# CS 229r Notes

Sam Elder

November 26, 2013

## Tuesday, September 3, 2013

Standard information:

- Instructor: Jelani Nelson.
- Grades (see course website for details) will be based on scribing (10%), psets (40%)[1], and final project (50%).[2]
- Email: cs229r-f13-staff@seas.harvard.edu.

A lot of the course material is new, within the last 10 years, and since the area is so ripe for contributions, the project will be cutting-edge. There are three possible routes: Make a theoretical contribution to some topic in the course; write a survey covering many related papers in an area (one option is to be risky and try to make a contribution and fall back on a survey if you're not successful); do something with real data. The course doesn't have a TF so far, so we might have to grade each other's psets.

This is a theory course, expecting a level of background of an Intro to Algorithms course, along with basic discrete math, probability, and linear algebra.

Topics for the semester:

1. Sketching/streaming. We compress some data to be able to compute functions on it. For instance, if we're a search engine, we don't want to return a bunch of mirrors of the same content, so we want to compare webpages, but we don't want to simply read the entire pages, but compare some compression of both of them. Streaming is where you want to see a sketch of the data, but the data comes at you progressively.
2. Dimensionality reduction. We have high-dimensional data, which could be a big problem with some of our algorithms. We'll look at general techniques for reducing dimension while making our algorithms still work.
3. Numerical linear algebra. As our input, say we have a really big matrix and we want to algorithmically solve some linear algebra problems. A popular one is matrix completion, where you have a service and products that customers rate, and they try to infer which products you'd like. It's a rather sparse matrix and you want to guess what the rest of the matrix is. Usually you make some assumptions about the structure of the matrix and randomness in which entries are filled in.
4. Compressed sensing. Here you imagine that you have some high-dimensional signal with some structure (often sparse or approximately sparse). An image, for instance, might be sparse if you write the difference between each pixel and the average of its neighbors.

[1] All psets must be typed in LaTeX and e-mailed. There will be maximum page requirements on the solutions, designed so that the solutions are fairly short. They will be every week or week and a half. Collaboration policy: You can look things up and work together, but you have to write it up yourself and cite all sources of collaboration.

[2] The final project will be due the last day of reading period, December 11. The last two days of classes will be devoted to presentations.

5. External memory model. Usually when measuring running time, we count the number of "simple steps" in an algorithm. We do that because it fairly accurately predicts performance when you implement it, but it fails in certain circumstances, like if you have more data than memory. Then touching data outside of the memory takes a much bigger cost (e.g. six orders of magnitude). The better abstraction is that you have some bounded size memory and a disk of memory which is taken to be infinite. Touching data on disk costs 1, and anything on memory is free. It's the other extreme. Why does the disk take so long? It has to spin around to the location that you want to read. But sometimes you can read several entries successively if they're on the same location on the disk, since it's already spun around to that spot. So we introduce one more parameter $B$ known as the block size. You imagine that memory itself is divided into $M/B$ pages of size $B$, and you need to access the pages from the disk one at a time.

6. Mapreduce/Hadoop. These are systems that have been developed for massively parallel computation on large data sets. There have been recent results trying to model these systems mathematically and then develop efficient algorithms on those models.

## Probability Review

We'll mostly be working with discrete random variables in this class. Let $X, X_1, \ldots, X_n$ be discrete r.v.'s taking values in $S$.

**Definition.** The *expectation* $\mathbb{E}X = \sum_{i \in S} \Pr(X = i) \cdot i$. If $X$ is a positive integer, we have $\mathbb{E}X = \sum_{i=1}^{\infty} \Pr(X \geq i)$. The *variance* is $\mathrm{Var}[X] = \mathbb{E}(X - \mathbb{E}X)^2 = \mathbb{E}X^2 - (\mathbb{E}X)^2$.

Some basic tail bounds we'll use all over the place:

**Lemma** (Markov's inequality)**.** *If $X$ is a nonnegative r.v., then $\Pr(X > \lambda) \leq \frac{\mathbb{E}X}{\lambda}$ for any $\lambda > 0$.*

*Proof.* If not, $\mathbb{E}X$ would be too big. □

We will often be informal with these proofs like that, and trust that you can check.

**Lemma** (Chebyshev's inequality)**.** *For all $\lambda > 0$, $\Pr[|X - \mathbb{E}X| > \lambda] < \frac{\mathrm{Var}[X]}{\lambda^2}$.*

*Proof.* $|X - \mathbb{E}X| > \lambda$ iff $(X - \mathbb{E}X)^2 > \lambda^2$ and apply Markov to this. □

There are many ways of stating the next bound, but we'll pick one that fits our purposes.

**Lemma** (Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent r.v.'s with $X_i \in \{0, 1\}$ and $\Pr(X_i = 1) = p_i$, and $X = \sum X_i$ with $\mathbb{E}X = \mu$. Then*

$$\Pr(|X - \mathbb{E}X| < \lambda\mu) \lesssim e^{-c\lambda^2\mu},$$

*for some constant $c$.*

Here $\lesssim$ means that there is a constant that makes the inequality true. In this case, it's a 2, but we don't want to have to worry about it.

*Proof.* We have $\Pr(|X - \mathbb{E}X| > \lambda) = \Pr(X > (1+\lambda)\mu) + \Pr(X < (1-\lambda)\mu)$. We'll bound the first one. Then $X > (1+\lambda)\mu$ iff $e^{tX} > e^{t(1+\lambda)\mu}$ where $t > 0$ is a parameter of our choice. By Markov on $e^{tX}$,

$$\Pr(X > (1+\lambda)\mu) < \frac{\mathbb{E}e^{tX}}{e^{t(1+\lambda)\mu}}.$$

To bound the numerator,

$$\mathbb{E}e^{tX} = \mathbb{E}e^{t\sum_i X_i} = \mathbb{E}\prod_i e^{tX_i} = \prod_i \mathbb{E}e^{tX_i} = \prod_i (p_i e^t + (1-p_i)) = \prod_i [p_i(e^t - 1) + 1] \leq \prod_i e^{p_i(e^t-1)} = e^{\mu(e^t-1)}.$$

We can choose $T = \ln(1+\lambda)$, so we conclude that

$$\Pr(X > (1+\lambda)\mu) \leq \frac{e^{\lambda\mu}}{e^{(1+\lambda)\mu\ln(1+\lambda)}} = \left[\frac{e^{\lambda}}{e^{(1+\lambda)\mu\ln(1+\lambda)}}\right]^{\mu}.$$

So we just need to show that this expression is less than 1. Now $\ln(1+\lambda) = \lambda - \frac{\lambda^2}{2} + O(\lambda^3)$, so the expression is $\exp(\lambda - (1+\lambda)(\lambda - \lambda^2/2 + O(\lambda^3)))$. The $\lambda$ cancels, leaving a $c\lambda^2$ term, as desired. □

Now that we're done with the review, let's talk about big data, so we can see some things that might seem impossible.

## Morris Algorithm

You're a counter, and after $n$ increments, I ask you what you read, and you say "$n$." That takes $\log_2 n$ bits. Instead, if I allow you to be within a factor of 2 of the value with some high probability, how many bits do you need? Well, you need at least $\log\log n$ bits, since you need to remember the answer is between (roughly) $2^i$ and $2^{i+1}$, and you need $\log i$ bits to remember that number to say. Now we'll see an algorithm which matches that. It was the first streaming algorithm, by Morris in the 1970s.

1. Initialize $X$ to 0.
2. If asked to increment $X$, do so with probability $1/2^X$, else do nothing.
3. Output $2^X - 1$.

We'll do a typical expectation-concentration result on this. First:

*Claim.* $\mathbb{E}(2^X) = n + 1$.

*Proof.* Let the counter's state after $n$ increments be $X_n$. We condition over $X_{n-1}$:

$$\mathbb{E}2^{X_n} = \sum_{j=0}^{\infty} \Pr(X_{n-1} = j)\mathbb{E}(2^{X_n} | X_{n-1} = j).$$

The proof is by induction on $n$. We increment with probability $1/2^j$, so this conditional expectation gives

$$\mathbb{E}2^{X_n} = \sum_{j=0}^{\infty} \Pr(X_{n-1} = j)\left(\frac{1}{2^j}2^{j+1} + \left(1 - \frac{1}{2^j}\right)2^j\right) = \sum_{j=0}^{\infty} \Pr(X_{n-1} = j)(2^j + 1) = 1 + \mathbb{E}(2^{X_{n-1}}),$$

which implies the claim by induction. □

Now we do concentration. We'll use Chebyshev this time, so we need to calculate the expectation of the square.

*Claim.* $\mathbb{E}2^{2X_n} = \frac{3}{2}n^2 + \frac{3}{2}n + 1$.

Let's take a step back. Suppose we wanted to be within a factor of $1+\epsilon$ rather than 2. What could we do? Answer: We could run multiple counters in parallel and average them at the end.

If we have $t$ counters $X^{(1)}, \ldots, X^{(t)}$, and we will output $\hat{n} = \frac{1}{t}\sum_{j=1}^{t}(2^{X^{(j)}} - 1)$, then the variance is at most $O(n^2/t)$.

*Claim.* If $t \geq \frac{c}{\epsilon^2}$, then $\Pr(|\hat{n} - n| > \epsilon n) < \frac{1}{3}$.

*Proof.* Chebyshev. This probability is less than $O\left(\frac{n^2}{t}\right)\frac{1}{\epsilon^2 n^2} < \frac{1}{3}$ if we set $t = \frac{c}{\epsilon^2}$. □

Notice though that if we want a probability of failure of $\delta$, we need to multiply the number of counters by $1/\delta$. But Chernoff can get us something stronger. Here's a modified algorithm:

1. Initialize $X^{(1)}, \ldots, X^{(t)}$, with $t = O(1/\epsilon^2)$ each to zero.
2. Upon increment, run the basic incremental step on each $X^{(j)}$ independently.
3. Output $\frac{1}{t}\sum_j (2^{X^{(j)}} - 1)$.
4. Do this $m = O(\log \frac{1}{\delta})$ times independently in parallel and output the median result $\tilde{n}$.

This gives two levels of running things in parallel, and the pset problem 1 will analyze it. Sometimes the median of the averages is the best way to do this sort of thing, but this isn't a case.

*Claim.* We have $\Pr(|\tilde{n} - n| > \epsilon n) < \delta$.

*Proof.* Let $Y_i$ be an indicator for $|\hat{n}_i - n| \leq \epsilon n$, where $i = 1, \ldots, m$ and $n_i$ is the output (average) of the $i$th trial. Then let $Y = \sum Y_i$. This fits exactly the Chernoff setup with probability at least $2/3$ as we calculated before. So the expectation is at least $2m/3$, and as long as at least half of them work, the median will work, too. Since this is an error of $m/6$, which is at most $1/4$ of the expectation, the Chernoff bound applies: The probability of deviation is $\lesssim e^{-c' \log(1/\delta)} < \delta$. □

# Thursday, September 5, 2013

Let's start with an outline for today. Today we'll start by talking about the *distinct elements problem* in streaming. You have a stream of integers $i_1, \ldots, i_m \in [n] := \{1, \ldots, n\}$. You need to output the size of the set of distinct elements.

You might encounter this in network traffic monitoring (number of distinct IP addresses), so this has been used for monitoring the spread of worms. But it's pretty simple and could show up a lot of places.

Then we'll talk about a variant looking at a "real" distinct elements algorithm with limited computational resources. The original problem is called the $F_0$ problem, and to solve it, you need both approximation and randomization. So you can't give an approximate answer deterministically, and you can't probabilistically get an exact answer (say with $2/3$ probability).

Then we'll switch to turnstile streams and $F_2$-estimation, which we'll define when we get there.

## Distinct elements

First, what's the simplest naive algorithm you might use? Well you could just record which elements you've seen so far, using $n$ bits. You could alternatively remember the whole stream, which takes $m \log n$ bits. So you could just take whichever's smaller of these. It isn't entirely clear that you could do better.

Here's an idealized version of a streaming algorithm, which we'll abbreviate ISA. Pick a random function $h : [n] \to [0, 1]$. Maintain in memory the smallest $z = \min h(i)$ that we've seen. When someone asked, we'll output $1/z - 1$. This is idealized for a couple reasons. First, we can't really get infinite precision. The other problem is that we can't remember this whole random function $h$, and that's more than $n$ bits.

