CS 229r: Algorithms for Big Data Fall 2013

Lecture 3 — September 10, 2013

Prof. Jelani Nelson *Scribe: Christopher Musco*

1 Overview

In the previous lecture we discussed algorithms (idealized and practical) for the Distinct Elements problem, or the"F_0 problem". A section was also added to the notes on F_2 estimation for a continually updated vector $\boldsymbol{x} \in \mathbb{R}^n$. Using the AMS sketch algorithm presented in [AlonMS99], we saw that, with just $O(\epsilon^{-2}\log(1/\delta)\log(n))$ bits of space, it is possible to maintain an ϵ-good approximation for $\|\boldsymbol{x}\|_2^2$ with probability $> 1-\delta$.

In this lecture we'll continue looking at sketching. We will:

1. Review the "turnstile stream" problem.
2. Consider linear sketches for F_p estimation when $0 < p < 2$
3. Consider the $p > 2$ case

2 Turnstile Streams

As described in the Lecture 2 notes, the general streaming model we will follow is:

- Begin with an n length vector x initialized to $\mathbf{0}$.
- Stream in a set of updates in the form $\{i, v\}$, where $i \in \{1, \ldots, n\}$ and $v \in \mathbb{R}^n$.
  For each update, set $x_i \leftarrow x_i + v$. In practice, we probably want to be bounding v.
- After streaming, return $f(x)$ for some f. Depending on what f is, we should be able to do this without actually maintain x (which could have many elements) explicitly.

In linear streaming, we'll reduce out space complexity by maintaining an m length vector $y = \Pi\boldsymbol{x}$, where $\Pi \in \mathbb{R}^{m\times n}$. Our space complexity will be $O(m)$ + whatever space is required to store Π, which better be $< O(n)$. Let Π_i be the ith column on Π (an m length vector). Then, in the linear streaming case, our update rule becomes:

$$y \leftarrow y + v\Pi_i \tag{1}$$

Okay, cool. So how did this model fit with the F_2 problem we looked at last time?

2.1 F_2 Problem Review

The function $f(x)$ we wanted to return after streaming was:

$$F_2 \overset{\text{def}}{=} \sum_{i=1}^{n} x_i^2 = \|x\|_2^2 \tag{2}$$

We choose Π to be:

$$\Pi = \frac{1}{\sqrt{m}} \begin{bmatrix} \pm1 & \dots & \pm1 \\ \vdots & \vdots & \vdots \\ \pm1 & \dots & \pm1 \end{bmatrix} \tag{3}$$

Where each element was chosen randomly from $\{-1, 1\}$. Finally, we saw that, instead of storing mn random bits to represent our matrix, it sufficed to store just a single 4-wise independent hash function from $[n] \to [n]$ ($\in \mathcal{H}_{poly-4}$) for each row. That means only $O(\log n)$ bits for each row. To get our approximation error low with high probability, we needed to choose an m dependent on ϵ and we had to run our algorithm multiple times in parallel, with the number of instances dependent on δ. But regardless, this gave us $O(\log n)$ space complexity for storing Π, with constants depending on ϵ, δ, which is what we're looking for.

Okay, so we saw that linear sketching was possible for $F_2 = \|x\|_2^2$ approximation. Additionally, F_0 is actually just the number of distinct elements in a vector, so we saw that was possible as well. Can we generalize and show that a linear sketching algorithm exists for other p-norms? Recall that:

$$\|x\|_p = \left(\sum_{i=1}^{n} \|x_i|^p \right)^{(1/p)} \tag{4}$$

3 The F_p Problem: $0 < p < 2$

In the $0 < p < 2$ case, we'll use **Indyk's Algorithm** [Indyk06].

3.1 Idealized Algorithm

Definition 1. *(p-stable distribution) D_p is a p-stable distribution if $\forall x \in \mathbb{R}^n, \sum_{i=1}^{n} x_i Z_i \sim Z\|x\|_p$ where $Z, Z_1, \dots, Z_n \sim D_p$ and are independent r.v.'s.*

We already know about one p-stable distribution: the Gaussian distribution with mean $\mu = 0$ is 2-stable. The Cauchy distribution is 1-stable.

