CS 15-851: Algorithms for Big Data Spring 2025

Lecture 9 — 3/20

Prof. David Woodruff *Scribe: Ramsey Boyce*

0.1 Completing heavy hitters

Remark 1. Finding the top k heavy hitters quickly

Say we are trying to find the ℓ_1 heavy hitters of a vector x with all coordinates $x_i \geq 0$. Suppose at most k heavy hitters. Find all i with $x_i \geq \epsilon \|x\|_1$, here $k = 1/\epsilon$.

Our previous algorithms take $O(n)$ time to find heavy hitters. How can we decrease the time complexity?

Construct a new vector y^1 by grouping x into $2k$ groups of size $n/2k$. Run ℓ_1 heavy hitter algorithm on y^1, the top k heavy hitters in x are also heavy hitters in y^1.

Construct a new vector y^2 by grouping x into $4k$ groups of size $n/4k$. Run ℓ_1 heavy hitter algorithm on y^2 in parallel. In y^2, we only need to examine the groups that were heavy hitters for y^1!

Repeat this process for $y^3, y^4, \ldots, y^{\log n/2k}$. This creates a binary tree structure of height $O(\log n)$ where we only consider $2k$ nodes at each level.

This results in $O(k \log n)$ total time to find the heavy hitters, from $O(n)$. The space complexity increases by a $\log n$ factor. The assumptions made can also be removed, but it complicates the algorithm.

1 Estimating the number of non-zero entries

Suppose we would like to estimate the number of non-zero entries, where our input data comes from a stream. We create some notation to refer to this quantity below.

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

Our goal is to output a number Z such that $(1-\epsilon)Z \leq |x|_0 \leq (1+\epsilon)Z$ with probability 9/10. Additionally, our algorithm should have low space complexity.

1.1 Sparse x

Suppose $|x|_0 = O(1/\epsilon^2)$. In this case we can run a k-sparse recovery algorithm, setting $k = O(1/\epsilon^2)$. Alternatively, we could use CountSketch.

If $|x|_0 >> 1/\epsilon^2$, our general idea will be to sample x to get a sufficiently sparse vector, and then use sparse recovery. However, finding the correct sampling rate is the challenge.

1.2 General x

Suppose we somehow had an estimate Z such that $Z \le |x|_0 \le 2Z$. We would like to improve this estimate to a $(1 \pm \epsilon)$ bound.

Since we have a rough estimate, we can determine the right sampling rate. Independently sample each coordinate i with probability $p = 100/(Z\epsilon^2)$.

Let Y_i be an indicator random variable if coordinate i is sampled. Let y be the vector restricted to coordinates i for which $Y_i = 1$.

Analyze $|y|_0$ by calculating its mean and variance.

$$\mathbb{E}[|y|_0] = \sum_{i:\, x_i \neq 0} \mathbb{E}[Y_i] = \sum_{i:\, x_i \neq 0} p = p|x|_0 = \frac{100}{Z\epsilon^2}|x|_0 \ge \frac{100}{\epsilon^2}.$$

$$\mathrm{Var}[|y|_0] = \sum_{i:\, x_i \neq 0} \mathrm{Var}[Y_i] = \sum_{i:\, x_i \neq 0} p(1-p) \le p|x|_0 = \frac{100}{Z\epsilon^2}|x|_0 \le \frac{200}{\epsilon^2}.$$

Note that we used that the Y_i are independent for the variance calculation. By Chebyshev's inequality,

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

Then we can use sparse recovery or CountSketch to compute $|y|_0$ exactly, and output $|y|_0/p$. However, we don't know what value to use for Z!

To determine Z, we will guess it in powers of 2! Since $0 \le |x|_0 \le n$, there are $O(\log n)$ guesses.

Guess i will sample at the rate $p_i = \min(1, 100/(2^i\epsilon^2))$. Also, we sample the coordinates as nested subsets $[n] = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots \supseteq S_{\log n}$. This means that to get to the next subset from the previous one, we just have to sample at a $1/2$ rate.

For each guess of Z, we will run our previous algorithm. One of our guesses Z satisfies $Z \le |x|_0 \le 2Z$, and we should use that guess. But how do we know which one?

Use the largest guess $Z = 2^i$ for which $400/\epsilon^2 \le |y|_0 \le 3200/\epsilon^2$.

p	Notes
1	$\lvert y\rvert_0$ large
$1/2$	$\lvert y\rvert_0$ is half as large as before
$1/2^i$	$400/\epsilon^2 \le \lvert y\rvert_0 \le 3200/\epsilon^2$
$1/2^{i-1}$	$\lvert y\rvert_0 < 400/\epsilon^2$
$\vdots$	$\lvert y\rvert_0 = 0$

As you go to smaller and smaller subsets, $|y|_0$ decreases. If $800/\epsilon^2 \le \mathbb{E}[|y|_0] \le 1600/\epsilon^2$,

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

with probability at least $49/50$ by a Chebyshev bound. If $100/\epsilon^2 \le \mathbb{E}[|y|_0] \le 200/\epsilon^2$,

$$|y|_0 < \frac{400}{\epsilon^2}$$

with probability at least 49/50 by Chebyshev again. So combining these, with probability 98/100, we choose an i for which

$$\frac{200}{\epsilon^2} \leq \mathbb{E}[|y|_0] \leq \frac{1600}{\epsilon^2}.$$

Note that using nested subsets lets us avoid a union bound over all levels.

There are only 4 such indices i (as $Z = 2^i$), so by a union bound all 4 satisfy $|y|_0 = (1 \pm \epsilon)\,\mathbb{E}[|y|_0]$ simultaneously with probability $1 - 4/50$. So it doesn't matter which of them we choose. Overall, our success probability is $1 - 2/50 - 4/50 > 4/5$.

1.3 Space complexity

If we use our k-sparse recovery algorithm with $k = O(1/\epsilon^2)$, it takes $O(\log n/\epsilon^2)$ space for $\log n$ levels, making the total complexity $O(\log^2 n/\epsilon^2)$ excluding random bits.

For the random bits (sampling), we only need pairwise independence for Chebyshev's inequality. The nested sampling can be implemented with a hash function $h\colon [n] \to [n]$. Our scheme will be: if $h(i) \leq n/2$, sample i, then for the next subset, if $h(i) \leq n/4$, sample i, ... This gives us nested subsets and pairwise independence. In general, we check if the first i bits of $h(j)$ are 0. This hash function only requires $O(\log n)$ bits.

The space complexity can also be reduced to

$$O\left(\frac{\log n(\log(1/\epsilon) + \log\log n)}{\epsilon^2}\right).$$

This is because sparse recovery gives us the values of the non-zero entries. However, we don't actually care what the values are, just that they're non-zero. So we'll reduce the space needed for the counters in sparse recovery using primes.

The number of primes in $\{1, \ldots, x\}$ is $\Theta(x/\log x)$. There are $O(1/\epsilon^2)$ counters, each in the range $\{\text{poly}(n), \ldots, \text{poly}(n)\}$. Then at most $O(\log n/\epsilon^2)$ primes divide at least one of these counters. A random prime

$$q = O\left(\frac{\log n \log\log(n/\epsilon^2)}{\epsilon^2}\right)$$

is unlikely to divide any of our counters. So we can maintain the sparse recovery mod q, requiring space

$$O\left(\frac{\log\log n + \log(1/\epsilon)}{\epsilon^2}\right)$$

for $O(\log n)$ sparse recovery instances.