CS 15-859: Algorithms for Big Data Fall 2017

Lecture 7-1 — October 19, 2017

Prof. David Woodruff *Scribe: Manuel Fernandez*

1 Recap of l_1-regression, and how to speed it up

Recall that the goal for l_1 regression is to find x minimizing

$$\|Ax - b\|_1$$

for a given A and b. Before, we took a $d \log d \times n$ sketch matrix R and tried to solve

$$\min_x \|RAx - Rb\|_1$$

Then we found a special basis U with the property that

$$\frac{\|x\|_1}{\text{poly}(d)} \leq \|Ux\|_1 \leq \text{poly}(d) \|x\|_1$$

and subsequently sampled rows from the special basis dependent on it' l_1 norm, and then solved exactly the resulting linear regression problem on the sample matrix.

The problem with this approach is that we have to compute $R \cdot A$, which is expensive. Instead we claim we can solve this problem by choosing $R = S \cdot C$, where S is a CountSketch matrix and C is a diagonal matrix of i.i.d. Cauchy random variables. In particular we have that

$$\frac{\|Ax\|_1}{d^2 \log^2(d)} \leq \|RAx\|_1 \leq O(d \log d) \|Ax\|_1 ,$$

which speeds up the problem from $\text{nnz}(A) \cdot d \log d + \text{poly}(d/\varepsilon)$ to $\text{nnz}(A) + \text{poly}(d/\varepsilon)$.

1.1 Interesting Cauchy facts

We note that for n i.i.d. copies $R_1, \ldots, R_n$ of a standard Cauchy random variable, the distribution of

$$\frac{\sum_{i=1}^n R_i}{n}$$

is still a Cauchy random variable.

2 Introduction to the Streaming Model

2.1 Turnstile Streaming Model

In the Turnstile Streaming Model, we start with a vector $x_0 = 0 \in \mathbb{R}^n$. Then we are give a long stream of updates (i.e. $x_{i+1} := x_i + \Delta_i$) where $\Delta_i \in \{-M, -M+1, \ldots, M-1, M\}$, where we assume that $M \leq \operatorname{poly}(n)$. Furthermore, we may assume that $x_i \in \{-M, \cdots, M\}$. We then wish to to output a good approximation of $f(x)$ with high probability at the end for some function f. Note that in this model, out x is massive and the amount of space we wish to use to compute $f(x)$ is far smaller than x itself. We study this model with the intent to study problems such as stock transactions, weather data, and genomes.

2.2 Testing if $x = 0^n$

One question we could ask is if we can determine if $x = 0^n$ after the stream runs. More specifically, we wish to investigate whether we can do so efficiently (with only $O(\log n)$ bits of space). One way to see how to do this is to note that a CountSketch matrix S with $O(\frac{1}{\varepsilon^2})$ is a $(1 \pm \varepsilon)$subspace embedding and so we may pick $\varepsilon = O(1)$ so that Sx has only $O(1)$ entries and we can store each entry with $O(\log(n))$ bits and therefore the entire vector with $O(\log(n))$ bits. Using hash functions, we can store a representation of S with $6\log(n)$ bits so the total amount of storage needed is $O(\log(n))$.

Interestingly, the problem cannot be solved deterministically, as there is in fact an $\Omega(n\log(n))$ lower bound on the problem. To see this consider the case where on the first 1/2 of a stream the vector a is in $\{0, 1, \operatorname{poly}(n)\}^n$ and in the second 1/2 is in $\{-\operatorname{poly}(n), \cdots, -1, 0\}$ using $o(n \log n)$ bits. Then, this means that there are vectors a, a' which collide. We may then apply a standard pigeonhole principle argument on the state of the algorithm to conclude that the algorithm must output the same result for the two different inputs.

2.3 Recovery of k-sparse vectors

What if we are asked to recover x given that only k entries are non-zero? It turns out you can do this with $k \operatorname{poly}(\log n)$ bits of space, and in fact we can do so deterministically.