Before fixing those things, let's see why this works. We'll take the same approach as last time to show it's unbiased and has low variance. Once you have that, you could duplicate it and average, and so on. Let's say that $S$ is the size of the set of $i$ in the stream, and the true answer is $t$: $S = \{j_1, \ldots, j_t\}$. Then $h(j_1), \ldots, h(j_t)$ are iid uniform in $[0, 1]$, so call them $X_1, \ldots, X_t$, and $Z = \min X_i$.

*Claim.* The expectation of $Z$ is $1/(t + 1)$.

*Proof.* $\mathbb{E}Z = \int_0^\infty \Pr(Z > \lambda) d\lambda$ (true for any nonnegative random variable). $Z > \lambda$ iff $X_i > \lambda$ for all $i$, so this is

$$\mathbb{E}Z = \int_0^1 \prod_{i=1}^{t} \Pr(X_i > \lambda)\, d\lambda = \int_0^1 (1 - \lambda)^t\, d\lambda = \int_0^1 u^t\, du = \frac{u^{t+1}}{t+1}\Big|_0^1 = \frac{1}{t+1}.$$ □

Now as we did on Tuesday, we bound the variance. We need to do a computation here:

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

*Proof.* We use the same steps. We have

$$\mathbb{E}Z^2 = \int_0^1 \Pr(Z^2 > \lambda)\, d\lambda = \int_0^1 \Pr(Z > \sqrt{\lambda})\, d\lambda = \int_0^1 (1-\sqrt{\lambda})^t d\lambda.$$

Now we'll substitute $u = 1 - \sqrt{\lambda}$, so $\lambda = (1-u)^2$ and $d\lambda = 2(u-1)du$, so we get

$$\mathbb{E}Z^2 = 2\int_0^1 u^t(1-u)\, du = \frac{2}{t+1} - \frac{2}{t+2} = \frac{2}{(t+1)(t+2)},$$

as desired. □

Now we do the usual trick where we run $q = O(1/\epsilon^2)$ ISAs in parallel to get $\{Z_i\}_{i=1}^q$. We average them to get $\bar{Z} = \frac{1}{q}\sum Z_i$ and output $\frac{1}{\bar{Z}} - 1$.

*Claim.* $\Pr\left(\left|\left(\frac{1}{\bar{Z}} - 1\right) - t\right| > \epsilon t\right) < \frac{1}{3}$.

We won't always write down this analysis formally, but since this is only the second time, we'll do it this time.

*Proof.* We'll show that $\Pr\left(\left|\bar{Z} - \frac{1}{t+1}\right| > \frac{\epsilon}{t+1}\right) < \frac{1}{3}$, from which the result will follow. By Chebyshev, we have

$$\Pr\left(\left|\bar{Z} - \frac{1}{t+1}\right| > \frac{\epsilon}{t+1}\right) < \mathrm{Var}(\bar{Z})\frac{(t+1)^2}{\epsilon^2} = O\left(\frac{1}{t^2 q}\right)\frac{(t+1)^2}{\epsilon^2} < \frac{1}{3}.$$

As usual, we can throw the constant factors into the value of $q$. □

Now we need to fix up the idealizations. We'll use a tool known as a $k$-wise independent hash function. First let's see a definition.

**Definition.** Let $\mathcal{H}$ be a set of functions mapping $[a] \to [b]$. Then $\mathcal{H}$ is $k$-wise independent if for all $j_1, \ldots, j_k \in [b]$ and distinct $i_1, \ldots, i_k \in [a]$, then

$$\Pr_{h\in\mathcal{H}}(h(i_1) = j_1 \wedge \cdots \wedge h(i_k) = j_k) = \frac{1}{b^k}.$$

If you were choosing a completely random function, this would be true for all $k$. So this is a weakening of this assumption.

So now we have to fix those problems. The continuous function isn't a big deal because we could discretize and show it still works. The bigger issue is that we assumed it was totally random, which was necessary when turning the statement about the minimum into a product. We really did use full independence there.

Specifying a random function from a family $\mathcal{H}$ takes $\log|\mathcal{H}|$ bits, so we want to make $\mathcal{H}$ small. We could take all $b^a$ possible functions, but we need to do better.

Here's what we'll do: Set $a = b = q > k$ a prime power, and define $\mathcal{H}_{\mathrm{poly},k}$ to be the set of all degree $\leq (k-1)$ polynomials in $\mathbb{F}_q[x]$. We claim that $\mathcal{H}_{\mathrm{poly},k}$ is a $k$-wise independent family.

*Proof.* Lagrange interpolation. We just write

$$P(x) = \sum_{r=1}^{k}\left(\frac{\prod_{\substack{y=1\\y\neq r}}^{k} x - i_y}{\prod_{\substack{y=1\\y\neq r}}^{k} i_r - i_y}\right)\cdot j_r.$$

You can easily check that this satisfies $P(i_r) = j_r$ for all $r = 1, \ldots, k$. Since we're in a finite field, this is unique. □

Alright, what is the size of $\mathcal{H}_{\text{poly},k}$? $q^k$. So to specify a random hash function we need $k \log q$ bits.

For this problem, we'll set $k = 2$, but we need to show that 2-independence is enough. Let's specify this new algorithm, which we'll call .

1. Assume we know $t$ (the answer) up to a constant factor $c$.
2. Pick $h : [n] \to [n]$ from a 2-wise independent family. If you need to, round $n$ up to the nearest power of 2 so it's a prime power.
3. We'll imagine that there are $\log_2 n$ different streams, and we put $i$ in the stream of the least significant bit of $h(i)$.
4. When $i$ comes in the stream, write down $i$ in the stream it hashed to, but stop keeping track of a stream if there are $1000/\epsilon^2$ variables.
5. Then output the stream $j$ such that $\frac{1}{\epsilon^2 c} \leq \frac{t}{2^{j+1}} \leq \frac{c}{\epsilon^2}$ and output its size times $2^{j+1}$.

The big idea is that what we're doing in stream $j$ is only going to work if the answer is close to $2^j$.

Here's an example: If $h(7) = 0110011000$ comes in the stream, we write 7 in stream 3, since the 1 in the $2^3$ bit is least significant. So we expect $t/2$ elements to land in the 0th stream, $t/2$ in the 1st, etc. But we'll start ignoring streams that get too big, i.e. more than $1000/\epsilon^2$. There will be a sweet spot where they won't overflow, so we can tell how many are actually there.

How much space does this take? Each number we store is $\log n$, there are $\log n$ streams that don't get bigger than $1000/\epsilon^2$.

Why does it work here that we only have a pairwise independent hash function? Analyzing it only involves an expectation calculation and a variance calculation, and variance only depends on pairwise independence. There's also the space to store $h$, which takes $k \log q = 2 \log n$ bits.

This is mostly an instructive example for how you can get away with an unidealized algorithm. It's not the best algorithm for distinct elements, which does it with $\Theta(\frac{1}{\epsilon^2} + \log n)$ bits.

## Necessity of approximation and randomization

Now we'll see why both of these weakenings are necessary. First let's see why a determinitic exact algorithm is impossible using $o(n)$ bits. This is easy to see. Suppose the space of our algorithm $A$ uses $s$ bits of space. Then there's an injection $f : \{0,1\}^n \to \{0,1\}^s$.

To define $f$, let $x \in \{0,1\}^n$ and send over the stream all of the bit locations that are 1. Run the algorithm $A$ on this stream and output its memory. Then for the recovery algorithm, throw each possible element into the end of the stream and if the size doesn't change, there must be a 1 in that place.

Now we'll also claim that a deterministic approximate algorithm using $o(n)$ bits is also impossible. Formally, we have to show that any approximation factor is impossible, but just to be concrete, we'll show that a $(1 \pm .001)$-approximation algorithm is impossible.

*Proof.* We claim that there exist $N = 2^{cn}$ bit vectors $x_1, \ldots, x_N \in \{0,1\}^n$ such that any two distinct ones differ on at least $n/3$ bits, where $0 < c < 1$ is a constant. We won't prove this, but the strategy is to choose them completely randomly and use Chernoff.

Now we claim that there is a compression $f : T \to \{0,1\}^s$ which is an injection, which will be a contradiction. What is the recovery algorithm? We want to know which $x \in T$ was compressed. Well, you can try all $y \in T$ and see if it was $y$. We append all of the bits where $y_i = 1$ to the stream, and ask what the new number of distinct elements is. If $y$ was the thing that was compressed, this won't change the answer at all. Otherwise, $y$ differs on at least $n/3$ bits, so the number of distinct elements will increase by $n/3$, and our approximation algorithm will pick that up. That implies that the space is at least $\Omega(\log |T|) \geq cn$. □

For the second problem on the pset, you should use this claim as-is. For the third problem, there's a reference to something we didn't get to in class, and while it's not necessary, there will be something on the website about it. Office hours will be on Monday and e-mailed out. Talk to Jelani if you didn't show up on Tuesday.

# Tuesday, September 10, 2013

Today: Linear sketches. Consider turnstile streams: $x \in \mathbb{R}^n$ represents our current knowledge, and the update $(i, v)$ with $x_i \leftarrow x_i + v$ for $v \in \mathbb{R}$ and $i \in [n]$. At the end of the stream, we want to know some function $f(x)$. For example, on the homework, we wanted to know $f(x) = \|x\|_2$.

In *linear sketches*, we maintain some $y = \Pi x$ for $\Pi \in \mathbb{R}^{m \times n}$ where $m \ll n$. The space needed is $m$, along with the space to represent $\Pi$. Then update $(i, v)$ is executed by adding $v$ times the $i$th column of $\Pi$ to $y$. We can imagine $\Pi$ as $n$ column vectors $\Pi_i \in \mathbb{R}^m$, and $y \leftarrow y + v \cdot \Pi_i$.

The AMS sketch we had in the notes for $f(x) = \|x\|_2$ took $\Pi$ to have elements $\pm\frac{1}{\sqrt{m}}$ chosen independently (or 4-independently). The estimate of $\|x\|_2$ was $\|y\|_2$, and $\Pr(\|y\|_2 \in (1-\epsilon, 1+\epsilon)\|x\|_2) \geq \frac{2}{3}$.

What about other norms, $\|x\|_p$ for $p \neq 2$? Recall that $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$. We'll define another linear sketch, but first we need a generalization of the Gaussian.

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

So you're taking the dot product of a joint distribution with some fixed value. If $p = 2$, Gaussians satisfy this, since the sum of Gaussians is a Gaussian and the variances add.

**Theorem.** *p-stable distributions exist iff* $0 < p \leq 2$.

On the second pset, there's an optional exercise to prove this theorem, but it's beyond the scope of the course. They're hard to describe; the simplest is to know that their PDF has a closed-form Fourier distribution. You don't need to understand this, but this property implies that the Fourier transform $\hat{\varphi}_p(t) = e^{-|t|^p}$. Well, $p = 1$ has a closed form, known as the Cauchy distribution. For the rest of this class, we'll just use them as a black box.

## Indyk's Algorithm

We need to specify $\Pi$ and the estimator. The $(i, j)$th entry of $\Pi$ is $Z_{ij} \sim D_p$. We estimate $\|x\|_p$ as the median of $|y_i|$ for $1 \leq i \leq m$, divided by the median of $|D_p|$ where $m = O(1/\epsilon^2)$. The median of $|D_p|$ is the $t$ such that with probability $1/2$, $-t \leq D_p \leq t$. For convenience, we'll let this equal 1.

*Claim.* The median of $|y_i|$ is between $(1 \pm \epsilon)\|x\|_p$ wp at least $2/3$.

*Proof.* We have $y_i = \langle Z_i, x \rangle$, where $Z_i$ is the $i$th row of $\Pi$. Therefore, $y_i \sim \|x\|_p \cdot Z$, where $Z \sim D_p$. If we define the indicator function $I_{[a,b]}(x) = 1$ iff $a \leq x \leq b$, then by the definition of the median, $\mathbb{E} I_{[-1,1]}(y_i / \|x\|_p) = 1/2$.

Now let's see what happens when we move the boundaries of our interval by $\epsilon$. We can see that it should change by about $2\epsilon$ times the pdf of $Z_i$ at $\pm 1$, which is a constant. So we have $\mathbb{E} I_{[-1\mp\epsilon, 1\pm\epsilon]}(y_i / \|x\|_p) = \frac{1}{2} \pm \Theta(\epsilon)$.