Theorem 2. *A p-stable distribution exists iff $0 < p \leq 2$*

We'll use the existence of p-stable distributions for the $0 < p < 2$ case. Following the turnstile model above, consider the following algorithm:

- Choose $m = O(\frac{1}{\epsilon^2})$ - we'll see why later
- Define $\Pi \in \mathbb{R}^{m \times n}$ by setting each entry $\Pi_{i,j} = Z_{i,j}$ where each $Z_{i,j} \sim D_p$
- Recall that $y \in \mathbb{R}^m = \Pi x$. Estimate:

$$\|x\|_p \approx \frac{median_{i=1,\ldots,m}|y_i|}{median_{i=1,\ldots,m}D_p} \tag{5}$$

What do we by "$median(D_p)$" - that's a distribution?

Definition 3. *For a probability distribution D with probability density function $D(x)$, $median(D)$ is the value t such that $D(t) - D(-t) = \frac{1}{2}$.*

So, half the density of the distribution falls in $[-t, t]$. For simplicity, when we analyze this algorithm, let's assume $\mathbf{t = 1}$ for D_p, which is the distribution the entries of our Π are drawn from. Under this assumption, our estimate becomes:

$$\|x\|_p \approx median_{i=1,\ldots,m}|y_i| \tag{6}$$

We claim that, if we choose m as described, this estimate is good with high probability.

Claim 4. $\mathbb{P}\left((1-\epsilon)\|x\|_p < median|y| < (1+\epsilon)\|x\|_p\right) \geq 2/3$

Proof. Let Z_i be the ith row of Π

$$y_i = \langle Z_i, x\rangle = \sum_{j=1}^{n} x_j Z_{i,j} \rightarrow y_i = \|x\|_p Z \tag{7}$$

Where $Z \sim D_p$. This follows from Definition 1 since each $Z_{i,j} \sim D_p$. Define $\mathrm{I}_{[a,b]}$ to be the indicator function of the interval $[a, b]$:

$$\mathrm{I}_{[a,b]}(w) = \begin{cases} 1 \text{ if } w \in [a,b] \\ 0 \text{ otherwise} \end{cases} \tag{8}$$

Since we assumed $median(D) = t = 1$, and since $\frac{y_i}{\|x\|_p} = Z$ we see that:

$$\mathbb{E}\, I_{[-1,1]}\left(\frac{y_i}{\|x\|_p}\right) = \frac{1}{2} \tag{9}$$

Since any PDF is bounded by 1, we also know that:

$$\mathbb{E}\, I_{[-1-\epsilon,1+\epsilon]}\left(\frac{y_i}{\|x\|_p}\right) = \frac{1}{2} + \Theta(\epsilon) \tag{10}$$

$$\mathbb{E}\, I_{[-1+\epsilon,1-\epsilon]}\left(\frac{y_i}{\|x\|_p}\right) = \frac{1}{2} - \Theta(\epsilon) \tag{11}$$

To make sure out estimator actually centers around the correct value for $\|x\|_p$, we want to make sure that, in expectation, $median\left(\frac{y_i}{\|x\|_p}\right) \in [1-\epsilon, 1+\epsilon]$. The count of the number of $|y_i|$ in $[0, (1-\epsilon)\|x\|_p]$ is:

$$\sum_{i=1}^{m} I_{[-1+\epsilon,1-\epsilon]}\left(\frac{y_i}{\|x\|_p}\right) \tag{12}$$

The expectation of which is $m\left(\mathbb{E}\, I_{[-1+\epsilon,1-\epsilon]}\left(\frac{y_i}{\|x\|_p}\right)\right) < \frac{1}{2}m$. Similarly, The count of the number of $|y_i|$ in $[0,(1+\epsilon)\|x\|_p]$ is:

$$\sum_{i=1}^{m} I_{[-1-\epsilon,1+\epsilon]}\left(\frac{y_i}{\|x\|_p}\right) \tag{13}$$

The expectation of which is $m\left(\mathbb{E}\, I_{[-1-\epsilon,1+\epsilon]}\left(\frac{y_i}{\|x\|_p}\right)\right) > \frac{1}{2}m$. Now, if less than half of the m elements in $\{1_i,\ldots,i_m\}$ lie below $(1-\epsilon)\|x\|_p$, the median must be above this value. If more than half of the m elements lie above $(1+\epsilon)\|x\|_p$, the median must be below this value. So, we see that, in expectation, $median\left(\frac{y_i}{\|x\|_p}\right) \in [1-\epsilon, 1+\epsilon]$ as desired.

