CS 15-851: Algorithms for Big Data Spring 2024

Lecture 10 — 3/28

Prof. David Woodruff *Scribe: Hoai-An Nguyen*

1 L_0 Estimation in a Stream

Our problem setting is the following. We get a stream of updates to a vector x. We want to estimate the number of non-zero entries in x. In other words, we want to output

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

Note that other notes/people may call this the L_0 norm. However, strictly speaking, this is not a norm because it doesn't scale.

We relax the problem in the following way. We want to output a number Z such that we have

$$(1-\epsilon) \cdot Z \leq |x|_0 \leq (1+\epsilon) \cdot Z$$

with probability 9/10. We want to aim (and will present) for an algorithm that uses $O((\log n)/\epsilon^2)$ bits of space.

1.1 Sparse x

Suppose for a moment that x is sparse, or that $|x|_0 = O(1/\epsilon^2)$. How would we solve the problem in this case? Well, we could either use our algorithm for recovering a k-sparse vector from last time with $k = O(1/\epsilon^2)$. We could also use CountSketch with roughly $O(1/\epsilon^2)$ rows which will recover these k items exactly.

But, we want to be able to handle general x. So, what do we do? As a general idea, we could subsample x such that our subsample is sufficiently sparse, and then run sparse recovery on this subsampled vector. But, we have a problem. Since our sampling rate then depends on $|x|_0$ which we do not know, what sampling rate do we use?

Let's now show a simple example to see if this can give us a better intuition for what sampling rate we should choose later. Suppose that $|x|_0 \leq \sqrt{n}$. In this case, we could just run sparse recovery with $\sqrt{n}$ space. If $|x|_0 > \sqrt{n}$, then we could sample with rate $1/(\epsilon^2\sqrt{n})$ which would leave us with roughly $\sqrt{n}$ entries. Then, we would be in the first case. We note two things. First, we would need to determine which case we are in so that we can determine if we need to subsample. In addition, in general, $\sqrt{n}$ space is too large. However, this is a good starting point to build our intuition. Now, we get into the algorithm.

1.2 The Algorithm

Let us first suppose that we had an estimate Z such that $Z \leq |x|_0 \leq 2Z$. Since we have a rough approximation of $|x|_0$, we can now determine our sampling rate such that we can get an appropriately

sparse subsampled vector. We want to refine this to a $(1 \pm \epsilon)$ approximation.

First, we sample each coordinate i with probability $p = 100/(Z\epsilon^2)$. Note that the constants do not matter too much; they can be adjusted depending on the constant probability of success one wants.

Let Y_i be an indicator random variable if coordinate i is sampled, and let y be the vector restricted to coordinates i for which $Y_i = 1$. Let us prove that with high probability, y is sufficiently sparse or $O(1/\epsilon^2)$.

We know that

$$\mathbb{E}[|y|_0] = \sum_{i \text{ such that } x_i \neq 0} \mathbb{E}[Y_i] = p|x|_0 \geq \frac{100}{\epsilon^2}$$

where the inequality is because we know $|x|_0 \geq Z$.

Now, let us calculate the variance. We have

$$Var[|y|_0] = \sum_{i \text{ such that } x_i \neq 0} Var[Y_i] \leq \frac{200}{\epsilon^2}.$$

In the last inequality, we are using the fact that the $Y_i's$ are independent (note that we only need pairwise independence here which will be important for later) and that $2Z \geq |x|_0$. So, using Chebyshev's, we have

$$\mathbb{P}[||y|_0 = \mathbb{E}[|y|_0]| > \frac{100}{\epsilon}] \leq \frac{Var[|y|_0]\epsilon^2}{100^2} \leq \frac{1}{50}.$$

Now, we can use sparse recovery or CountSketch to compute $|y|_0$ exactly. We output $|y|_0/p$. But, we don't know Z. How do we find a coarse approximation?

We will use a technique that is popular in streaming: we'll simply guess Z in powers of 2. And since $O \leq |x|_0 \leq n$ there will be $O(\log n)$ guesses. The i^{th} guess $Z = 2^i$ will correspond to sampling each coordinate with probability $p = \min(1, \frac{100}{2^i\epsilon^2})$.

We will furthermore sample the coordinates as nested subsets $[n] = S_0 \supseteq S_1 \subseteq S_2 \subseteq ...S_{\log n}$. Here S_0 represents sampling such that $Z = 2^0$, S_1 represents sampling such that $Z = 2^1$, and so on. Note that for the first few sampling rounds, we will sample with probability $p = 1$. Then, sampling the nested subsets means that to get the next subset, we sample the previous subset with $p = 1/2$. Now, for each guess of Z, we can run the algorithm we had above which assumed getting a 2-approximation of $|x|_0$. One of our guesses satisfies $Z \leq |x|_0 \leq 2Z$. But how do we know which one?

We will use the largest guess $Z = 2^i$ for which

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

Let's prove why this will give us the desired approximation with the right success probability. If we know that $\frac{800}{\epsilon^2} \leq \mathbb{E}[|y|_0] \leq \frac{1600}{\epsilon^2}$, then we know that

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

with probability at least 49/50. On the other hand, we know that if $\frac{100}{\epsilon^2} \leq \mathbb{E}[|y|_0] \leq \frac{200}{\epsilon^2}$, then we know

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

with probability at least 49/50.

We get these both from our inequality using Chebyshev's above. So, combining these together, we have that with probability at least 48/50, we choose an i for which

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

Since there are only 4 such possible indices i (since we guess Z in powers of 2), and all 4 of them satisfy $|y|_0 = (1 \pm \epsilon)\mathbb{E}[|y|_0]$ simulataneously with probability at least $1 - 4/50$, it doesn't matter which one we choose.

1.3 Space Complexity

Now, let us verify our space complexity. If we use our k-sparse recovery algorithm for $k = O(1/\epsilon^2)$, then it takes $O(\log n/\epsilon^2)$ bits for each level. This is because we need $O(\log n)$ for the counter, and we are using the k-sparse algorithm for $k = O(1/\epsilon^2)$. Since there are $\log n$ total levels, there are $O(\log^2 n/\epsilon^2)$ total bits of space ignoring random bits.

For the random bits (or the sampling part), we do the following. We only need pairwise independence for our proof using Chebysehv's inequality. We can simply implement the nested sampling by choosing a hash function

$$h : [n] \to [n]$$

and checking if the first i bits of $h(j) = 0$. This gives us our pairwise independence and nested sampling. This is because checking if the first i bits of $h(j)$ are 0 depends on if the first $i - 1$ bits of $h(j)$ are 0. Specifically, it is half of that probability. This hash function only requires $O(\log n)$ bits.

We note that we can actually improve the overall space to

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

Intuitively, using sparse recovery actually gives us the non-zero items. But, in this problem, we only need to know how many there are. In other words, if we have a counter for each subsampled element in a specific level of sampling, we only need to know the number of non-zero counters (not what the counters actually are). So we do the following to reduce the counters in each level from $\log n$ bits to $O(\log(1/\epsilon) + \log\log n)$ bits.

We do this via primes. We note that if we look at the numbers between 1 and $\Theta(m \log m)$, there are $\Theta(m)$ primes in that range. So we do the following. In the sampling levels that we care about, we have $O(1/\epsilon^2)$ counters each of $O(\log n)$ bits. There can be at most $O(\log n/\epsilon^2)$ prime numbers dividing any of these counters. We will choose a random prime

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

It is unlikely that q divides any of our counters. So, we can just instead maintain our sparse recovery structure mod q. Therefore, we only need

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

bits per each of the $O(\log n)$ sparse recovery instances.