Now we want to argue that the total number of $|y_i| \in [0, 1-\epsilon]\|x\|_p$ is less than $m/2$, and the total number of $|y_i| \in [0, 1+\epsilon]\|x\|_p$ is greater than $m/2$, since these will imply that the median $|y_i|$ is in $(1 \pm \epsilon)\|x\|_p$. These counts are just sums of independent indicator variables:

$$\sum_{i=1}^{m} I_{[-1\pm\epsilon, 1\mp\epsilon]}(y / \|x\|_p).$$

The expectation is then $\frac{1}{2} \mp \Theta(\epsilon)$ as we said, and we need to bound the variance. Since they're independent indicator variables, the variance is at most $m$, and if you plug it into Chebyshev, with really good probability, the statements are true. (You can use the union bound at the end.) □

What about this looks suspicious? Well, first, how do you generate samples from a $p$-stable distribution? There are algorithms that will generate those from uniform random variables. It's infinite precision, but as usual, we don't really need to worry about that. What kind of independence did we need? For Chebyshev's variance calculation, you just need pairwise independence of the rows. But we did use full independence of the entries in the same row when we applied the $p$-stability condition. There are $n$ of them, and that's kind of a problem.

The solution comes from Pseudorandomness. The goal is that given a short truly random string, generate a long "pseudorandom" string that "looks random" to some class of algorithm.

What class will we be looking at? We'll consider "small space algorithms" like what we're trying to do here. We want to say that a pseudorandom string will behave approximately the same as a random string, in that it'll give us the same values.

This is called *Nisan's PseudoRandom Generator* (PRG) against space-bounded computation. Formally, we model a space $S$ algorithm that utilizes $R$ random bits with a layered graph with nodes in a $2^S \times (R+1)$ array. From each vertex in the $r$th column, $1 \leq r \leq R$, there are two edges going out labeled 0 and 1 corresponding to what you do when you get a 0 or 1 in the $r$th bit. In this way, a distribution over the possible initial states gives you a distribution over final states. What we want to say is that with our PRG generating $R$, every induced distribution is approximately the same as if the $R$ bits were completely random. The main work is:

**Theorem** (Nisan)**.** *There is a PRG* $h : \{0,1\}^{O(S \log(R/S)+S^2)} \to \{0,1\}^R$ *such that an induced distribution on the last layer in both cases cannot be distinguished with probability* $1/2^S$.

What's the generator? We consider a perfect binary tree, and put a string $x \in \{0,1\}^S$ uniformly random at the root. From that one, every time we go down to a left descendant, we copy the string, and every time we go down to a right descendant at level $i$, we apply a hash function $h_i : [2^S] \to [2^S]$, chosen from a 2-wise independent family. (Note that we use the same hash function at each level.) We do this with depth $\log(R/S)$, so we need $S \log(R/S)$ space to write down our hash functions. We're not sure where the $S^2$ might come from, but it won't matter for our purposes.

Notice that the grid graph we've been considering can depend on the input $x$ exactly, and we'll examine how it works for a particular small-space algorithm which it suffices to check. Here's the algorithm we need to fool:

1. Initialize counters $cur = C_1 = C_2 = 0$ and $i \in [m], j \in [n]$ start at 1.
2. Maintain $\langle Z_i, X \rangle$, the dot product with the current row, in $cur$, scaning entries in $\Pi$ row by row from left to right (scan these with $i, j$).
3. For each $i$ at the end of row $i$, if $|cur| \leq (1-\epsilon) \|x\|_p$, then increment $c_1$, and if $|cur| \leq (1+\epsilon) \|x\|_p$ then increment $c_2$.

So this algorithm at the end stores how many of the dot products $\langle Z_i, X \rangle$ are too small and how many are too big. So Indyk's algorithm succeeds iff $c_1 \leq m/2$ and $c_2 \geq m/2$.

How much space does this take? We need $2 \log m$ bits for $c_1, c_2$, $\log m$ bits with $i$ and $\log n$ bits with $j$. For $cur$, that depends on the precision that we're going for, which will be like $\log(qn/\epsilon)$ bits. So the size of the seed we need will be about $\log^2(qn/\epsilon)$. Then this checker algorithm will succeed whenever Indyk's algorithm does, up to a probability $1/2^S$ error, which is neglibable compared to 2/3. So indeed, Indyk's algorithm is possible with few random bits.

### $\|x\|_p, p > 2$ estimation

We don't have $p$-stable random variables anymore, but we're still going to use a linear sketch, starting with the same matrix from the Homework Problem 3. We have a random hash function $h : [n] \to [m]$ and sign $\sigma \in \{-1, 1\}^n$ with $P_{h(i),i} = \sigma_i$ and all other entries 0. We'll multiply this by a diagonal matrix $D$ with entries $u_i^{-1/p}$ where $u_i \sim \text{Exp}(1)$. Recall that this means $\Pr(u_i > t) = e^{-t}$. We then compute $y = PDx$, and our estimate for $\|x\|_p$ is $\|y\|_\infty = \max_i |y_i|$.

The space we need (apart from the randomness) is $m = O(n^{1-2/p} \log^{O(1)} n)$. Notice that if $p = 2$, the polynomial factor goes away, and as we found, there is indeed an algorithm that works in polylog space. For $p > 2$, though, you can't get rid of that polynomial factor; that's been proved.

*Claim.* $\|y\|_\infty \in [\frac{1}{4} \|x\|_p, 4 \|x\|_p]$ with probability $\geq 2/3$.

The algorithm is due to Andoni in March 2013, based on ideas from [Andoni, Krauthgamer, Onak '10], [Jowhari, Saglam, Tardos '10] and [Indyk, Woodruff '05]. You can improve it to a $(1 \pm \epsilon)$-approximation, but that involves some polynomial factors in $1/\epsilon$ for the space that we don't want to worry about.

## Thursday, September 12, 2013

Today, we'll analyze the algorithm for $F_p$, $p > 2$. Then we'll look at a streaming algorithm for the "heavy hitters" problem, that will arise in Google's algorithm to find trends among search terms. We'll see that version next lecture, and a slightly different one today.

For $p > 2$, the $F_p$ algorithm will produce a vector $y$ which is a linear sketch of the input vector $x$. It'll be obtained by two matrices $P$ and $D$, so $y = PDx$, where $P$ is a $m \times n$ matrix consisting of columns with one random nonzero entry $\sigma_i$ and $D$ is a diagonal $n \times n$ matrix with elements $1/u_i^{1/p}$.

Here $m = O(n^{1-2/p} \log n)$ and $h : [n] \to [m]$, $\sigma[n] \to \{\pm 1\}$ and $u : [n] \to \text{Exp}(1)$ are fully random hash functions, so $h(i)$ and $u(j)$ are independent random variables, for instance. (Recall that the cdf $\Pr(u > \lambda) = e^{-\lambda}$ if $\lambda > 0$.) Then $P_{h(i),i} = \sigma_i$ and the rest of $P$ is zero. Finally, we output $\|y\|_\infty = \max_i |y_i|$.

We already know that $n^{1-2/p}$ bits of space are necessary, so this algorithm is tight up to the $\log n$ factors.

**Theorem** (1)**.** $\Pr(\|y\|_\infty \in [\frac{1}{4} \|x\|_p, 4 \|x\|_p]) > .51$.

Recall that we can boost this probability to $1 - \delta$ by taking the median of $O(\log 1/\delta)$ many trials. To do this, we'll state another version of the Chernoff bound.

**Theorem** (Chernoff v2)**.** *If $X_1, \ldots, X_n$ are independent and $|X_i| \leq K$ for all $i$. If $\mathbb{E} \sum X_i = \mu$ and $\sum_i (\mathbb{E}(X_i - \mathbb{E}X_i))^2 \leq \sigma^2$. Then*

$$\Pr\left(\left|\sum_i X_i - \mu\right| > \lambda\right) \leq \max\left\{e^{-c\lambda^2/\sigma^2}, \left(\frac{\sigma^2}{\lambda K}\right)^{c\lambda/K}\right\}.$$

Morally, this is the same proof as Chernoff's bound. The proof is found by Googling "Terry Tao concentration of measure".

*Proof of Theorem 1.* Let $z = Dx$ and the big idea is that the largest element of $z$ will be about $\|x\|_p$. Well, there might be multiple large elements, so then we'll hash them to different buckets if $m$ is large enough. Let's write that down formally.

*Claim.* $\|z\|_\infty \in [\frac{1}{2} \|x\|_p, 2 \|x\|_p]$ w.p. $> 3/4$.

(Again, you can get a $1 \pm \epsilon$ approximation by increasing $m$. We'll point out where the analysis changes later.)

*Proof of Claim.* Let $q = \min\left\{\frac{u_1}{|x|^p}, \ldots, \frac{u_n}{|x|^p}\right\}$. We claim that this is an exponential random variable. Indeed,

$$\Pr(q > \lambda) = \Pr(\forall i, u_i / |x_i|^p > \lambda) = \prod_i e^{-\lambda |x_i|^p} = e^{-\lambda \|x\|_p^p}.$$

Therefore, $q \sim \text{Exp}(1)/\|x\|_p^p$. So since $\|z\|_\infty = q^{1/p}$, we have

$$\Pr(\|z\|_\infty \in [\frac{1}{2}\|x\|_p, 2\|x\|_p]) = \Pr(q \in [\frac{1}{2^p}\|x\|_p^{-p}, 2^p \|x\|_p^{-p}]) = e^{-1/2^p} - e^{-2^p} > 3/4,$$

since $p > 2$. □

Now let's finish the proof of Theorem 1. Let's make $T = \|x\|_p$ for convenience. We want to calculate the expected number of $|z_i|$ such that $z > T/(c \log n)$ (call these "heavy") and show that this isn't too large.

*Claim.* Expected number of $i$ such that $|z_i| > T/l$ is at most $l^p$.

*Proof.* Let $E_i$ be an indicator for the random event that $|z_i| \leq T/l$. We want to bound $\mathbb{E} \sum_i E_i$. Recall that $z_i = x_i / u_i^{1/p}$. Then

$$\begin{aligned}\Pr(|z_i| > T/l) &= \Pr(|x_i|^p / u_i > T^p / l^p) = \sum_i \Pr\left(u_i < \frac{l^p |x_i|^p}{T^p}\right) \\ &= \sum_i \left(1 - e^{-\frac{l^p |x_i|^p}{T^p}}\right) \leq \sum_i \frac{l^p |x_i|^p}{T^p} = l^p.\end{aligned}$$

□

So by Markov, the number of $i$ such that $|z_i| > T/l$ is at most $O(l^p)$ with probability 99/100.

Not all of the class has heard of perfect hashing, so let's explain what we mean by that. In perfect hashing, we have a universe of size $n$ and $m$ buckets to hash into, so a hash function $h : [n] \to [m]$. It's random, and there might be collisions. We care about some subset $S \subset [n]$ and we say that $S$ is *perfectly hashed* by $h$ if for all $i \neq j \in S$, $h(i) \neq h(j)$.

*Claim.* As long as $m \gg |S|^2$ and $h$ is 2-wise independent, $S$ is perfectly hashed with high probability (depending on the constant).

We won't prove this, but you define indicator variables for every pair, and you just get the expectation of those indicator variables small enough.

Now recall that $i \in [n]$ is "heavy" if $|z_i| > \frac{T}{c \log n}$. This claim tells us that the number of heavy indices is polylog $n$, so we just need to have $m \gg \text{polylog}^2 n$, and $m$ is polynomial in $n$. This implies that heavy coordinates are all perfectly hashed with probability at least 99/100. (We'll be just using union bound to sum up the probabilities of these events that will happen with 99/100.)

So now we've hashed to the $m$ buckets and all of the heavy coordinates are hashed to different places. Then we'd be done if there weren't light coordinates, too. Let $L$ be the set of light coordinates. These introduce some noise into the buckets, but since $\sigma_i \in \{\pm 1\}$. The expectation is zero, and we could also use the variance, but since we need something stronger, instead, we'll use Chernoff version 2.

