CS 15-859: Algorithms for Big Data	Fall 2019
Lecture 7— October 25, 2019	
Prof. David Woodruff	*Scribe: Irene Li*

1 Find the Top k Heavy Hitters Quickly

Goal: quickly find all heavy hitters for $\phi = k$, $\epsilon = \frac{1}{2}k$.

A straightforward way is to search among the estimates of x_i's. However, there are n entries and searching among them is too costly. We want time complexity linear to k instead of n. The general idea is to use a full binary tree where at level i, there are 2^i groups, each containing continuous $\frac{n}{2^i}$ entries from x. Starting from the level with 4k groups, estimate the norm of each group (treat each group as one entry by summing up all values in the group) and pick the top $2k$ groups to continue to the next level and repeat this process till the leaves.

Why does this approach work?

1.1 Restricted Case

Restriction: $\forall i$, $x_i \geq 0$; want l_1-heavy hitter.

The goal is to output a set T which contains all items j for which $x_j \geq \frac{1}{k}|x|_1$, and no items j' for which $s_{j'} < \frac{1}{2k}|x|_1$.

1.1.1 Algorithm and Correctness

Consider on level i where there are 2^i groups each of size $\frac{n}{2^i}$. Let s_j represent the sum of values in group j. If group j contains a heavy hitter $x_k \geq \frac{1}{k}|x|_1$. Then $s_j \geq x_k \geq \frac{1}{k}|x|_1$. Thus, s_j is a $\frac{1}{k}$-heavy hitter among the numbers s_1, ..., s_{2^i}.

How do we estimate s_j? We hash all the 2^i items into $10k$ buckets and repeat this procedure $\mathcal{O}\log n$ times independently. Now, we estimate s_j by looking at the bucket it lands in in each of the $\mathcal{O}\log n$ independent repetitions and taking the median. Note that in each repetition, the bucket it lands in has sum at least s_j, which is at least $\frac{1}{k}|x|_1$ if there is a heavy hitter x_k in the j-th group, since then $s_j \geq x_i \geq \frac{1}{k}|x|_1$. Thus, the median will also have this property. On the other hand, suppose $s_{j'} \leq \frac{1}{2k}|x|_1$. Then since we hash into 10k buckets, the expected count in the bucket containing $s_{j'}$ in a given repetition is at most $s_{j'} + \frac{|x|_1}{10k}$. By a Markov bound, with probability at least $\frac{2}{3}$, it is at most $s_{j'} + \frac{3|x|_1}{10k} \leq \frac{4|x|_1}{5k}$. Thus, if we repeat the procedure $\mathcal{O}(\log n)$ times independently and take the median, with probability $1 - 1/poly(n)$, our estimate of $s_{j'}$ will be less than $\frac{|x|_1}{k}$, and so $s_{j'}$ will not be returned as a heavy hitter.

Thus, the only items returned by the heavy hitters algorithm are those items $s_{j'}$ for which $s_{j'} \geq \frac{|x|_1}{2k}$ and every item with $s_j \geq \frac{|x|_1}{k}$ will be returned. Thus, any groups with heavy hitters will be returned.

By pigeon hole principle, we also know that at most $2k$ items are returned.

1.1.2 Space Complexity

For each value of i such that $(4k \le 2^i \le n)$, we had $\mathcal{O}(\log n)$ repetitions and $10k$ buckets in each repetition. Each bucket maintained an $\mathcal{O}(\log n)$ bit counter. Since there are $\mathcal{O}(\log n)$ different values of i, this gives $\mathcal{O}(k \log^3 n)$ bits of memory. (this bound can be further optimized)

1.1.3 Time Complexity

To find the l_1-heavy hitters, namely, those $x_i \ge \frac{|x|_1}{k}$, we do the following. Start at the level containing $4k$ groups (it's important that we start from a level with $\mathcal{O}(k)$ groups for the sake of efficiency). Run the above heavy hitters procedure, and it is guaranteed that it will return at most 2k items. All heavy hitters are guaranteed to be within those 2k groups. Then on the next level, we only need to look at the children of these 2k groups. Those 2k groups have 4k children in the next level. Therefore, we still only need to estimate 4k items. We repeat this process until reaching the bottom where we find the actuall heavy hitters. As we step down on the binary tree, it is guaranteed that we never left out a heavy hitter and it's also guaranteed that the heavy hitter procedure only needs to deal with at most 4k items at each level.

