step** was executed. In particular, if the **decrement step** is executed D times over the course of the entire algorithm, then it was executed at most D times when element u was in the **Table**, and so we have

$$x_u - D \leq \tilde{x}_u \leq x_u$$

Furthermore, since each **decrement step** decreases k from the total count of all elements in the table, we can only execute **decrement step** at most

$$D \leq \frac{N}{k} = \frac{||x||_1}{k}$$

times. If we keep track of the total count across every entry in the table, $\sum_u \tilde{x}_u$, we know that we reduce this count by k on every **decrement step** and increase the count by 1 for every insertion, so we have exactly that

$$\sum_u \tilde{x}_u + Dk = \sum_u x_u = ||x||_1.$$

We can say more in terms of how concentrated the stream is around its most frequent elements. Let $H_a(x)$ denote the stream restricted to only its a most frequent elements, which we denote $u_1, u_2, \cdots, u_a$. We have

$$\begin{aligned}
\sum_{i=1}^{\frac{k}{2}} x_{u_i} &\leq \sum_{i=1}^{\frac{k}{2}} \tilde{x}_{u_i} + \frac{Dk}{2} \\
&\leq \sum_u \tilde{x}_u + \frac{Dk}{2} \\
&= \sum_u x_u - \frac{Dk}{2} \\
\implies \frac{Dk}{2} &= \sum_u x_u - \sum_{i=1}^{\frac{k}{2}} x_{u_i} \\
\implies D &\leq \frac{2}{k}||x - H_{\frac{k}{2}}(x)||_1
\end{aligned}$$

□

2.2 Count-Min

Now, we would like to solve the problem for the general case, including deletions: that is, you are given updates (u, α) in a stream, which corresponds to the operation $x_u \leftarrow x_u + \alpha$, where α is not necessarily nonnegative.

To do this, recall how hash tables operate: you store an array of values. The index in this array that you operate on given a certain input is given by a hash function on the input, and collisions are handled by "chaining". So, a potential solution would be to create a hash table on the inputs, and update the value in the hash table corresponding to this hash. However, this requires linear space in the number of inputs, and so is too much storage.

So, instead, simply don't do chaining, and allow colliding hash values to use the same entry in your hash table. In particular, take a pairwise independent hash function $h : U \to [B]$, where $B = \Theta(k)$, and let

$$y_j = \sum_{u:h(u)=j} x_u$$

Estimate $\hat{x}_u$ as $y_{h(u)}$. Note that in strict turnstile, $x_u \leq \hat{x}_u$. Further, the error is, in expectation, similar to what we had earlier:

$$\mathbb{E}[\hat{x}_u - x_u] = \mathbb{E}\left[\sum_{u \neq v} x_v \cdot 1_{h(v)=h(u)}\right] = \sum_{v \neq u} x_v \, \mathbb{P}[h(v) = h(u)] \leq \frac{\|x\|_1}{B} = \frac{N}{B}$$

where we have used that the hash function is pairwise independent.

However, the issue with this is that $\|\hat{x} - x\|_\infty$ is large, since many small values could hash to the same location as the largest ones, causing lots of error on those coordinates.

So, as usual, to fix this, we repeat this process R times, with R hash functions, getting $\hat{x}^{(1)}, \hat{x}^{(2)}, \hat{x}^{(3)}, \ldots \hat{x}^{(R)}$ such that

$$\forall u, \forall r \in [R], \mathbb{E}[\hat{x}_u^{(r)} - x_u] \leq \frac{N}{B}$$

Now, all of these estimates are going to be larger than the true value, and so the minimum of them is likely to be close to the true value.

Claim 2. *If we take $\hat{x}_u = \min_r \hat{x}_u^{(r)}$, then*

$$\mathbb{P}\left[\|\hat{x} - x\|_\infty \geq \frac{2N}{B}\right] \leq |U| 2^{-R}$$

Proof. By Markov's inequality, we have that for some fixed r,

$$\mathbb{P}\left[\hat{x}_u^{(r)} - x_u \geq \frac{2N}{B}\right] \leq \frac{1}{2}$$

So, since all of the R instances are independent,

$$\mathbb{P}\left[\hat{x}_u - x_u \geq \frac{2N}{B}\right] = \mathbb{P}\left[\hat{x}_u^{(r)} - x_u \geq \frac{2N}{B}\right]^R = 2^{-R}$$

So, union bounding over all the indices, we get the claim. □

In particular, if we take $R = O(\log |U|)$, we can get the result with high probability.

	Count-Min	Frequent Elements
Space	$K \log \lvert U \rvert$	$O(K)$
Estimate	Overestimate	Underestimate
Update time	$O(\log \lvert U \rvert)$ time	$O(1)$ average, $O(K)$ worst case
Recovery time	$O(\lvert U \rvert)$ (iterate over everything)	Simply output dict
Misc	Supports deletions	Is deterministic