Special Topics in Theoretical Computer Science January 28, 2019

Lecture 6: The Heavy Hitters Problem

Huy L. Nguyen *Scribe: Akshar Varma*

Overview

In the last lecture we finished looking at ℓ_p norm estimation by analyzing the case of $p > 2$. We now move on to a different problem, that of estimating frequently occuring elements or the heavy hitters. After defining the problem formally, we look at the MG sketch that solves the problem when there are only positive updates. Then we look at the Count-Min Sketch which can also handle negative updates.

1 Heavy Hitters Problem

We want to find frequently occuring elements in a data stream. For example, if we have a network and get a stream of packets from various machines, we want to find the machines that generate a lot of traffic, or perhaps the packets that are most frequently requested.

The simplest formulation of the problem is to ask for the most frequent item. However one can information theoretically get a lower bound which says that we would need linear memory to find the most frequent item. Intuitively this can be understood by considering the case where all machines can send at most 1 packet, except for a special machine which can send 2 packets. Without storing all previously seen packets/machines, it is impossible to distinguish the special machine from the other machines.

1.1 Formal Problem Definition

Definition 1 (Heavy Hitters) *Given a stream of length m and some parameter k, find all elements that occur $\geq \frac{m}{k}$ times (possibly with a few false positives).*

Definition 2 (Frequency Vector Definition) *For a frequency vector x, find some $\hat{x}$ such that $\forall\, i$, $|x_i - \hat{x_i}| \leq \frac{m}{k} \iff \|x - \hat{x}\|_\infty \leq \frac{\|x\|_1}{k}$.*

Note: The $\|x - \hat{x}\|_\infty \leq \frac{\|x\|_1}{k}$ condition in the frequency vector based definition is a common form (with various other norms being considered) in the compressed sensing domain which we will be looking at later on in the course.

2 Solution for $k = 2$

Consider that the parameter value is $k = 2$, we want to find any element that occurs more than half of the time. This can be done using a simplified version of the Misra-Gries Sketch which is also known as the Frequent Sketch. Algorithm 1 provides pseudocode level details of how the simplified version works.

Intuitively, it can be thought of as keeping track of a frequent index. Anytime we see an element that is distinct from what we are keeping track of, we pair the indices up and remove them so that it was as if we never saw either in the first place. Whatever remains at the end must have been a strict majority or else it would not have survived.

Algorithm 1 "Simple" MG Sketch

```
index ← 0                                          ▷ Initialize values
counter ← 0

On update (i, +1)
if counter == 0 then          ▷ First time update, or if previous values discarded
    index ← i
    counter ← 1
else if index == i then
    counter ← counter + 1                          ▷ Repeated element, increment
else
    counter ← counter − 1          ▷ Throw away the distinct pair of elements
```

3 MG sketch or Frequent Sketch [Misra-Gries '82]

We can generalize the intuition of the "simple" MG sketch to get it to work for arbitrary parameter values k. While earlier we were pairing up distinct indices and discarding such pairs, we now "pair up" k-tuples of distinct indices and discard all the indices in such a tuple. This way, the indices that survive (have positive counts) must all necessarily have occured in the stream more than $\frac{m}{k}$ times.[1]

Algorithm 2 Misra-Gries Sketch

```
(index_j, counter_j) ← (0, 0)      ▷ Initialize k − 1 pairs of (index, counter) values.

On update (i, +1)
if ∃j s.t. count_j == 0 then                   ▷ If there is a free spot, insert there
    index_j ← i
    counter_j ← 1
else if ∃j s.t. i == index_j then                  ▷ Repeated element, increment
    counter_j ← counter_j + 1
else
    for all j from 1 to k − 1 do     ▷ Throw away the distinct k-tuple of elements
        counter_j ← counter_j − 1
```

During the for loop in the last else clause, for any element e, the frequency of e might go down by 1. This allows us to make the following claim for the MG sketch which is enough to show that it solves the heavy hitters problem.

Claim 3 $|x_e - \hat{x_e}| \leq \frac{\|x\|_1 - \|\hat{x}\|_1}{k} \leq \frac{m}{k}$

[1] Note that the MG sketch only retains elements that are strictly more than $\frac{m}{k}$. Which means that, for example, to find elements that may be present *exactly* $m/2$ times we would need to set $k = 3$.

3.1 Merging multiple MG sketches

In general, we may need to merge multiple MG sketches into a single sketch. Particularly, in the network monitoring scenario, we may be monitoring multiple routers and might be interested in a consolidated set of heavy hitters. We now look at how to merge two MG sketches, the method trivially generalizes to more sketches.

Suppose the index-counter tuples from the first sketch are $(i_1, c_1), \ldots, (i_{k-1}, c_{k-1})$ and the second sketch's tuples are $(i_k, c_k), \ldots, (i_{2k-2}, c_{2k-2})$. We first merge the common elements to get $(i_1, c_1), \ldots, (i_l, c_l)$, where $l \leq 2k-2$. Let us also assume that these have been sorted so that $c_1 \geq c_2 \geq \cdots \geq c_l$. We can now imagine performing the MG sketch update on this "stream" by subtracting the smallest counter value c_l from every index-counter pair. And then subtracting the next smallest counter and so on until only $k-1$ indices remain. The net effect of such subtractions is to simply subtract c_k from all counters and remove all $c_i \leq 0$.