In each level of the binary tree, we only have to spend $\mathcal{O}(k \log n)$ time to compute $4k$ medians, much better than $\mathcal{O}(2^i \log n)$ time (if we consider all groups on that level). There are $\mathcal{O}(\log n)$ levels, so we spend $\mathcal{O}(k \log^2 n)$ time in total. Note that our algorithm can be much smaller than $\mathcal{O}(n \log n)$ time of computing the median estimate of all n items directly.

1.2 More General cases

l_2**-heavy hitter**:

Replace s_j with $\hat{s_j}$, a sketch of the 2-norm for the group. Then we have:

$$\hat{s_j} \ge (1-\epsilon_0)|x|_2^2 \ge (1-\epsilon_0)x_k^2 \ge \frac{1-\epsilon_0}{k}|x|_2^2$$

if x_k is a l_2-heavy hitter. Also,

$$\sum_j \hat{s_j} \le (1+\epsilon_0)|x|_2^2$$

.

Using the above two facts, for sufficiently small constant ϵ_0, the above analysis can be generalized to show that at each level of the tree only $2k$ groups can be returned.

l_1**-heavy hitter with negative entries in x**: Replace s_j with $\hat{s_j}$, a sketch of the 1-norm. Now we have,

$$\hat{s_j} \ge (1-\epsilon_0)|x|_1 \ge (1-\epsilon_0)|x_i| \ge \frac{1-\epsilon_0}{k}|x|_1$$

if x_i is an l_1-heavy hitter. Also,

$$\sum_j \hat{s_j} \leq (1+\epsilon_0)|x|_1$$

Again, we can still make the same argument with sufficiently small constant ϵ_0.

2 Estimating Number of Non-Zero Entries (l_0)

Definition: $|x|_0 = |\{\text{i such that } x_i \neq 0\}|$

Goal: Output a number Z with $(1-\epsilon)Z \leq |x|_0 \leq (1+\epsilon)Z$ with probability 9/10 and use $O((\log n)/\epsilon^2)$ bits of space.

2.1 Algorithm for the Sparse Case

Suppose $|x|_0 = O(\frac{1}{\epsilon^2})$. We can compute it exactly in two ways:

- Use k-sparse recovery algorithm from last lecture.
- Use CountSketch (with tail guarantee).

2.2 Algorithm for the General Case ($|x|_0 \gg \frac{1}{\epsilon^2}$)

2.2.1 Reduce Error Given a 2-approximate estimate

Suppose we already have an estimate Z with $Z \leq |x|_0 \leq 2Z$. We can obtain a better estimate up to a multiplicative factor of ϵ. The algorithm proceeds as follows:

1. Independently sample each coordinate i with probability $p = \frac{100}{Z\epsilon^2}$.
2. Let Y_i be an indicator random variable if coordinate i is sampled. Let y be the vector restricted to coordinates i for which $Y_i = 1$. In other words, $y_i = x_i$ if coordinate i is sampled and is 0 otherwise (note that $|y|_0 \leq |x|_0$ because y contains a subset of the non-zero entries in x).
3. Use sparse recovery or CountSketch to compute $|y|_0$ exactly.
4. Output $\frac{|y|_0}{p}$ to be the estimate of $|x|_0$.

Claim: $(1-\epsilon)|x|_0 \leq \frac{|y|_0}{p} \leq (1+\epsilon)|x|_0$ with high probability.

