CS 229r: Algorithms for Big Data Fall 2013

Lecture 2 — Sept. 5, 2013

Prof. Jelani Nelson *Scribe: William Chen and Sebastian Chiu*

1 Overview

In this lecture we will talk about some problems in Streaming, including ...

1. ... the Distinct Elements Problem (aka the F_0 Problem), whereby you have a stream of m integers $i_1, i_2, ..., i_m$, and you want to output the number of distinct elements in the stream. We will have an approximate algorithm that can probabilistically estimate the number of distinct elements. A simple motivating example is identifying the number of distinct IP addresses that you encounter.
2. ... a "real" Distinct Elements Algorithm. In the first problem, we will assume pure randomness. Here, we will show an impossibility result for streaming algorithms. To solve the F_0 Problem, we require both an approximation and randomization.
3. ... Turnstile Streams, F_2-estimation

2 The Distinct Elements Problem

There are a number of simple ways we can count the number of distinct elements in a set. First, we can maintain a bit vector of length n, and if you see the integer n, set that bit to 1. You can also record all integers m that you see. The simple approach takes $\min\{n, O(m \log m)\}$ bits. n refers to the approach using a bit vector, and $O(m \log m)$ refers to the approach whereby we remember the whole stream of integers.

2.1 Idealized Streaming Algorithm (ISA)

1. Pick random hash function $h : [n] \to [0, 1]$
2. Calculate $z = \min_{i \in \text{stream}} h(i)$
3. Output $\frac{1}{z} - 1$

This is idealized because we don't have perfect precision computers, so we can perfectly work with the numbers between 0 and 1. We have to make sure we use the same h. We have to remember the randomness associated with each i. That's already at least n bits, and this does not perform better than our simple approach.

Why does this work? We will show that it is an unbiased estimator and has bounded variance. Then we can just do this multiple times independently, and take the average of the many estimators we do get, and hence reduce the variance of or overall estimate.

Let's say the $\mathcal{S}$ is the set of $i \in$ stream

$$\mathcal{S} = \{j_1, ..., j_t\}$$

Let

1. $h(j_1), ..., h(j_t) = X_1, ..., X_t$ be independent Unif$[0, 1]$
2. $z = \min\{X_i\}_{i=1}^t$

Claim 1. $\mathbb{E}\, Z = \frac{1}{t+1}$

Proof.

$$\begin{aligned}
\mathbb{E}\, Z &= \int_0^\infty \mathbb{P}(Z > \lambda) d\lambda \\
&= \int_0^1 \mathbb{P}(\forall i X_i > \lambda) d\lambda \\
&= \int_0^1 \prod_{i=1}^t \mathbb{P}(X_i > \lambda) d\lambda \\
&= \int_0^1 (1-\lambda)^t d\lambda \\
&= \int_0^1 u^t du \\
&= \left[\frac{u^{t+1}}{t+1}\right]_0^1 = \frac{1}{t+1}
\end{aligned}$$

□

Claim 2. $\mathbb{E}\, Z^2 = \frac{2}{(t+1)(t+2)}$

Proof.

$$\begin{aligned}
\mathbb{E}\, Z^2 &= \int_0^1 \mathbb{P}(Z^2 > \lambda) d\lambda \\
&= \int_0^1 \mathbb{P}(Z > \sqrt{\lambda} d\lambda \\
&= \int_0^1 (1-\sqrt{\lambda})^t d\lambda \\
&= 2\int_1^0 u^t (u-1) du \\
&= 2\int_0^1 u^t (1-u) du
\end{aligned}$$

□

Averaging Algorithm

1. Run $q = O(\frac{1}{\epsilon^2})$ ISA's in parallel
2. $\bar{z} = \frac{1}{q}\sum_{i=1}^{q} z_i$
3. output $\frac{1}{\bar{z}} - 1$

Claim 3.

$$\mathbb{P}\left(|(\frac{1}{z} - 1) - t| > \epsilon t\right) < \frac{1}{3}$$

Proof. Will show the above claim.

$$\mathbb{P}\left(|(\frac{1}{z} - 1) - t| > \frac{\epsilon}{t+1}\right) < Var(\bar{z}) \cdot \frac{(t+1)^2}{\epsilon^2}$$
$$= O\left(\frac{1}{t^2 q}\right) \cdot \frac{(t+1)^2}{\epsilon^2} < \frac{1}{3}$$

□

In practice, to remove ourselves from the idealized case, we will still use hash functions, but we need one that doesn't take n bits or more to represent (ie. doesn't use floats). So we will use k-wise independent hash functions. What are k-wise independent hash functions?