Look at one particular $j \in [m]$. We have $\mathbb{E}_{h,\sigma} \sum_{i \in L, h(i)=j} \sigma_i z_i = 0$. We also have that

$$\mathbb{E}_{h,\sigma} \sum_{i \in L, h(i)=j} \sigma_i^2 z_i^2 = \mathbb{E}_{h,\sigma} \sum_{i \in L} \delta_{h(i),j} z_i^2 = \sum_{i \in L} (\mathbb{E}\delta_{h(i),j}) z_i^2 = \sum_{i \in L} \frac{z_i^2}{m} \leq \frac{\|z\|_2^2}{m}.$$

Now expanding $z$,

$$\mathbb{E}\|z\|_2^2 = \sum_{i=1}^n x_i^2 \mathbb{E} \frac{1}{u_i^{2/p}} = \sum_{i=1}^n x_i^2 \cdot O(1).$$

This just a simple calculation, but we need $p > 2$:

$$\mathbb{E}\frac{1}{u_i^{2/p}} = \int_0^\infty e^{-\lambda}\frac{1}{\lambda^{2/p}}.$$

Split the integral at 1 and bound by $\lambda^{-2/p}$ (integrates finitely since $p > 2$) between 0 and 1, and $e^{-\lambda}$ between 1 and infinity.

We haven't seen the $1 - 2/p$ exponent, but now is where it comes in. We need to bound the $p$-norm, but we've bounded the error in terms of the 2-norm. So we need to apply Hölder's inequality:

$$\sum_i f_i g_i \le \left(\sum_i |f_i|^p\right)^{1/p} \cdot \left(\sum_i |g_i|^q\right)^{1/q}, \text{ where } 1/p + 1/q = 1.$$

So we can write

$$\|x\|_2^2 = \sum_i x_i^2 \cdot 1 \le \left(\sum_i (x_i^2)^{p/2}\right)^{2/p} \cdot \left(\sum_i 1\right)^{1-2/p} = n^{1-2/p}\|x\|_p^2.$$

Putting everything together, what we've done is show that the noise in a bucket has expectation 0 and variance $\le n^{1-2/p}T^2/m$, so we just need to choose $m \gg n^{1-2/p}\log n$ because then the variance is $\le \frac{c'T^2}{\log n}$. So then Chebyshev could finish the job, but we'll use Chernoff version 2. The big idea is that the $\infty$-norm of $z$ is roughly correct, so there's some element of $z$ that's the right size. We hash that big guy to a particular bucket, and we need to show that his bucket is still the right size, and that no other buckets get big enough to matter. So we need $\Pr(|\text{noise}| > \frac{T}{10})$ to be small for every bucket.

Our variance $\sigma^2 = \text{Var}(\text{noise}) \le \frac{n^{1-2/p}}{m} \le \frac{c'T^2}{\log n}$, and $\mathbb{E}\text{noise} = 0$. In Chernoff v2, the upper bound on all of the noise terms is $K = \frac{T}{c\log n}$. Therefore, the bound is

$$\max\left\{e^{-\frac{T^2}{10^2}\frac{c\log n}{T^2}}, \left(\frac{\sigma^2}{\lambda K}\right)^{c\lambda/K}\right\}.$$

We cancel a bunch of stuff and get this to be less than some small constant like 1/100. □

## Heavy Hitters

We have a turnstile stream $x_i \leftarrow x_i + v$. We want to *point query*: Given $i$, output $x_i$ with an error of at most $\epsilon\|x\|_1$. We also want to know the "$L_1$ heavy hitters": Output $L \subseteq [n]$ such that if $|x_i| \ge \epsilon\|x\|_1$ then $i \in L$, and if $i \in L$, $|x_i| \ge \frac{\epsilon}{2}\|x\|_1$. We want to compress this into small space, and we only have to remember $x$ within some error.

We'll use a linear sketch for *deterministic* point queries. There isn't randomness in the algorithm today. We have $y = \Pi x$, where $\Pi$ again is an $m \times n$ matrix. Our estimate for a point query will be $\tilde{x} = \Pi^T y = \Pi^T\Pi x$ and we want to claim that $\tilde{x}_i = x_i \pm \epsilon\|x\|_1$ if $\Pi$ is "nice." That is, we estimate $x_i$ with $\Pi_i \cdot y$, where $\Pi_i$ is the $i$th column of $\Pi$.

What does "nice" mean? Here's what we mean:

**Definition.** $\Pi$ is $\epsilon$-incoherent if $\forall i$, $\|\Pi_i\|_2 = 1$ and $\forall i \ne j$, $\left|\left\langle \Pi^i, \Pi^j\right\rangle\right| \le \epsilon$.

That is, $\Pi^T\Pi$ looks a lot like the identity matrix: Its diagonal entries are 1, and off-diagonal entries are at most $\epsilon$ in absolute value. The question becomes, "How do we get $\epsilon$-incoherent $\Pi$?"

One way is error-correcting codes. A code is a set of vectors $\mathcal{C} = \{C_1, \ldots, C_n\}$ where $C_i \in [q]^t$ and $d = \min_{i\ne j}\Delta(C_i, C_j)$, where $\Delta$ measures the number of coordinates of disagreement.

Break the $i$th column of $\Pi$, a $m$-dimensional vector, into $t$ blocks of size $q$. Put a $1/\sqrt{t}$ in the position of each block corresponding to what the symbol is in each block of the code. So if the $i$th codeword $C_i = (\alpha_1, \ldots, \alpha_t)$ where $\alpha_j \in [q]$, $\Pi_{(j-1)q+\alpha_j, i} = 1/\sqrt{t}$. Observe that these are $\epsilon$-incoherent, where $\epsilon = 1 - \frac{d}{t}$. So if a code with large enough distance exists, we have an $\epsilon$-incoherent matrix.

Lastly, let's see why incoherent matrices take point queries.

*Claim.* If $\Pi$ is $\epsilon$-incoherent, $|\tilde{x}_i - x_i| \leq \epsilon \|x\|_1$.

*Proof.* It's really simple: $\tilde{x}_i = \left\langle \Pi^i, \sum_{j=1}^n x_j \Pi^j \right\rangle = x_i + \sum_{j \neq i} \left\langle \Pi^i, \Pi^j \right\rangle x_j$ which is at most $\epsilon \|x\|_1$ away from $x_i$ □

Now, we just need to know that codes exist. If you use a random code, Chernoff version 2 can give a code with $q = O(1/\epsilon)$, $t = O(\frac{1}{\epsilon} \log n)$ and $m = O(\frac{1}{\epsilon^2} \log n)$.

# Tuesday, September 17, 2013

Problem set 2 was due this morning, and problem set 1 is graded and will be handed back after class. There was one set of 2 points that no one got, though. We were working over $\mathbb{F}_{2^r}$ and wanted to evaluate a polynomial $p : \mathbb{F}_{2^r} \to \mathbb{F}_{2^r}$ of degree less than 4 in constant time. The problem is that doing multiplication over $\mathbb{F}_{2^r} \to \mathbb{F}_{2^r}$ isn't necessarily constant time. The standard thing to do is to think of elements as polynomials over $\mathbb{F}_2$ and multiplying them modulo an irreducible degree $r$ polynomial. We were expected to throw a red flag here about how to evaluate in constant time.

Instead, we were supposed to work over a field $\mathbb{F}_p$ where $p$ is a prime, where $p = n/\epsilon$ to get a $(1 \pm \epsilon)$-approximate algorithm. Look at a SODA '04 paper of Thorup and Zhang for the details, which showed up as PSet 1 Problem 3.

Today, we're going to look at turnstile streams. We have $x \in \mathbb{R}^n$ and updates of the form $x_i \leftarrow x_i + v$, where $v \in \mathbb{R}$ can be positive or negative. We'll look at three problems:

1. Point query. Given $i$, output $x_i$ with some bounded error.
2. Heavy hitters. We'll want to find $\varphi$-HH$^p = \{i : |x_i|^p \geq \varphi \|x\|_p^p\}$.
3. Sparse approximation. Recover sparse $\tilde{x}$ such that $\|x - \tilde{x}\|$ is small (we'll specify this later).

We'll do all of these things via two different algorithms, the count-min sketch and the count sketch. To make sure we're all on the same page, we already covered point query in Thursday's lecture and in PSet 2. These two new ones will be randomized, so they'll give probabilistic guarantees.

## CountMin

This was introduced by Cormode and Muthukrishnan in J. Algorithms '05. Yes, C and M are their initials, too, but if you ask them about it, they don't respond.

We start with a $w \times t$ grid, each location with a counter $C_{ij}$ for $i \in [t]$ and $j \in [w]$, so we'll need $tw$ space. We have a hash function for each row $h_1, h_2, \ldots, h_t : [n] \to [w]$. These are drawn independently from a 2-wise family. When we update, we'll add $v$ to all of the counters at $(j, h_i(j))$. We'll maintain that $x_i \geq 0$ always, called the "strict turnstile" assumption.

When we want to point-query $i$, we'll output $pq(i) = \min_{j \in [t]} C_{j, h_j(i)}$.

*Claim.* If $t \geq \log_2(1/\delta)$ and $w \geq 2/\epsilon$, then $\Pr(pq(i) = x_i \pm \epsilon \|x\|_1) \geq 1 - \delta$.

Note that this is the same error as the incoherent matrices, but this time, we only need space $\Theta(1/\epsilon)$, not $\Theta(1/\epsilon^2)$. The cost is that it's only a probabilistic guarantee.

*Proof of Claim.* Focus on some $(i, h_j(i))$ counter. We have

$$C_{j,h_j(i)} = x_i + \sum_{r:h_j(r)=h_j(i),r\neq i} x_r = x_i + \sum_{r\neq i} \delta_r x_r,$$

where $\delta_r$ is an indicator function for $h_j(r) = h_j(i)$. The error term is some noise, and by pairwise independence of $h$, its expectation is $\frac{1}{w}\sum_{r\neq i} x_i \leq \frac{\epsilon\|x\|_1}{2}$, so we then apply Markov. So we know that $C_{j,h_j(i)} \geq x_i$ and with probability at least $1/2$, $C_{j,h_j(i)} \leq x_i + \epsilon\|x\|_1$. Then if we repeat this $\log_2(1/\delta)$ times, with probability $1-\delta$, since these are independent. □

What happens when we remove the strict turnstile assumption? We should use the median instead of the minimum. The noise analysis is the same, but we have to use Chernoff at the end instead of Markov. We need that the $h_i$ are chosen iid from the same family.

Now let's talk about how we find the heavy hitters. Recall we want $\varphi$-HH$^1 = \{i : |x_i| \geq \varphi\|x\|_1\}$. Our goal will be to output a list $L$ such that $\varphi$-HH$^1 \subseteq L \subseteq \frac{\varphi}{2}$-HH$^1$.

We claim that if we didn't care about running time, we could just run point-query at every index with $\epsilon = \varphi/4$ and see which ones are big enough, which means at least $3\varphi/4\|x\|_1$. As long as we let $\delta \ll 1/n$, all of them will succeed with some probability by the union bound. If we needed, we could keep a $\|x\|_1$ estimation on the side.

So more formally, we run point query with error $\epsilon = \varphi/4$ and $\delta = 1/(\gamma n)$. For each $i \in [n]$, include $i$ if $pq(i) \geq \frac{3}{4}\varphi\|x\|_1$. Then $w = O(1/\varphi)$ and $t = O(\log n)$ so this uses space $O(\log n/\varphi)$. The output satisfies the conditions with probability $\geq 1-\gamma$.

The huge problem here is that the time to produce $L$ is $\Theta(n\log n/\gamma)$. Can we do it faster? The answer is that yes, we can, and we'll describe that faster algorithm now.

Imagine that we have the elements $1, 2, \ldots, n$ and we build a perfect binary tree on them. Each node will represent all of the vertices in its subtree. We will then run point queries on every row of this binary tree. Partition $I^j = \{\{1, \ldots, 2^j\}, \{2^j+1, \ldots, 2^{j+1}\}, \ldots\}$. Then we think of $x$ in level $j = 0, 1, \ldots, \log_2 n$ as being $x^j \in \mathbb{R}^{n/2^j}$, where $(x^j)_i$ is the sum of all the $x_r$'s that are in the $i$th partition set of $I^j$.

Then each CountMin sketch will have error $\epsilon = \varphi/4$ and we'll let $\delta = \frac{\log n}{\gamma\varphi}$, aiming for a probability of $1-\gamma$. If we assume the strict turnstile algorithm, then any heavy set containing a heavy hitter will still be heavy. There might be more spurious heavy sets, but in a row we can only have at most $2/\varphi$ apparently-heavy sets. We just walk down the tree, visiting children which point-query says have at least weight $\frac{3}{4}\varphi\|x\|_1$.