Next we should look at the variance of these count estimates to make sure that the median actually falls in this range with high probability.

$$\operatorname{Var}\left(\sum_{i=1}^{m} I_{[-1+\epsilon,1-\epsilon]}\left(\frac{y_i}{\|x\|_p}\right)\right) = \sum_{i=1}^{m}\operatorname{Var}\left(I_{[-1+\epsilon,1-\epsilon]}\left(\frac{y_i}{\|x\|_p}\right)\right) \leq \sum_{i=1}^{m} 1 \tag{14}$$

Simply by the fact that our indicator function can only take values $0,1$. Similarly,

$$\operatorname{Var}\left(\sum_{i=1}^{m} I_{[-1-\epsilon,1+\epsilon]}\left(\frac{y_i}{\|x\|_p}\right)\right) \leq m \tag{15}$$

Now, we can apply Chebyshev's inequality to each of these estimates for our y_i counts to show that, with appropriately chosen m, with high probability (i.e. $> 2/3$):

$$(1-\epsilon)\|x\|_p < median|y| < (1+\epsilon)\|x\|_p$$

$\square$

So, this algorithm works. However, our Π consisted of mn random numbers drawn from D_p. That requires too much space to store. How do we "derandomize" this algorithm?

3.2 Pseudorandomness

General approach: given a short truly random bit string, generate a longer "pseudorandom" string that looks random to some class of algorithms (the algorithms behavior doesn't change by much).

Can we replace $\{Z_{i,1},\ldots,Z_{i,n}\}$ with a pseudorandom string? We can by using Nisan's Pseudorandom Generator (PRG) for space-bounded algorithms, which is described in [Nisan92]. *Trivia*: Pseudorandom was actually misspelled as "Psuedorandom" in the title of Nisan's original conference paper (corrected for the journal version)...

3.2.1 Nisan's PRG

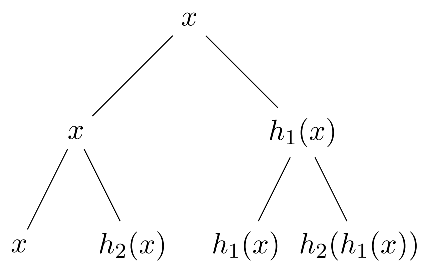
Prof. Nelson had a nice diagram in class that unfortunately my LaTeX skills aren't currently able to reproduce. I'll try to get something together and put up a change soon.

Anyway, imagine we have a program that runs using at most s bits of space and R random bits. Then, the program has at most 2^s possible memory states, $\{S_1, S_2, \ldots, S_{2^s}\}$. Let's call the state of the program just before reading in our first random bit, r_1, "Layer 1" or L_1. $L_1 \in \{S_1, \ldots, S_{2^s}\}$. L_i will be the state of the program just before reading in random bit r_i. If we use at most R random bits, there are R such layers and also a final layer, L_{R+1}, which is the state of the program just after reading in the last random bit. At any layer, L_a, in our execution, if we're at state S_i, based on the next random bit, L_{a+1} will equal 1 of 2 possible other states, S_j, S_k. Note that j and k don't have to be distinct (maybe even after reading a random bit the execution is deterministic) and either can equal i (our memory contents might not change at all). Based on this model, our R random bits will induce a distribution on the final layer, $L_{R+1} \in \{S_1, \ldots, S_{2^s}\}$ of our execution.

Theorem 5. *([Nisan92]) - For any $R \leq 2^{cs}$ for some constant $c > 0$, there is a PRG $h : \{0,1\}^{O(s\log(R/s))} \to \{0,1\}^R$ such that the final distribution on L_{R+1} when using the pseudorandom bits is indistinguishable from the distribution when using R truly random bits with probability more than $1/2^s$. That is to say, for any function $f : \{0,1\}^s \to \{0,1\}$, if x is uniform in $\{0,1\}^R$ and y is uniform in $O(s\log(R/s))$, then*

$$|\mathbb{P}(f(x) = 1) - \mathbb{P}(f(h(y)) = 1)| \leq \frac{1}{2^s}.$$

What does such a generator look like? Say we start with a string $x \in \{0,1\}^s$ of truly random bits. Place x at the root of a tree. In this tree, every left child of a node will simply equal the bit string at that node. To find right children, choose a random $h_i \in \mathcal{H}_{poly-2} : [2^s] \to [2^s]$ for each level of the tree. Hash your bit string with this function and place it at your right child. So, the first few levels of our tree would look like:



At each leaf we have s bits, so to get R pseudorandom random bits, we R/s leaves $\to \log(R/s)$ levels $\to \log(R/s)$ hash functions. Recall from Lecture 2 that each hash function requires s random bits $\to O(s\log(R/s))$ random bits total.