Let's say that $\mathcal{H}$ is a set of functions that map $[a] \to [b]$.

Definition 4. *$\mathcal{H}$ is a k-wise independent hash family if*

$$\forall i_1 \neq i_2 \neq ... \neq i_k \in [a] \text{ and } \forall j_1, ..., j_k \in [b],$$
$$\mathbb{P}_{h \sim \mathcal{H}}(h(i_1) = j_1 \wedge ... \wedge h(i_k) = j_k) = \frac{1}{b^k}$$

An example is as follows. The set of all functions from $[a]$ to $[b]$ is k-wise independent $\forall k$. The size of $\mathcal{H}$ is b^a ($h \leftarrow \mathcal{H}$ can be specified using $\log|\mathcal{H}|$ bits). The goal now is to come up with small $\mathcal{H}$.

Fulfilled goal: $a = b = q =$ prime power. $\mathcal{H}$ will be the set of all degree $k-1$ polynomials in $\mathbb{F}_q[x]$

We will show that this is a k-wise independent family.

Claim 5. *$\mathcal{H}_{poly-k}$ is a k-wise family.*

Proof. Lagrange interpolation. If we know $i_1, ..., i_k$ and $j_1, ..., j_k$ and that no i's repeat, it follows

$$p(x) = \sum_{r=1}^{k} \left(\frac{\prod_{y=1;y\neq r}^{k} x - x_y}{\prod_{y=1;y\neq r}^{k} x_r - x_y}\right) \cdot j_r$$

satisfies

$$\forall r \; p(i_r) = j_r$$

and this polynomial is unique since $\mathbb{F}_q$ is a field. It follows that $|\mathcal{H}_{\text{poly}-k}| = q^k \Rightarrow h \in \mathcal{H}_{\text{poly}-k}$ representable using $k \log q$ bits □

This raises a new algorithm we can consider.

1. Assume we know t (the answer) up to some constant factor c
2. Pick $h : [n] \to [n]$ from a 2-wise family
3. We will imagine there are $\log_2 n$ different streams and we put i in stream lsb$(h(i))$ (where lsb stands for least significant bit)
4. When i comes in stream, write down i in the stream it hashed to (but stop keeping track of a stream if more than, say, $\frac{1000}{\epsilon^2}$ different indices hashed into it.

Output - Look at stream j s.t. $\frac{1}{c} \leq \frac{t}{2^{j+1}} \leq c$ and output its size times 2^{j+1}

2.2 Necessity of approximation and randomization

Claim 6. *Deterministic exact algorithm is impossible using $o(n)$ bits.*

Proof. Suppose that space of our algorithm A is s bits. That implies that there exists an injection $f : \{0,1\} \to \{0,1\}$.

Let's define the injection

$$x \in \{0,1\}^n$$

1. Define a stream containing the i for which $x_i = 1$
2. Run A on this stream
3. $f(x)$ =memory content of A at the end of the stream

We can recover x bit by bit. To recover a certain bit, we can run A, starting with memory contents $f(x)$ and append the result i to the stream. Now, we can check if F_0 increased. Given an n bit stream, the of F_0 will appear in a stream, and the memory contents of that stream is my compression. How do we prove that this is an injection? Jelani claims that just knowing the compression, you know what x was, so it has to be a injection. □

Claim 7. *A deterministic approximation algorithm for F_0 providing a $(1 \pm 1/1000)$-approximation using $o(n)$ bits is impossible.*

Proof. Suppose there exists $N = 2^{cn}$ bitvectors ($c \in (0,1)$) so that for all $i \neq j$, x_i and x_j differ on at least $\frac{n}{3}$ bits. We then define

$$T = \{x_1, ..., x_N\}$$

The compression can then be defined as

$$f : T \to \{0,1\}^S$$

using the same f as above in the case of a deterministic exact algorithm.