How much space does this use? It's something like

$$O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\log n\right) = O\left(\frac{\log n}{\varphi}\left(\log\frac{1}{\gamma\varphi}\right) + \log\log n\right).$$

The time to produce our guess $L$ is then $O\left(\log n\frac{1}{\varphi}\log\left(\frac{\log n}{\gamma\varphi}\right)\right)$.

Finally, let's talk about sparse approximation. We want to recover a $k$-sparse $\tilde{x} \in \mathbb{R}^n$ such that $\|x-\tilde{x}\|_\infty \leq \alpha\left\|x_{\text{tail}(k)}\right\|_1$, where $x_{\text{tail}(k)}$ is $x$ with the heaviest $k$ coordinates (in magnitude) zeroed out. This just amounts to proving $\tilde{x}_i = x_i \pm \alpha\left\|x_{\text{tail}(k)}\right\|_1$.

*Claim.* $pq$ on ContMin with $\delta = \frac{1}{\gamma n}$, $w = O(k/\alpha)$ works.

*Proof.* Let $L$ be the top $k$ coordinates in magnitude. If $w = O(k/\alpha)$, we can do the same analysis to see how many other heavy hitters we collide with, and bound again in the same way. □

Finally, let's see the Count Sketch algorithm. It came in a TCS '04 paper by Charikar, Chn, Farach-Colton. You have a $w \times t$ table, and you have two hash functions $h_i : [n] \to [w]$ and $\sigma_i : [n] \to \{-1, 1\}$ for each row. You update with $C_{i,j} = \sum_{r:h_i(r)=j} \sigma_i(r)x_r$. You bound the error of the 2-norm instead of the 1-norm, and this is actually PSet 1 Problem 3, repeating it a bunch of times and taking the median.

# Thursday, September 19, 2013

Today we have a guest lecture by a postdoc named Karthik. We'll start discussing graph problems in the streaming model.

The big iea is that we have updates to our graph on $n$ nodes coming at us in a stream, which could be edge additions or deletions. We can't store the whole graph, in fact, we just have $O(n \operatorname{polylog} n)$ space.

Some of the problems we'll want to solve: connectivity, number of connected components, spanning forest, bipartiteness testing, and $k$-edge connectivity.

Let's start with an easy case: Only edge additions. How would we solve connectivity? We can store a spanning forest in linear space. But if there are edge deletions, it's unclear whether that will disconnect a component if we only have a spanning tree.

The big idea here is that there are algorithms for testing these properties of a graph that take quadratic space. We're going to first sketch the graph to get a smaller graph and then use this or a similar algorithm on that sketch.

There's an important subproblem: We update a vector $x \in \{0, 1, -1\}^n$ and then update with $x_i \leftarrow x_i + v$ through the turnstile stream. Our goal will be to find some $i \in \operatorname{supp}(x)$, while maintaining $O(\operatorname{polylog} n)$ space.

As an easy subproblem, suppose that at the end, we know that $|\operatorname{supp}(x)| = 1$. Then we can maintain $B = \sum_{i=1}^{n} i x_i$ and read off the index as $|B|$, solving the problem in $O(\log n)$ space.

Recall the AMS Sketch: $\forall \epsilon, \delta \in (0, 1/2)$, $x$ updated by the turnstile stream, there exists a sketch to estimate $(1 \pm \epsilon) \|x\|_2$ with probability $1 - \delta$ using space $O(1/\epsilon^2 \log(1/\delta))$.

Pick $g : [n] \to [n]$ from a pairwise independent hash family and define $h : [n] \to [\log n]$ as the least significant bit of $g(i)$, so $\Pr(h(i) = t) = 1/2^t$. Maintain $\log n$ streams $I^1, \dots, I^{\log n}$ so $I^t = \{i \in [n] : h(i) = t\}$. For one of these streams $I^t$, the expected number is $n/2^t$ and $x^t = x|_{I^t}$ has expected support size $|\operatorname{supp}(x)| / 2^t$. So there is some $t \in [\log n]$ such that $\mathbb{E} |\operatorname{supp}(x^t)|$ is constant. But we'd like to find that the support size is exactly 1, so how could we do that?

We can split all of the substreams $I^t$ into some constant number of streams $I^t_j$ for, say, $1 \le j \le 60$, using a hash function from another pairwise independent hash family. Then maintain $B^t_j = \sum_{i \in I^t_j} i x_i$ for each $j, t$. One of these will be size 1 with a reasonably large probability.

Now to make this all work, we keep an AMS sketch for $X^t_j$ for every $j, t$. This will tell us whether $\left|\operatorname{supp}(X^t_j)\right| = 1$ or not if we take $\epsilon = 0.1$, with failure probability $\delta = 1/\operatorname{poly}(n)$. Then the probability any of them fail is at most $60 \log(n)/\operatorname{poly}(n)$, which we can make as small as we want.

Let's analyze this.

*Claim.* There exists $t \in [\log n]$ such that $1 \le |\operatorname{supp}(X^t)| \le 20$ with probability at least $1/2$.

*Proof.* Fix $t$. Then define the indicator variables $Y_1, \dots, Y_n$ for if $i \in I^t$, which are pairwise independent because we used a pairwise-independent hash function. Then

$$\mathbb{E} \left|\operatorname{supp}(X^t)\right| = \frac{1}{2^t} |\operatorname{supp}(X)|, \qquad \operatorname{Var} \left|\operatorname{supp}(X^t)\right| = \operatorname{Var}\left( \sum_{i \in \operatorname{supp}(x)} Y_i \right) = \sum_{i \in \operatorname{supp}(X)} \operatorname{Var}(Y_i),$$

where we used pairwise independence in the last step. We want to use Chebyshev, so we calculate:

$$\operatorname{Var}(Y_i) = \mathbb{E}(Y_i^2) - \mathbb{E}(Y_i)^2 \le \mathbb{E}(Y_i^2) = \frac{1}{2^t} \implies \operatorname{Var}(\left|\operatorname{supp}(X^t)\right|) \le \mathbb{E}(\left|\operatorname{supp}(X^t)\right|).$$

Now fix $t$ such that $6 \le \mathbb{E} |\operatorname{supp}(X^t)| \le 12$ (we're assuming the starting size is at least 12). Then by Chebyshev,

$$\Pr(\left|\left|\operatorname{supp}(X^t)\right| - \mathbb{E} \left|\operatorname{supp}(X^t)\right|\right| > 5) \le \frac{12}{25} < 1/2,$$

so otherwise, $1 \le |\operatorname{supp}(X^t)| \le 17 \le 20$ as desired. □

Finally, picking that $t$, we need to check that there exists $j \in [60]$ with $\left|\text{supp}(X_j^t)\right| = 1$ with probability $2/3$. Indeed, pick the substream where some $e \in \text{supp}(X^t)$ is hashed, and the other 19 elements fall independently, so the probability of any collision is at most $19/60 < 1/3$ as desired.

To wrap this all up, the total probability of failure in the two steps is $1/2 + 1/3 = 5/6$ so we succeed with at least $1/6$ probability. We just need to amplify this by maintaining $\Omega(\log n)$ streams and one of them will succeed with probability $\geq 1 - 1/\text{poly}(n)$.

The total space we end up using is $O(\log^3 n)$ ($\log n$ for each AMS sketch, $\log n$ of those, and $\log n$ iterations in all).

How can we use this to solve connectivity? Well, a well-known connectivity algorithm just says to pick an edge incident to each vertex, then contract along all of those edges and continue until there are no edges. Notice that this takes at most $O(\log n)$ rounds because in each connected component, we contract to at most half its original size, until it has just one node.

We want to represent this with a sketch. We represent the node-edge incidences for every node. So for node $i$, define

$$a_i[j,k] = \begin{cases} 1 & \text{if } (j,k) \in E, i = j < k \\ -1 & \text{if } (j,k) \in E, i = j > k \\ 0 & \text{otherwise.} \end{cases}$$

This captures a matrix representing the graph, with rows $a_i$ and columns corresponding to every pair of nodes. A column is all 0's if it doesn't correspond to an edge and has exactly one 1 and one $-1$ for every edge. This has one very nice property: If $S$ is a subset of the nodes (say, that we want to contract), then

$$E(S, V \setminus S) = \text{supp}(\sum_{i \in S} a_i).$$

Therefore, to find an edge incident to a vertex, just find some $i$ in its support, and use this algorithm. There are two issues here that we can fix. The first is that there might be an error at each vertex in a given round, but we can just repeat $\Omega(\log n)$ times to keep this from happening. Secondly, we have dependencies if we use the same sketch, so just use a different sketch each time. So here's our theorem:

**Theorem.** *There exists a single-pass algorithm that uses $O(n \log^5 n)$ space to solve connectivity and the number of connected components.*

You can also use this to get a spanning forest, since the union of contracted edges contain a spanning forest, which you still have to find, but you only contracted linearly many edges. (In the first round, you contracted linearly many, and in subsequent rounds, the size is half as big as the previous round.)

On the homework, you'll see how to test for bipartiteness. So let's look at $k$-edge connectivity in our last minutes. We need this fact: A graph is $k$-edge connected if and only if it contains $k$ disjoint spanning forests. Slightly stronger:

*Claim.* For $i = 1, \ldots, k$, let $F_i$ be a spanning forest for $E \setminus \bigcup_{j=1}^{i-1} F_j$. Then $G$ is $k$-edge connected if and only if $\bigcup_{i=1}^{k} F_i$ is $k$-edge connected.

The algorithm we'll do uses sketches $I_1, \ldots, I_k$ for finding a spanning forest for $i = 1, 2, \ldots, k$. Then update $I_{i+1}, \ldots, I_k$ to delete the edges in $F_i$. Finally, use Karger's algorithm for min-cut to test $k$-edge connectivity. (You'll have to repeat it so it succeeds.)

# Tuesday, September 24, 2013

Pset 3 is due on Thursday, and you'll get Pset 2 back after class today.

Today is the last day we'll cover streaming algorithms, but we'll start to see the lower bound perspective, so we won't move too far past them. Then we'll move onwards to dimensionality reduction.

The last problem which we'll cover today is related to small-space dynamic programming. The work in this area is fairly recent.

*Example.* Longest increasing subsequence. You have some sequence of integers and want to find the longest length of an increasing subsequence. Very related is the "distance to monotonicity" which is the number of elements we need to remove to get a monotone subsequence. This is just $n$ minus the LIS. There are some related things like "longest common subsequence" but this doesn't directly have any applications Jelani knows of.

These both have a dynamic program, which we'll write in pseudocode: First, give $i$ weight $w(i) = 1$ and prepend/postpend items 0 and $n+1$, respectively, both with infinite weight (since you want to include them). Initialize with $R = \{0\}$, $W(0) = 0$ and $s(0) = 0$.

For $t = 1$ to $n+1$, let $W(t) = W(t-1) + w(t)$, $s(t) = \min\{s(i) + W(t-1) - W(i) : i \in R, x[i] \le x[t]\}$ and add $t$ to $R$. Then output $S(n+1)$.

Here, we store all the optimal solutions to $t$ in $R$ and $s$ keeps the total weight that we've skipped. When we want to jump forward, we need $x[i] \le x[t]$ and we would add $W(t-1) - W(i)$ to the elements we have skipped. The way we've written this, it just gives the weight.

This gives a $O(n^2)$ algorithm. Previously, here's what we known. Let's say that $x[i] \in [m]$ for all $i$.

- A 1-pass deterministic streaming algorithm which is a $(1 \pm \epsilon)$-approximation in space $O(\sqrt{n/\epsilon} \log m)$. This was in Gopalan, Jayram, Krauthgamer, Kumar '07.
- Any deterministic 1-pass $(1+\epsilon)$-approximation algorithm requires $\Omega(\sqrt{n/\epsilon} \log(m/\epsilon n))$ space. It's still open whether there is a randomized algorithm in polylog space.
- A solution to DTM in a 1-pass streaming algorithm to $(1 \pm \epsilon)$-approximate which succeeds with probability $1 - \frac{1}{n^3}$. This is by Saks and Seshadhri this year (2013).