3.2.2 Algorithm to Fool

So how do we know how much pseudorandomness we can use in Indyk's algorithm? If we need R random bits total, how many truly random bits do we need to pass into Nisan's PRG to get an acceptable string of R pseudorandom bits out? Well, lets consider a new algorithm, "ATF", the Algorithm to Fool. This will be an algorithm that checks Indyk's algorithm to determine whether or not is was actually given random bits (as opposed to pseudorandom bits). If we can fool it with a certain level of pseudorandomness, then we're fine to pass our pseudorandom bits into Indyk's algorithm.

Since it's a checker algorithm, we can assume that we already know the true value of $\|x\|_p$.

1. Initialize 3 counters $curr = c_1 = c_2 = 0$
2. Initialize 2 more counters $i \in [m]$ and $j \in [n]$ to 1
3. Maintain $\langle Z_i, x\rangle$ in $curr$. Look at entries in Π row-by-row from left to right.
4. For each i, at the end of row i, if $|curr| \leq (1-\epsilon)\|x\|_p$, $c_1 \leftarrow c_1 + 1$.
   If $|curr| \leq (1+\epsilon)\|x\|_p$, $c_2 \leftarrow c_2 + 1$.

So, after running ATF:

$$c_1 \rightarrow \text{ for how many } i \text{ was } |y_i| \leq (1-\epsilon)\|x\|_p \tag{16}$$
$$c_2 \rightarrow \text{ for how many } i \text{ was } |y_i| \leq (1+\epsilon)\|x\|_p \tag{17}$$

Indyk's algorithm succeeds iff $c_1 < \frac{m}{2}$ and $c_2 < \frac{m}{2}$.

Now, the space complexity of ATF is $O(\log n + \log(\text{precision with which we store curr}))$. Additionally, we can store Π using $O(n)$ truly random bits. Based on Theorem 5, we need $\approx s\log(R/s)$ bits of true randomness to fool ATF $\rightarrow O(\log n \log(n)) = O(\log^2(n))$ random bits total.

4 The F_p Problem: $p > 2$

This is an algorithm from [Andoni12] based on the works [JST11, AKO11]. The first nearly optimal space algorithm for $p > 2$ is due to [IW05].

In this algorithm, we store a sketch $y = PDx$. P is an $m \times n$ matrix. For each column in P, we insert either ± 1 randomly in a random row. So, there will be n non-zero entries, and exactly 1 in each column. Define $n \times n$ matrix D as:

$$D = \begin{bmatrix} 1/u_1^{1/p} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/u_n^{1/p} \end{bmatrix} \tag{18}$$

where each $u_i \sim$ Exp. Recall that, if $X \sim$ Exp:

$$\mathbb{P}(X > t) = \begin{cases} 1 \text{ if } t \leq 0 \\ e^{-t} \text{ if } t > 0 \end{cases} \tag{19}$$

Our estimate of $\|x\|_p$ is:

$$\|y\|_\infty = \max_{i=1,\dots,m} y_i \tag{20}$$

If we set:

$$m = O\left(n^{1-\frac{2}{p}} \log^{O(1)} n\right) \tag{21}$$

Then, it is possible to guarantee that our estimate $\|y\|_\infty$ is in $[\frac{1}{4}\|x\|_p, 4\|x\|_p]$ with probability $\geq 2/3$.

References

[AlonMS99] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[Andoni12] Alexandr Andoni. High frequency moments via max-stability. Manuscript, 2012. `http://web.mit.edu/andoni/www/papers/fkStable.pdf`

[AKO11] Alexandr Andoni, Robert Krauthgamer, Krzysztof Onak. Streaming Algorithms via Precision Sampling. *FOCS*, pgs. 363–372, 2011.

[Indyk06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM* 53(3): 307–323, 2006.

[IW05] Piotr Indyk, David P. Woodruff. Optimal approximations of the frequency moments of data streams. *STOC*, pgs. 202–208, 2005.

[JST11] Hossein Jowhari, Mert Saglam, Gábor Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. *PODS*, pgs. 49–58, 2011.

[Nisan92] Noam Nisan. Pseudorandom Generators for Space-Bounded Computation. *Combinatorica*, 12(4):449-461, 1992.