To show f is an injection on T, we will show a recovery algorithm that, given $f(x)$ for $x \in T$, can determine x. We want to know which $x \in T$ was compressed. So we try all $y \in T$ and see if it was y, by appending all i for which $y_i = 1$ into the stream then asking for the new estimate of the number of distinct elements. Doing so yields two possibles cases. The first case is that y was the thing that was compressed, and so number of distinct elements does not increase. The second case is that it wasn't the thing that was compressed, which implies that the number of distinct elements increased by at least $n/3$, so a $(1 \pm 1/1000)$-approximation algorithm will notice this gap. Thus f is an injection implying that the space s must satisfy

$$s = \Omega(log|T|) \geq cn$$

□

3 AMS sketch (F_2 estimation)

we will study the AMS sketch for second moment estimation [1]. In the problem of *second moment estimation*, we imagine we have a vector $x \in \mathbb{R}^n$ that starts as the 0 vector. Every update in the stream is of the form "add v to x_i" for some (i, v). This model for updating vectors in a data stream, where arbitrary amounts can be added to arbitrary coordinates of a high-dimensional vector, is known as the *turnstile model* of streaming. At the end of the stream, we would like to output an estimate of

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

In general we have the definition of the pth frequency moment:

$$F_p \stackrel{\text{def}}{=} \sum_{i=1}^{n} |x_i|^p = \|x\|_p^p,$$

and this is why the distinct elements problem is usually referred to as F_0-estimation (treating 0^0 as 0). This is because in the distinct elements problem where the stream indices are in $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$, we treat i being in the stream as an update $(i, 1)$ to an n-dimensional vector x (so that x_i keeps track of how many times i appeared in the stream). Thus F_0 counts the number of non-zero entries in x, i.e. the number of distinct integers in the stream.

To solve F_2 estimation we will use the strategy of designing a *linear sketch*, i.e. a matrix $\Pi \in \mathbb{R}^{t \times n}$. Our streaming algorithm will simply maintain Πx in its memory, and then our output to estimate F_2 will be some function of Πx. Note that Πx can easily be maintained in turnstile streams, since $x \to x + v \cdot e_i$ (where e_i corresponds to the ith standard basis vector) causes Πx to change in the following way: v times the ith column of Π should be added to Πx. Of course Π is a huge matrix, it is $t \times n$, so to obtain a small-space algorithm we should not maintain Π explicitly in memory, but rather have it implicitly defined (e.g. as we shall see in this example, by using k-wise independent hash functions).

3.1 AMS sketch presentation and analysis

Recall the usual approach for designing a streaming algorithm:

1. Find a way to maintain a random variable X consuming little memory which gives an unbiased estimator of the thing you want to compute, and also bound its variance.
2. Keep many independent random variables $X_1, \ldots, X_{q_1}$ in memory, each distributed as X, and compute the average $\hat{X} = (\sum_{i=1}^{q_1} X_i)/q_1$.
3. Keep many independent random variables $\hat{X}_1, \ldots, \hat{X}_{q_2}$ in memory, each distributed as in the last step, and at the end output the median $\overline{X}$ of the $\hat{X}$'s.

In the second step, if the variance is on the order of the square of the expectation, it suffices to set $q_1 = O(1/\varepsilon^2)$ for $\hat{X}$ to be a $(1+\varepsilon)$-approximation with $2/3$ probability. It then suffices to set $q_2 = O(\log(1/\delta))$ for $\overline{X}$ to be a $(1+\varepsilon)$-approximation with $1-\delta$ probability.

So, it all boils down to the first step: designing an unbiased estimator. The AMS sketch of [1] does exactly this for F_2 estimation. Here is the AMS sketch:

1. Pick a random hash function $\sigma : [n] \to \{-1, 1\}$ from a 4-wise independent family. Abusing notation, let σ be the vector with $\sigma_i = \sigma(i)$.
2. Maintain $X = \langle \sigma, x \rangle$ in memory (so when v is added to x_i, add $v \cdot \sigma_i$ to X).
3. **Output:** X^2

It should be noted that in class we talked about how to design a k-wise independent hash family $\mathcal{H}_{poly-k}$ of functions mapping $[n]$ into $[n]$ of size n^k for n a power of 2 (since $\mathbb{F}_n$ is a finite field for n a power of 2). Such families also imply families mapping from $[n]$ into $\{-1, 1\}$. We make a set $\mathcal{H}'_{poly-k}$ of the same size such that for any $h \in \mathcal{H}_{poly-k}$, we define an $h' \in \mathcal{H}'_{poly-k}$ defined as follows: for any i, if $h(i) \in [n]$ ends in a 0 when written in binary, then we set $h'(i) = 1$; else we set $h'(i)$ to -1.