Thus, the MG sketch can be merged quite easily. However, it is easy to see that the MG sketch will not work if our stream also has negative updates. Our algorithm cannot handle an update of the form $(i, -1)$. We will now look at a solution to the heavy hitters problem that works even in the presence of negative updates.

4 Count-Min Sketch [Cormode-Muthukrishnan '05]

The Count-Min Sketch keeps d separate estimates for $\hat{x_i}$ and finally returns the median of all estimates. The original sketch of Cormode and Muthukrishnan returned the minimium of all estimates (hence the name Count-Min) but that only worked for positive updates. By using the median, we can extend the sketch to work for negative updates.

For each $j \in [d]$ an estimate is maintained using a hash function $h_j : [n] \to [w]$, and for each bucket $b \in [w]$ we store the sum of all elements hashed into that bucket $c_{j,b} = \sum_{h_j(i)=b} x_i$. Estimate j of $\hat{x_i}$ is then simply the value stored in the bucket that x_i was hashed to: $c_{j,h_j(i)}$.

For one hash function we have that the expected estimate $\hat{x_i}$ deviates from the correct value by the sum of all other elements that get hashed into the same bucket:

$$\mathbb{E}\left[c_{1,h_1(i)}\right] = x_i + \frac{1}{w}\sum_{s \neq i} x_s \in [x_i - \frac{\|x\|_1}{w}, x_i + \frac{\|x\|_1}{w}]$$

We will only analyze the case with positive updates to keep calculations simple.[2] Since we only look at positive updates, we know that $c_{1,h_1(i)} - x_i \geq 0$, and we also know that the expectation is bounded from above. Using Markov's inequality gives us that the probability of failing on one estimate is bounded by a constant.

$$\mathsf{Pr}\left[c_{1,h_1(i)} - x_i \geq \frac{3\|x\|_1}{w}\right] \leq \frac{1}{3}$$

[2]To get the same guarantees with negative updates, we will additionally need to calculate the variance of our estimate which is much more cumbersome.

Since we return the median of all d estimates, the sketch when more than $d/2$ of the estimates fail. This failure probability can be upper bounded using the Chernoff bound to be $\exp(-\Omega(d))$. By setting the number of estimates $d = \Theta(\log n)$, we get the probability of failure on one index of x to be less than $1/n^2$. Using the union bound over all indices we can show that the sketch succeeds with very high probability.

4.1 Improving to an ℓ_2 norm error [proof/fix from here]

For the heavy hitters problem, we are often interested in the case where the values in x are almost all the same, except for a few outliers. In such a scenario, the ℓ_2 norm error will be better at picking outliers compared to the ℓ_1 norm. For example, in the load balanced case, the ℓ_1 norm will be n while the ℓ_2 norm will be $\sqrt{n}$. Thus, if our error was bounded in terms of the ℓ_2 norm (as in the following definition) we would have a much tighter bound.

Definition 4 (Heavy Hitters with ℓ_2 norm error) *For a frequency vector x, find some $\hat{x}$ such that $\|x - \hat{x}\|_\infty \leq \frac{\|x\|_2}{k}$.*

We can solve the Heavy Hitters problems with an ℓ_2 norm error bound by using the Count Sketch which we will now state and analyze in the next lecture.

4.2 Count Sketch [Charikar–Chen–Farach-Colton '04]

The only change from the Count-Min Sketch is that in the Count Sketch before adding up the values hashed to a bucket, we multiply them by a random sign.

As before, we have d estimates and for each $j \in [d]$, we have a hash function $h_j : [n] \to [w]$ and now a sign function $s_j : [n] \to \{\pm 1\}$ and for each bucket $b \in [w]$ we store $c_{j,b} = \sum_{h_j(i)=b} x_i \cdot s_j(i)$. Since we are multiplying by a random sign, our estimates will need to correct by multiplying by the same sign. This means our estimate $\hat{x_i}$ is given by the expression $c_{j,h_j(i)} \cdot s_j(i)$.

Multiplying by the sign means we would probably get some values to cancel each other. That's a very high level, hand-wavy intuition for why Count Sketch gives an error bounded by the ℓ_2 norm while the Count-Min Sketch is only able to give an error bounded by the ℓ_1 norm. However the Count-Min sketch only needs $O(\frac{1}{\epsilon} \log n)$ space while the Count sketch needs $O(\frac{1}{\epsilon^2} \log n)$ space. There is a tradeoff between space and error guarantee among the two sketching schemes and the choice between them depends on the regime one is in.

References

[1] Misra, J., & Gries, D. (1982). Finding repeated elements. Cornell University.

[2] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1), 58-75.

[3] Charikar, M., Chen, K., & Farach-Colton, M. (2004). Finding frequent items in data streams. Theoretical Computer Science, 312(1), 3-15.