What's their weird algorithm? The idea is that they randomly throw away parts of $R$, i.e. for each $i \in R$, remove $i$ from $R$ with probability $p(i,t)$ which we'll define. There are two claims we want:

*Claim.* Wtih probability $1 - 1/\operatorname{poly}(n)$, $r(n+1) \le (1+\epsilon)s(n+1)$. With probability $1 - 1/\operatorname{poly}(n)$, $|R| \le O(\log^2 n/\epsilon)$.

To prove this, we need $p(i,t)$. First define

$$q(i,t) = \min\left\{1, \frac{1+\epsilon}{\epsilon} \cdot \ln(4t^3/\delta) \frac{w(i)}{w([i,t])}\right\}$$

and then let $p(i,i) = 1$ and for $t > i$, $p(i,t) = q(i,t)/q(i,t-1)$. In this way, the probability of still being in $R$ at step $t$ is $q(i,t)$. Notice also that this probability goes down as $t - i$ grows. Let's prove these claims.

*Proof of Claims.* Let $C$ be the indices of some optimal LIS. We'll just compare how far off we get from $C$. Define $i \in C$ to be "unsafe" at time $t$ if $C \cap [i,t] \cap R^t = \emptyset$. Being unsafe is a random event in this way, and let $U = \bigcup_{t=1}^{n+1} Ut$, all the indices that every become unsafe.

**Lemma.** *$r(n+1) \le w(\bar{C} \cup U)$, which is the optimal plus the weight of $U$.*

*Proof of Lemma.* By induction we will show this for all $i \in C$: $r(i) \le w(\bar{C}_{\le t-1} \cup U^{t-1})$. Let's show the inductive step. There are a couple cases. In case 1, $U^{t-1} = C_{\le t-1}$, i.e. the whole chain is unsafe at time $t-1$. Then $w(\bar{C}_{\le t-1} \cup U^{t-1}) = w([1, \dots, t-1]) \ge r(t)$.

In case 2, there is some $j \in U^{t-1} \setminus C_{\le k-1}$, so let $j < t$ be the largest such index remembered by $R^t$. Then $x[j] \le x[t]$ because $C$ is a valid IS, so

$$r(t) \le r(j) + w([j+1, t-1]) \le w(\bar{C}_{\le j-1} \cup U^{j-1}) + w([j+1, t-1]) = w(\bar{C}_{\le t-1} \cup U^{t-1}),$$

since everything between $i$ and $t$ is either not in $C$ or unsafe. This completes the proof of the lemma. □

Now we need more definitions. We say that an interval $I \subset [n]$ is "dangerous" if $|C \cap I| \leq \frac{\epsilon}{1+\epsilon}|I|$. In other words, there aren't many elements from our increasing sequence in that interval. We say that an element $i \in [n]$ is "dangerous" if there is a dangerous interval $I$ with $i$ as its left endpoint. Let $D \subset [n]$ be the set of dangerous indices. Finally, define a set $B$ by taking the first dangerous interval, then the next dangerous interval that starts after that one ends, and so on.

We then have a series of claims:

*Claim.* 1. $\bar{C} \subseteq D \subseteq B$.

2. $w(B) \leq (1+\epsilon)w(\bar{C})$.

3. With probability $1 - \delta/2$, $U \subseteq B$.

Now let's see why those claims finish the problem. By the lemma earlier, $r(n+1) \leq w(\bar{C} \cup U)$ and if $U \subseteq B$, this is at most $w(B) \leq (1+\epsilon)w(\bar{C})$, a $1+\epsilon$-approximation of the optimal answer. So now we just need to prove the claims.

*Proof of Claims.* First, $\bar{C} \subseteq D$ is clear since an interval of length 1 with only an element not in $C$ has 0 elements of $C$, so that's clear. For $D \subseteq B$, otherwise there is some dangerous element that dodges $B$, but it would have been chosen in the greedy construction of $B$. So Claim 1 is clear.

For Claim 2, we know that $w(B \cup C) \leq \frac{\epsilon}{1+\epsilon}w(B)$, so $w(\bar{C}) \geq \frac{1}{1+\epsilon}w(B)$, as desired. So only claim 3 remains.

Fix some $t \in [n]$ and $i \in \bar{B} \cap [t]$. We will show that $\Pr(i \in U^t) \leq \delta/4t^3$. This will be enough for the union bound:

$$\begin{aligned}
\Pr(U \subseteq B) &\geq 1 - \sum_{t=1}^{n} \Pr(\bar{B} \cap U^t = \emptyset) \\
&= 1 - \sum_{t=1}^{n} \sum_{i \in \bar{B} \cap [t]} \Pr(i \in U^t) \\
&\geq 1 - \sum_{t=1}^{n} \sum_{i \in \bar{B} \cap [t]} \frac{\delta}{4t^3} \\
&\geq 1 - \frac{\delta}{4} \sum_{t=1}^{n} \frac{1}{t^2} \geq 1 - \frac{\delta}{2}.
\end{aligned}$$

So now we just need to prove that claim. If $i \in \bar{B} \cap [t]$, then $i$ is not dangerous and so $[i,t]$ is not dangerous. Therefore, $\frac{1+\epsilon}{\epsilon}w(C \cap [i,t]) \geq w([i,t])$. But $i \in U^t$ only if we forgot everything in $C \cap [i,t]$ at time $t$. Therefore,

$$\begin{aligned}
\Pr(i \in U^t) &= \prod_{j \in C \cap [i,t]} (1 - q(j,t)) = \prod_{j \in C \cap [i,t]} \left(1 - \frac{1+\epsilon}{\epsilon} \ln\left(\frac{4t^3}{\delta}\right) \frac{w(i)}{w([i,t])}\right) \\
&= \prod_{j \in C \cap [i,t]} \left(1 - \ln\left(\frac{4t^3}{\delta}\right) \frac{w(j)}{w(C \cap [i,t])}\right) \\
&\leq \exp\left(- \sum_{j \in C \cap [i,t]} \ln\left(\frac{4t^3}{\delta}\right) \frac{w(j)}{w(C \cap [i,t])}\right) = \frac{\delta}{4t^3}.
\end{aligned}$$

□

So we've finally finished arguing the correctness of the algorithm. We still need to talk about the space. Space here is basically related to how big $R$ can get, $\max_t |R^t|$, so we'll show that the probability that $R^t$ is large is less than $\delta/2n$, which implies by a union bound that the probability that $\exists t$ such that $R^t$ is large is at most $\delta/2$. So we define the indicator variable $Z_i^t$ for $i \in R^t$, and $|R^t| = \sum_{i=1}^{t} Z_i^t$. Then you just apply the Chernoff bound since we know what these probabilities are. There's really nothing to it. □

This is the last streaming lecture. What are some places where streaming might arrive? There's a package for routers which is used at AT&T for package rejection using the algorithm that we used. Google also analyzes trends in search query logs using the count sketch that we learned. Someone at Dropbox was talking with Jelani about this, and they have server farms that distribute data, and some files are hotter (e.g. Justin Bieber songs) so there's some prioritization to be done there, and they were talking about using heavy-hitters, but given their stats, it wasn't enough to justify the use of that algorithm.

# Thursday, September 26, 2013

Today is our last day of focusing on streaming. We've been doing streaming algorithms before this, and today we'll focus on lower bounds. The main idea here is called *communication complexity*. We'll then prove that it takes $\Omega(1/\epsilon^2 + \log n)$ bits for $F_0$, and that exact median requires $\Omega(n)$ space. Finally, we'll show a high-level approach for showing that you need $\Omega(n^{1-2/p})$ for 2-approximation of $F_p = \sum_{i=1}^{n} |x_i|^p$. This will show that there is something fundamental about crossing $p = 2$ besides simply the existence of $p$-stable distributions.

## Communication Complexity

The model here is that we have two computers, Alice and Bob, who each have some element $x \in X$ and $y \in Y$, respectively, and who are trying to collaborate to compute the value of some function $f : X \times Y \to \{0, 1\}$ at $(x, y)$. The idea is that they get to send messages $m_1, m_2, \ldots$ back and forth and someone then has to declare that they've found the answer. They want to minimize the total number of communicated bits sent across.

A one-way communication lower bound requires that Bob output $f(x, y)$ just given $m_1$. This gives us streaming space lower bounds, because we can consider a streaming algorithm as the process that determines what Alice sends to Bob.

One reason you might want to use streaming is that your memory is on disk and you want to minimize the number of times you have to go through it. Generalizing, $k$ rounds of communication correspond to a $k$-pass streaming algorithm.

There are different types of communication complexities:

- *Deterministic CC* $D(f)$ is the number of bits needed by a deterministic communication protocol that's always correct.
- *Randomized CC with private coins, with error probability* $\delta$ $R_\delta^{pri}(f)$ is the number of bits needed to get $1 - \delta$ success probability where Alice and Bob have private randomness.
- *Randomized CC with public coins, with error probability* $\delta$ $R_\delta^{pub}(f)$ is the number of bits needed to get $1 - \delta$ success probability using a random string written in the sky that everyone can see.

So we clearly have some inequalities: For every $\delta > 0$, $D(f) \geq R_\delta^{pri}(f) \geq R_\delta^{pub}(f)$.

For randomized streaming algorithms, the most relevant model is $R_\delta^{pri}(f)$, because if we're going to use some hash functions, we're going to need to store those, too. (We don't have a sky to write in.) However, for many examples it's easier to get a lower bound for $R_\delta^{pub}(f)$. There are some other relationships between these; read "Communication Complexity" by Kushilevitz and Nisan.

Now let's see why exact deterministic $F_0$ requires $\Omega(n)$ bits. Our function $f = EQ(x, y) = \delta_{xy}$. Then we claim that $D(EQ) \geq n$. The proof is easy pigeonhole.

We've used this to show that distinct elements takes $n$ bits to get exactly. The idea is that Alice puts the support of her string $x \in \{0, 1\}^n$ to Bob, and he counts its size and the size of his own string's support, and then adds his elements to the stream and counts the new size. In this way, he can determine if $x = y$, so $n$ bits must be communicated, as desired.

Now we'll see another thing we've seen before in another way. Suppose Alice gets $x \in \{0,1\}^n$ and Bob gets $j \in [n]$ and $f(x,j) = x_j$.

*Claim.* $R_\delta^{pub}(f) \geq (1 - H_2(\delta))n$, where $H_2(\delta) = -\delta \log_2 \delta - (1-\delta)\log_2(1-\delta)$ is the entropy function.

Recall that you did this on the first problem set using a code. We'll prove it again today using some information theory technology. First, let's define some information theoretic quantities.

**Definition.** Suppose that $X$ takes some values in $\mathcal{X}$. Then $H(x) = -\sum_{x \in \mathcal{X}} p_x \log_2(p_x)$ is the entropy of $X$. The *joint entropy* $H(X,Y) = -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{x,y} \log_2(p_{x,y})$, where $p_{x,y} = \Pr(X = x, Y = y)$. The *conditional entropy* $H(X|Y) = \mathbb{E}_{y \in Y} H(X|Y = y)$. The *mutual information* $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$.

$H(X)$ captures how complex your probability space is. The mutual information captures how much $X$ tells you about $Y$, and vice-versa.

Some properties that are easy to justify:

1. The *chain rule*: $H(X,Y) = H(X) + H(Y|X)$.
2. Subadditivity: $H(X,Y) \leq H(X) + H(Y)$ with equality iff $X, Y$ are independent.
3. $H(X|Y) \leq H(X)$.
4. $0 \leq H(X) \leq \log \mathcal{X}$.
5. Fano's Inequality: Suppose there is a deterministic function $g$ such that $\Pr(g(Y) \neq X) \leq \delta$. Then $H(X|Y) \leq H(X|g(Y)) \leq H_2(\delta) + \delta \log_2(|\mathcal{X}| - 1)$.

Back to the indexing lower bound.

*Proof.* Throughout what remains, we'll use $\Pi$ to denote the transcript of the communication protocol, in this case, just $m_1$. Bob gets $\Pi$ and $j$ and successfully predicts $x_j$.

There's one more notion of complexity, the distributional, $D_{\mu,\delta}(f)$, where $\mu$ is a distribution on inputs and this measure the number of bits that we need to transmit in the best possible random outcome, which is a deterministic algorithm. We have $R_\delta^{pub}(f) \geq D_{\mu,\delta}(f)$ for every distribution $\mu$.