To begin our investigation of this problem, first consider the case where x only has 1 non-zero entry. Then if we maintain $(1, \cdots, 1)x = x_i$ and $(1, 2, \cdots, n)x = ix_i$, we can recover the value of x_i and its index i. Now we attempt to generalize this idea. Consider the following algorithm:

1. Let A be an $s \times n$ matrix s.t. that any 2k columns are linearly independent
2. Maintain $A \cdot x$

Claim 1. One can recover the subset S of k non-zero entries and their values from $A \cdot x$.

Proof. This is a simple argument based on linear independence of columns. ■

An example of a matrix A that works is the Vandermonde matrix with $s = 2k$ rows and n columns and $A_{i,j} = j^{j-1}$. One can show that any $2k$ columns of A are linearly independent. Unfortunately, the entries of A are increase exponentially in n. To work around this, we store $A \cdot x \mod p$ for prime p of size $\operatorname{poly}(n)$. Note that this works since taking the entries modulo a prime p maintains the independence of the columns (consider the determinant for example) and $x \mod p = x$ since have chosen $x \leq \operatorname{poly}(n) < p$.

Remark 1. We claimed that we could make this algorithm deterministic, but the standard approach for finding primes is randomized. If you really want to make this truly deterministic, you can use the fact that there is always a prime in $[n, 2n]$ and check all values in this finite range.

2.4 Estimation of norms

Another streaming problem is the approximation of norms, i.e. $f = \|\cdot\|$. For instance, we'd like to approximate the 2-norm, i.e. compute Z for which

$$(1-\varepsilon)\|x\|_2^2 \leq Z \leq (1+\varepsilon)\|x\|_2^2$$

We can do this by taking a $\mathsf{CountSketch}$ matrix S with $(1/\varepsilon^2)$ rows and making incremental updates of the product Sx. Note that we have met our space requirements since S can be stored efficiently using limited independence. In addition, our theory of subspace embeddings shows that outputting $\|Sx\|_2$ at the end of the stream yields a good approximation with high probability. Note that our space complexity is $1/\varepsilon^2$ words, each with $\log n$ bits.

Now consider the corresponding problem for the 1-norm. For the problem of finding Z that satisfies

$$(1-\varepsilon)\|x\|_1 \leq Z \leq (1+\varepsilon)\|x\|_1,$$

we might consider sampling a random Cauchy matrix which we can store with $(1/\varepsilon)$ words, update $Sx := Sx + s_i S_{*,i}$ and at the end output $\|Sx\|_1$. Unfortunately, this doesn't work because Cauchy random variables are not well-concentrated. However, we can notice that

$$Sx = \|x\|_1 \sum_{i=1}^{(1/\varepsilon^2)} |C_i|_1 \frac{1}{n}$$

Consider taking the median value of the entries. Now note that the cdf of a Cauchy random variable evaluated at z is

$$F(z) = \int_0^z f(x)\,dx = \frac{2}{\pi}\arctan(z)$$

and $F(1) = 1/2$ exactly when $z = \tan(\pi/4)$, so $\text{median}(|C|) = 1$. Note that if we take $r = \frac{\log(1/\delta)}{\varepsilon^2}$ independent samples $X_1, \cdots, X_r$ from F and set $X = \text{median}_i X_i$, then $F(X)$ lies in $[1/2-\varepsilon, 1/2+\varepsilon]$ w.p. $(1-\delta)$. To see this, for each sample X_i we define A_i, B_i where A_i corresponds to $A_i \geq 1/2 - \varepsilon$ fraction of distribution and $B_i \leq 1/2 + \varepsilon$ and use Chernoff to show concentration around $[1/2-\varepsilon, 1/2+\varepsilon]$. From this we get that $F^{-1}(X) = \tan(\frac{x\pi}{2}) \in [1-4\varepsilon, 1+4\varepsilon]$ and thus we have shown that using the median as our estimator works.