Proof:

$$\mathbb{E}[|y|_0] = \sum_{i\ .\ \text{s.t.}\ x_i \neq 0} \mathbb{E}[Y_i] = p|x|_0 = \frac{100}{Z\epsilon^2}|x|_0 \in \left[\frac{100}{\epsilon^2}, \frac{200}{\epsilon^2}\right]$$

$$\mathrm{Var}[|y|_0] = \sum_{i\ .\ \mathrm{s.t.}\ x_i \neq 0} \mathrm{Var}[Y_i] = \sum_{i\ .\ \mathrm{s.t.}\ x_i \neq 0} p(1-p) \leq p|x|_0 \leq \frac{200}{\epsilon^2}$$

Apply Chebyshev inequality:

$$\Pr\left[||y|_0 - \mathbb{E}[|y|_0]| > \frac{100}{\epsilon}\right] \leq \frac{\mathrm{Var}[|y|_0]\epsilon^2}{100^2} \leq \frac{1}{50}$$

Thus, with probability $\frac{49}{50}$, we have $(1-\epsilon)\,\mathbb{E}[|y_0|] < |y|_0 < (1+\epsilon)\,\mathbb{E}[|y_0|]$ (note that $\frac{100}{\epsilon} \leq \epsilon\,\mathbb{E}[|y|_0]$). Therefore, $\frac{|y|_0}{p}$ approximates $|x|_0$ with a relative error of ϵ.

2.2.2 Find a 2-approximate estimate

We don't actually have Z such that $Z \leq |x|_0 \leq 2Z$, but we can find Z by guessing it in powers of 2. Since $0 \leq |x|_0 \leq n$, there are $O(\log n)$ guesses. The algorithm looks as follows:

1. The i-th guess $Z = 2^i$ corresponds to sampling each coordinate with probability $p = \min(1, \frac{100}{2^i\epsilon^2})$
2. Sample the coordinates as nested subsets $[n] = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots \supseteq S_{\log n}$
3. Run the previous algorithm for each guess $Z = 2^i$

One of our guesses Z satisfies $Z \leq |x|_0 \leq 2Z$ and we should use that guess.

Claim: The largest guess $Z = 2^{i^*}$ for which $\frac{400}{\epsilon^2} \leq |y|_0 \leq \frac{3200}{\epsilon^2}$ is a good guess.

Proof: Use y^i to denote the y vector for the i-th guess. For the i-th guess, $\mathbb{E}[|y^i|_0] = p_i|x|_0 = \frac{100}{2^i\epsilon^2}|x|_0$. Thus, $\mathbb{E}[|y^i|_0]$ decreases monotonically with i. Note that $|y|_0$ also decreases monotonically with i because the sampling is nested.

For $Z = 2^i$ such that $kZ \leq |x|_0 \leq 2kZ$. Then $\mathbb{E}[|y|_0] = \frac{100}{Z\epsilon^2}|x|_0 \in [\frac{100}{\epsilon^2}k, \frac{200}{\epsilon^2}k]$.

Consider $Z = 2^i$ such that $k = 8$. By the previous statement, we know:

$$\frac{800}{\epsilon^2} \leq \mathbb{E}[|y^i]_0 \leq \frac{1600}{\epsilon^2}$$

Then by Chebyshev's inequality,

$$\frac{400}{\epsilon^2} \leq |y^i|_0 \leq \frac{3200}{\epsilon^2}$$

with probability at least 49/50. Similarly, for $i' = i+3$, i.e., $Z' = 2^{i'} = 8Z$:

$$\frac{100}{\epsilon^2} \leq \mathbb{E}[|y^{i'}]_0 \leq \frac{200}{\epsilon^2}$$

Again by Chevyshev's inequality,

$$|y^{i'}|_0 \leq \frac{400}{\epsilon^2}$$

with probability at least 49/50.