We'll pick what we think will be the worst possible distribution, uniform on $\{0,1\}^n \times [n]$. So Bob gets $\Pi$ and $J \in [n]$ uniformly randomly and is able to predict $x_J$ with probability $\geq 1 - \delta$. Therefore,

$$\begin{aligned} H_2(\delta) \geq H(X_J|\Pi, J) &= \sum_{j=1}^n \Pr(J=j) H(X_J|\Pi, J=j) \\ &= \frac{1}{n}\sum_{j=1}^n H(X_j, \Pi) \geq \frac{1}{n}\sum_{j=1}^n H(X_j|\Pi, X_1, \ldots, X_{j-1}) \\ &= \frac{1}{n}\sum_{j=1}^n H(\Pi, X_1, \ldots, X_j) - H(\Pi, X_1, \ldots, X_{j-1}) \\ &= \frac{1}{n}(H(X_1, \ldots, X_n, \Pi) - H(\Pi)). \end{aligned}$$

Now we can determine $\Pi$ from $X_1, \ldots, X_n$ (which is Alice's input) so $H(X_1, \ldots, X_n, \Pi) = H(X_1, \ldots, X_n)$. The uniform distribution on $X_1, \ldots, X_n$ will have entropy $n$, so

$$H_2(\delta) \geq 1 - \frac{1}{n} H(\Pi) \geq 1 - \frac{1}{n} |\Pi|,$$

and then the result follows by some algebra. □

Now we'll use indexing to prove some lower bounds. First, we'll show that exact median requires $O(n)$ space. The situation is that our input is a bunch of numbers and we want to compute the median exactly wp $\geq 1 - \delta$.

*Claim.* The space needed is at least $(1 - H_2(\delta))n$ if all integers are in $[n]$ and the stream is of length $2n - 1$.

*Proof.* It'll be a reduction to indexing. The only randomness here is in the streaming algorithm. Alice is going to create a virtual stream, run the streaming algorithm for median, send Bob the memory of the stream, and then Bob is going to use that and $j$ to make a new stream.

Alice's virtual stream uses $2i + x_i$ for each $i$, and she runs the median streaming algorithm. Bob then will add to the stream $n - j$ copies of 0 and $j - 1$ copies of $2n + 2$. The median will then be $2j + x_j$, so Bob can extract $x_j$. Bob has just solved indexing using median-streaming, which implies that the lower bound for indexing applies to streaming. □

Of course, median is a problem that improves a lot if we allow multiple passes. Of course, we can't get a better indexing bound because indexing has a trivial 2-pass algorithm. So if we want a bound that holds for $k$-pass algorithms for $k > 1$, we won't be able to index.

**Proposition.** *Distinct elements $F_0$ requires $\Omega(1/\epsilon^2 + \log n)$ bits.*

To do this, we'll use another reduction from a different communication problem, called the *Gap Hamming Problem*: Alice is given $x \in \{0,1\}^N$ and Bob $y \in \{0,1\}^N$. Bob has to take Alice's message and compute the Hamming distance between $x$ and $y$ up to an additive error of $\pm\sqrt{N}$.

One way you'd do this is to sample a bunch of coordinates publicly and Alice sends those bits to Bob and he tests those against his own bitstring. Chernoff bounds will make this a good approximation for the probability with $\epsilon N$ error, but if we want $\sqrt{N}$ error, intuitively that'll require $1/\epsilon^2 = 1/(1/\sqrt{N})^2 = N$ samples.

**Theorem.** $R^{pub}_{1/12}(\textit{Gap Hamming}) = \Omega(N)$.

We won't prove this theorem, but it requires a reduction from indexing.

*Claim.* $F_0$ requires $\Omega(1/\epsilon^2)$ bits of space (as long as $1/\epsilon^2 < n$).

*Proof.* Reduction from GapHam. Set $N = c/\epsilon^2$ and Alice and Bob get $x, y \in \{0,1\}^N$, respectively. Then the number of distinct elements in both supports satisfies $2F_0 = w(x) + w(y) + \Delta(x, y)$, so Bob's estimate for $\Delta(x, y)$ is $2F_0 - w(x) - w(y)$, which he can compute, as desired. □

There's another useful communication problem called $t$-player Disjointness. The situation is a little different: we have $t$ Alices in a row, and want to compute $f(x_1, \ldots, x_t)$, and the space bound is at least the communication lower bound divided by $t - 1$. *Disjointness* takes in $x_1, \ldots, x_t \in \{0,1\}^n$ such that $x_1$ is an indicator function of the set $A_i \subseteq [n]$, and $f(x_1, \ldots, x_t) = 1$ iff for all $i \neq j$, $A_i \cap A_j = \emptyset$.

**Theorem** (Information Theory)**.** $R^{pub}_{1/3}(\textit{Disj}_t) = \Omega(n/t)$ *even when* $|A_i| = cn/t$.

This takes about 3 classes to prove, so we'll skip it and provide a reference.

*Claim.* 2-approximation of $F_p$ requires $\Omega(n^{1-2/p})$ bits of space.

The proof is a reduction from $\text{Disj}_t$ where $t = (2n)^{1/p}$. If $f = 1$, $F_p = \sum_n 1 = n$ and if $f = 0$, $F_p \geq t^p = 2n$, and the space is at least $n/(t(t-1)) \approx n^{1-2/p}$.

# Tuesday, October 1, 2013

Today we're starting Dimensionality Reduction, which has a bit of overlap with streaming algorithms.

The big idea: We have some high-dimensional computational geometry problem, such as clustering, nearest neighbor search, or numerical linear algebra. We reduce the dimensionality of the input while preserving the geometric structure so that (1) our algorithms run faster, and (2) the algorithm is still approximately correct on lower-dimensional transformed input.

Let's talk about the type of geometric structure we want to preserve.

**Definition.** Suppose $(X, d_X)$ and $(Y, d_Y)$ are two metric spaces and $f : X \to Y$. Then $f$ has *distortion* $D_f$ if $\forall x, x' \in X$,

$$C_1 d_X(x, x') \leq d_Y(f(x), f(x')) \leq C_2 d_X(x, x'),$$

where $C_2/C_1 = D_f$.

We'll usually look at normed spaces which have $d_X(x, x') = \|x - x'\|_X$.

First let's state some negative results.

**Theorem** (Brinkman, Charikar, 2005)**.** *There is a set of $N$ points $X$ (in some high dimension), such that if $f : (X, \ell_1) \to (X', \ell_1^m)$ (i.e. norms of the form $\|x\|_1 = \sum_i |x_i|$), then $D_f \leq C$ implies that* $\dim Y \geq N^{\Omega(1/c^2)}$.

Before going back to negative results, let's see a positive result.

**Theorem** (Johnson-Lindenstrauss lemma, 1984)**.** *For any $\epsilon \in (0, 1/2)$ and any set of points $x_1, x_2, \ldots, x_n \in \ell_2$ (finite dimensional), there exists $\Pi \in \mathbb{R}^{m \times n}$ with $m = O(\frac{1}{\epsilon^2} \log N)$ such that*

$$\forall i, j : (1 - \epsilon) \|x_i - x_j\|_2 \leq \|\Pi x_i - \Pi x_j\|_2 \leq (1 + \epsilon) \|x_i - x_j\|_2 .$$

Contrast these two results; one says that you'll always be polynomial in $N$ for constant distortion, while the other says you can get arbitrarily small distortion with only logarithmic space. This is the difference between $\ell_1$ and $\ell_2$. Now let's see one more negative result.

**Theorem** (Johnson-Naor, 2009)**.** *Suppose $\|\cdot\|_X$ is a normed space such that for every $N$-element subset, there exists a linear map $\Pi$ into $O(\log N)$ dimensions with $O(1)$ distortions. Then every $N$-point subsets of $\|\cdot\|_X$, there is a map into $\ell_2$ with distortion $2^{O(\log^*(n))}$.*

Now we're going to focus our attention on Johnson-Lindenstraus. It has a lot of proofs, but nearly all of them go through the following sublemma.

**Lemma** (Distribution JL)**.** *For all $0 < \epsilon, \delta < 1/2$, $\exists D_{\epsilon,\delta}$ distribution on $\Pi \in \mathbb{R}^{m \times n}$, where $m = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ such that for all $x \in \mathbb{R}^n$,*

$$\Pr_{\Pi \sim D_{\epsilon,\delta}} (\|\Pi x\|_2 \neq [(1 - \epsilon) \|x\|_2 , (1 + \epsilon) \|x\|_2]) < \delta.$$

*Equivalently, up to changing $\epsilon$ by a factor of 2,*

$$\forall \|x\|_2 = 1, \Pr_{\Pi \sim D_{\epsilon,\delta}} \left( \left| \|\Pi x\|_2^2 - 1 \right| > \epsilon \right) < \delta.$$

Now we claim that the distributional JL lemma implies the JL lemma. Set $\delta < \binom{N}{2}^{-1}$ and union bound over the $\binom{N}{2}$ points $x = \frac{x_i - x_j}{\|x_i - x_j\|_2}$.

We'll now focus on proving the Distributional JL lemma. There are various proofs because there are several distributions that work. JL's original proof was to apply a random rotation and project onto the first $m$ coordinates. Other proofs apply it with $\Pi_{ij}$ independent, mean 0, variance $1/m$ and subgaussian (decays faster than Gaussians). For computer science applications, you don't care about this full generality but probably just want the simplest to implement. We'll see more proofs later in the course.

A classic approach is to pick $\Pi$ a random sign entry matrix, i.e. $\Pi_{ij} = \sigma_{ij}/\sqrt{m}$ where $\sigma_{ij} \in \{\pm 1\}$ randomly. Then

$$\|\Pi x\|_2^2 - 1 = \frac{1}{m}\sum_{r=1}^{m}\sum_{i\neq j}\sigma_{r_i}\sigma_{r_j}x_i x_j.$$

You can use the method used to prove the Chernoff bound to finish this, and it'll be on an upcoming homework set.

We'll prove it a different way, in a way that hasn't been taught in a class before, to the best of Jelani's knowledge. We'll go through another tail bound:

**Lemma** (Hanson-Wright Inequality, 1971)**.** *If $\sigma_1, \ldots, \sigma_n \in \{\pm 1\}$ are iid uniform signs and $A = (a_{ij}) \in \mathbb{R}^{n\times n}$. Then*

$$\Pr_{\sigma}\left(\left|\sigma^T A\sigma - \mathbb{E}\sigma^T A\sigma\right| > \lambda\right) \lesssim \exp\left(-\min\left\{\frac{c\lambda^2}{\|A\|_F^2}, \frac{c\lambda}{\|A\|}\right\}\right),$$

*where $\|A\|_F^2 = \sum_{i,j} a_{ij}^2$ and $\|A\| = \sqrt{\lambda_{\max}(A^T A)}$, the largest magnitude of an eigenvalue of $A$ if $A$ is symmetric.*

We claim that Hanson-Wright implies Distributional JL.

*Proof.* We have $\|\Pi x\|_2^2 = \sigma^T A_x \sigma$, where $A_x$ is $\frac{1}{m}$ times a $mn \times mn$ block diagonal matrix where each of the $m$ blocks is $xx^T$. We just have to compute the norms of this.

$$\|A_x\|_F^2 = \frac{1}{m^2}\sum_{r=1}^{m}\sum_{i,j} x_i^2 x_j^2 = \frac{1}{m},$$

and the eigenvalues are just $1/m$ and 0, so $\|A_x\| = 1/m$ as well. So taking $\lambda = \epsilon$, as $\frac{c\lambda^2}{\|A\|_F^2} = cm\epsilon^2 < cm\epsilon = c\lambda/\|A\|$, we get

$$\Pr(\left|\|\Pi x\|_2^2 - 1\right| > \epsilon) \lesssim \exp(-cm\epsilon^2),$$

and we can let this be less than $\delta$ and solve for $m$, as desired. □

So we just need to prove the Hanson-Wright inequality, but this is just an excuse to see a bunch of other cool stuff.

If $X$ is a random real variable, then define $\|X\|_p = (\mathbb{E}|X|^p)^{1/p}$.

**Theorem** (Minkowski's inequality)**.** *$\|\cdot\|_p$ is a norm for $p \geq 1$.*

**Lemma** (Jensen's inequality)**.** *If $\Phi$ is a convex function, then $\Phi(\mathbb{E}X) \leq \mathbb{E}\Phi(X)$.*

A function $\Phi$ is convex if $\forall x, y$ and $t \in [0,1]$, $\Phi(tx + (1-t)y) \leq t\Phi(x) + (1-t)\Phi(y)$. You can see how for finite spaces, the overall inequality would result from this by induction.