Lemma 8. $\mathbb{E}\, X^2 = F_2 \stackrel{\text{def}}{=} \|x\|_2^2$.

Proof.

$$\begin{aligned}
\mathbb{E}\, X^2 &= \mathbb{E}\, \langle \sigma, x \rangle^2 \\
&= \mathbb{E} \left(\sum_{i=1}^{n} \sigma_i^2 x_i^2 + \sum_{i \neq j} \sigma_i \sigma_j x_i x_j \right) \\
&= \underbrace{\sum_{i=1}^{n} x_i^2}_{F_2} + \sum_{i \neq j} (\mathbb{E}\, \sigma_i \sigma_j) x_i x_j \\
&= F_2 + \sum_{i \neq j} \underbrace{(\mathbb{E}\, \sigma_i)}_{=0} \underbrace{(\mathbb{E}\, \sigma_j)}_{=0} x_i x_j \qquad (1) \\
&= F_2
\end{aligned}$$

where Equation 1 used that σ is 2-wise independent (since 4-wise independence implies 2-wise independence) to give that the expectation of the product is the product of expectations. □

Lemma 9. *For the variance,* $\mathbb{E}(X^2 - \mathbb{E}\,X^2)^2 \le 2F_2^2$.

Proof.

$$\begin{aligned}
\mathbb{E}(X^2 - \mathbb{E}\,X^2)^2 &= \mathbb{E}\left(\sum_{i\neq j}\sigma_i\sigma_j x_i x_j\right)^2 \\
&= \mathbb{E}\left(2\sum_{i\neq j}\sigma_i^2\sigma_j^2 x_i^2 x_j^2 + 4\sum_{i\neq j\neq k}\sigma_i^2\sigma_j\sigma_k x_i^2 x_j x_k + 24\sum_{i<j<k<\ell}\sigma_i\sigma_j\sigma_k\sigma_\ell x_i x_j x_k x_\ell\right) \\
&= 2\underbrace{\sum_{i\neq j} x_i^2 x_j^2}_{\le F_2^2} + 4\sum_{i\neq j\neq k}(\mathbb{E}\,\sigma_j\sigma_k)x_i^2 x_j x_k + 24\sum_{i<j<k<\ell}(\mathbb{E}\,\sigma_i\sigma_j\sigma_k\sigma_\ell)x_i x_j x_k x_\ell \\
&\le 2F_2^2 + 4\sum_{i\neq j\neq k}\underbrace{(\mathbb{E}\,\sigma_j)}_{=0}\underbrace{(\mathbb{E}\,\sigma_k)}_{=0}x_i^2 x_j x_k + 24\sum_{i<j<k<\ell}\underbrace{(\mathbb{E}\,\sigma_i)}_{=0}\underbrace{(\mathbb{E}\,\sigma_j)}_{=0}\underbrace{(\mathbb{E}\,\sigma_k)}_{=0}\underbrace{(\mathbb{E}\,\sigma_\ell)}_{=0}x_i x_j x_k x_\ell \qquad (2) \\
&= 2F_2^2
\end{aligned}$$

where Equation 2 used the 4-wise independence σ to say that the expectation of the product of 4 different σ_i's is the same as the product of their expectations. □

Because X^2 is an unbiased estimator with variance big-Oh of the square of its expectation, Chebyshev's inequality implies we can average $q_1 = O(1/\varepsilon^2)$ independent copies of X^2 to get a $(1+\varepsilon)$ approximation with $2/3$ probability. Note that this is the same as picking q_1 4-wise independent σ vectors, independent of each other, and dividng each vector by $\sqrt{q_1}$ then stacking them as rows of a $q_1 \times n$ matrix Π. Thus what we are actually doing is maintaining Πx in memory, and what our averaging estimator is finally outputting at the end is $\|\Pi x\|_2^2$, i.e. the second moment of Πx. As usual we can do this $q_2 = O(\log(1/\delta))$ times in parallel, using independent randomness, and outputting the median to get a $(1+\varepsilon)$-approximation with probability $1-\delta$. The total space is $q_1 q_2 = O(\varepsilon^{-2}\log(1/\delta))$, and there are also $q_1 q_2 = O(\varepsilon^{-2}\log(1/\delta))$ hash functions each themselves only requiring $O(\log n)$ bits to store (in fact, if you pay close attention, we really only need q_2 hash functions and we can imagine each σ actually maps $[q_1 n]$ into $\{-1, 1\}$).

References

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