By union bound, with probability 48/50, $\frac{400}{\epsilon^2} \le |y^i| \le \frac{3200}{\epsilon^2}$ and $|y^{i+3}| \le \frac{400}{\epsilon^2}$. If these two conditions are true, then i^* satisfies $\frac{200}{\epsilon^2} \le \mathbb{E}[|y^{i^*}|_0] \le \frac{1600}{\epsilon^2}$. Since i satisfies $\frac{400}{\epsilon^2} \le |y^i| \le \frac{3200}{\epsilon^2}$ and i' is the largest guess for which the inequality is true, we know $i^* \ge i$. Moreover, $i^* < i+3$ because $|y^{i+3}|_0 < \frac{400}{\epsilon^2}$ and y with larger guess would have even smaller l_0 due to nested sampling. Overall we have $i \le i^* < i+3$. Thus, there are only 3 possible values for i'. By union bound, the probability of all of them satisfying $|y|_0 = (1 \pm \epsilon)\mathbb{E}[|y|_0]$ is 1-3/50.

Overall, the success probability is $1 - \frac{2}{50} - \frac{3}{50} = \frac{9}{10}$.

2.3 Overall Space Complexity

k-sparse recovery: if we use k-sparse recovery algorithm for $k = \mathcal{O}(\frac{1}{\epsilon^2})$, then it takes $\mathcal{O}(\frac{\log n}{\epsilon^2})$ bits of space in each of the $\log n$ levels, so we need $\mathcal{O}(\frac{\log^2 n}{\epsilon^2})$ bits of space for k-sparse recovery.

Space for randomness: We can implement nested sampling by choosing a hash function $h : [n] \to [n]$. On level i, choose coordinate j to be in S_i if $h(j) < \frac{100}{2^i \epsilon^2} n$. This is equivalent to sample each coordinate with probability $\frac{100}{2^i \epsilon^2}$ and the resulting subsets are guaranteed to be nested because the threshold $\frac{100}{2^i \epsilon^2} n$ decreases with i. h only needs to be pairwise independent because this is enough for Chebyshev's inequality. We can store a pairwise independent hash function with $\mathcal{O}(\log n)$ bits. Thus, we need $\mathcal{O}(\log n)$ space for randomness.

Overall, the space complexity is $\mathcal{O}(\frac{\log^2 n}{\epsilon^2})$.

Improve space complexity to $\mathcal{O}\left(\frac{\log n(\log(\frac{1}{\epsilon}+\log\log n))}{\epsilon^2}\right)$: The improvement comes from shrinking space complexity of our k-sparse recovery algorithm. Note that for each entry, we don't care about its value. We only care if it's non-zero. Therefore, the main idea is to decrease the size of each counter by finding a "good" prime number and then operating in the modulo space. What is a "good" prime number p? One that does not divide any counter so that modulo p doesn't change the non-zeroness of each counter.

In sampling levels we care about (3 levels, see the end of Section 2.2.2), we have $\mathcal{O}(\frac{1}{\epsilon^2})$ counters, each of $\mathcal{O}(\log n)$ bits. Therefore, at most $n = \mathcal{O}(\frac{\log n}{\epsilon^2})$ prime numbers can divide any of these counters. If we randomly pick a prime p from the first $\mathcal{O}(n)$ primes numbers, then with high probability, p doesn't divide any counters in the important 3 levels (obtain high probability by picking a reasonable constant in $\mathcal{O}(n)$). Since an upper bound on the n-th prime number is $\mathcal{O}(n \log n)$, p is upper bounded by $\mathcal{O}(\log n \log\log n/\epsilon^2)$. Now the space needed for each counter is upper bounded by $\log p = (\log\log n + \log\frac{1}{\epsilon})$. Therefore, the space complexity is improved to $\mathcal{O}\left(\frac{\log n(\log(\frac{1}{\epsilon}+\log\log n))}{\epsilon^2}\right)$.

Open problem: is it possible to remove the term $log(\frac{1}{\epsilon})$?