We'll need another fact: If $1 \leq p < q$, $\|x\|_p \leq \|x\|_q$.

*Proof.* Use Jensen on $\Phi(z) = |z|^{q/p}$ and you get $(\mathbb{E}|X|^p)^{1/p} \leq (\mathbb{E}|X|^q)^{1/q}$. □

Finally, we need to know about Gaussians.

**Definition.** $N(0,1)$ is a distribution over $\mathbb{R}$ with pdf $f(t) = \frac{1}{\sqrt{2\pi}}e^{-t^2/2}$.

**Proposition.** *If $g \sim N(0,1)$, $\mathbb{E}g^p = 0$ if $p$ is odd, and $\mathbb{E}g^p = \frac{p!}{2^{p/2}(p/2)!}$ if $p$ is even. Note that by Stirling, this is $O(\sqrt{p})^p$.*

We'll need just two more lemmas.

**Lemma** (Concentration of Lipschitz functions of gaussians)**.** *Consider a map $f : \mathbb{R}^n \to \mathbb{R}$ and $g = (g_1, \ldots, g_n)$ a vector of independent Gaussians. Then*

$$\Pr_g(|f(g) - \mathbb{E}f(g)| > \lambda) \leq 2e^{c\lambda^2/\|f\|_{lip}^2}, \tag{1}$$

*where $f_{lip} = \sup_{x,y} \frac{f(x)-f(y)}{\|x-y\|_2}$.*

Note that (1) is equivalent to $\forall p \geq 1$, $\|f(g) - \mathbb{E}f(g)\|_p \lesssim \sqrt{p}\,\|f\|_{lip}$. The reverse direction follows from Markov's inequality, and the forward direction can be proved several ways, like integration by parts (and it might be in the homework). Either way, we only care about getting the tail bound.

We will prove this lemma next class, and we'll then state the next lemma and show how it proves Hanson-Wright.

**Lemma** (Decoupling)**.** *We have*

$$\left\| \sum_{i \neq j} a_{ij}\sigma_i\sigma_j \right\|_p \leq 4 \left\| \sum_{i,j} a_{ij}\sigma_i\sigma'_j \right\|_p,$$

*where $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$ are all independent $\pm 1$.*

This is nice because it removes the quadratic cross-terms. Notice that you add the diagonal back. We'll prove it next time, too.

*Proof of Hanson-Wright.* It suffices to show that

$$\left\| \sum_{i \neq j} a_{ij}\sigma_i\sigma_j \right\|_p \lesssim \sqrt{p}\,\|A\|_F + p\,\|A\|.$$

The expectation of $\sigma^T A\sigma$ is the trace of $A$, so we subtract off the diagonal elements when we subtract the expectation, which is why we get $i \neq j$. By decoupling,

$$\begin{aligned}
\left\| \sum_{i \neq j} a_{ij}\sigma_i\sigma_j \right\| &\leq 4 \left\| \sum_{i,j} a_{ij}\sigma_i\sigma'_j \right\|_p \\
&= \frac{4}{(\mathbb{E}\,|g|)^2} \left\| \sum_{i \neq j} a_{ij}\sigma_i\sigma'_j \mathbb{E}_{g,g'}\,|g_i|\,|g'_i| \right\|_p \\
&\lesssim \left\| \sum_{i \neq j} a_{ij}\sigma_i\,|g_i|\,\sigma'_j\,|g'_j| \right\|_p \\
&= \left\| \sum_{i \neq j} a_{ij} g_i g'_j \right\|_p \\
&= \|\langle Ag', g\rangle\|_p \\
&= \|\|Ag'\|_2\|_p \cdot \|g\|_p \\
&\lesssim \sqrt{p}\,\|\|Ag'\|_2\|_p \\
&\leq \sqrt{p}\left[\|\mathbb{E}\,\|Ag'\|_2\|_p + \|\|Ag'\|_2 - \mathbb{E}\,\|Ag'\|_2\|_p\right].
\end{aligned}$$

We can then use the increasing norms to bound $\mathbb{E}\,\|Ag'\|_2 \leq (\mathbb{E}\,\|Ag'\|_2^2)^{1/2} = (\|A\|_F^2)^{1/2} = \|A\|_F$. For the other term, this looks like the Lipschitz bound we had, on the function $f(g') = \|Ag'\|_2$, which has Lipschitz bound $\|A\|$. This completes the proof. □

# Thursday, October 3, 2013

As a reminder, project proposals are due in 4 weeks, October 31.

Last time, there was a mistaken statement that we should correct. Here is the correct statement:

**Theorem.** *Assume that for every $N$-point set $x_1, \dots, x_N$ in a normed space $\|\cdot\|_X$ we can embed it into a $O(\log N)$-dimensional subspace with $O(1)$ distortion. Then every $k$-dimensional subspace of $\|\cdot\|_X$ embeds into $\ell_2$ with distortion $2^{2^{O(\log^*(k))}}$.*

The takeaway is that if you have good embedding, it looks approximately like $\ell_2$. But we didn't quite make the statement we intended to.

Today:

- Prove Lipschitz concentration and decoupling.
- Prove a lower bound of Alon on dimension for JL: $m \gtrsim \frac{1}{\epsilon^2 \log \frac{1}{\epsilon}} \log N$.
- A more refined upper bound for JL dim which takes the properties of the point set into account. (We'll use the same matrix, but prove more things about it.)

**Theorem** (Pisier '86, more elegant proof by Maurey)**.** *If $X_1, \dots, X_n, Y_1, \dots, Y_n$ are iid $\sim N(0,1)$, $\Phi : \mathbb{R} \to \mathbb{R}$ convex, and $f : \mathbb{R}^n \to \mathbb{R}$ has a gradiant almost everywhere, then*

$$\mathbb{E}_X \Phi(f(X) - \mathbb{E}f(X)) \le \mathbb{E}_{X,Y} \Phi(\frac{\pi}{2} \langle \nabla f(X), Y \rangle).$$

For instance, we'll be interested in the special case $\phi(z) = |z|^p$.

Let's see why this implies Lipschitz concentration. Recall the statement:

**Lemma** (Concentration of Lipschitz functions of gaussians)**.** *Consider a map $f : \mathbb{R}^n \to \mathbb{R}$ and $g = (g_1, \dots, g_n)$ a vector of independent Gaussians. Then*

$$\Pr_g(|f(g) - \mathbb{E}f(g)| > \lambda) \le 2e^{c\lambda^2 / \|f\|_{lip}^2},$$

*where $f_{lip} = \sup_{x,y} \frac{f(x)-f(y)}{\|x-y\|_2}$.*

*Proof that Pisier implies Lipschitz.* Take $\phi(z) = |z|^p$, so

$$\begin{aligned} \mathbb{E}_X |f(x) - \mathbb{E}f(x)|^p &\le \left(\frac{\pi}{2}\right)^p \mathbb{E}_X \mathbb{E}_Y |\langle \nabla f(X), Y \rangle|^p \\ &\le \left(\frac{c\pi}{2}\right)^p \sqrt{p}^p \mathbb{E}_X \|\nabla f(X)\|_2^p, \end{aligned}$$

and the result follows from taking the $p$th root of both sides. □

So it suffices to prove Pisier.

*Proof of Pisier.* We can rewrite the left side as

$$\mathbb{E}_X \phi(\mathbb{E}_Y(f_X - f_Y)) \le \mathbb{E}_{X,Y} \phi(f(X) - f(Y)).$$

This step is known as symmetrization. Then we define a function $g(\theta) = X\sin\theta + Y\cos\theta$, so $\frac{d}{d\theta}g(\theta) = X\cos\theta - Y\sin\theta$. Then we can write our expression as

$$\begin{aligned}
\mathbb{E}_{X,Y}\phi(f(g(\pi/2)) - f(g(0))) &= \mathbb{E}_{X,Y}\phi\left(\int_0^{\pi/2}\frac{d}{d\theta}[f(g(\theta))]d\theta\right)\\
&= \mathbb{E}_{X,Y}\phi\left(\int_0^{\pi/2}\langle\nabla f(g(\theta)), g'(\theta)\rangle\, d\theta\right)\\
&\le \mathbb{E}_{X,Y}\phi\left(\mathbb{E}_{\theta\in[0,\pi/2]}\frac{\pi}{2}\langle\nabla f(g(\theta)), g'(\theta)\rangle\right)\\
&= \mathbb{E}_{X,Y}\phi\left(\frac{\pi}{2}\langle\nabla f(X), Y\rangle\right),
\end{aligned}$$

as desired. In the final step, we used the fact that $g(\theta), g'(\theta)$ are distributed in the same way as $X, Y$, independent Gaussians. This is a neat trick because it's a rotation by $\theta$ of the vector $(X, Y) \in \mathbb{R}^2$, and Gaussians are centrally symmetric. It's a neat trick. □

Finally, let's prove Decoupling. Recall the statement.

**Lemma** (Decoupling)**.** *We have*

$$\left\|\sum_{i\neq j}a_{ij}\sigma_i\sigma_j\right\|_{L^p_\sigma} \le 4\left\|\sum_{i,j}a_{ij}\sigma_i\sigma'_j\right\|_{L^p_{\sigma,\sigma'}},$$

*where $\sigma = (\sigma_1, \dots, \sigma_n)$ and $\sigma' = (\sigma'_1, \dots, \sigma'_n)$ are all independent $\pm 1$.*

We added the specification of what random variables the norms are using because it'll get confusing.

*Proof.* Introduce some iid 0-1 variables $\eta_1, \dots, \eta_n$, probability 1/2 on each. Then

$$\left\|\sum_{i\neq j}a_{ij}\sigma_i\sigma_j\right\|_{L^p_\sigma} = 4\left\|\mathbb{E}_\eta\sum_{i\neq j}a_{ij}\sigma_i\eta_i\sigma_j(1-\eta_j)\right\|_{L^p_\sigma} \le 4\left\|\sum_{i\neq j}a_{ij}\sigma_i\eta_i\sigma_j(1-\eta_j)\right\|_{L^p_{\sigma,\eta}}.$$

Since it holds on average, this inequality holds for some fixed $\eta' \in \{0,1\}^n$. Then let $S = \{i : \eta'_i = 1\}$, and

$$\begin{aligned}
\left\|\sum_{i\neq j}a_{ij}\sigma_i\sigma_j\right\|_{L^p_\sigma} &\le 4\left\|\sum_{i\in S}\sum_{j\notin S}a_{ij}\sigma_i\sigma_j\right\|_{L^p_\sigma}\\
&= 4\left\|\sum_{i\in S}\sum_{j\notin S}a_{ij}\sigma_i\sigma'_j\right\|_{L^p_{\sigma_S,\sigma'_{\bar S}}}\\
&= 4\left\|\mathbb{E}_{\sigma_{\bar S}\sigma'_S}\sum_{i,j}a_{ij}\sigma_i\sigma'_j\right\|_{L^p_{\sigma_S,\sigma'_S}}\\
&\le 4\left\|\sum_{i,j}a_{ij}\sigma_i\sigma'_j\right\|_{L^p_{\sigma,\sigma'}},
\end{aligned}$$

as desired. □

That finishes the statements we needed from last time. Now we'll see Alon's lower bound, which is almost the JL lower bound but off by a log factor:

**Theorem.** *There exist $x_0, \dots, x_N \in \mathbb{R}^N$ such that any embedding into $\ell_2^m$ with distortion at most $1+\epsilon$ has $m = \Omega\left(\frac{1}{\epsilon^2 \log \frac{1}{\epsilon}} \log N\right)$.*

*Proof.* What's the hard set? It's $x_0 = 0$, $x_i = e_i$, the $i$th coordinate vector. If we think about how we proved the JL inequality, we said that each vector was preserved with high probability, so we can just look at the $\binom{n}{2}$ differences and show that they're all about preserved. This example is sort of the worst case for the union bound of those probabilities.

Let's say that our embedding is $f : X \to \ell_2^m$ and let $v_i = f(e_i)$. Then $\|v_i\| = (1 \pm \epsilon)$ and $\|v_i - v_j\|^2 = \|v_i\|^2 + \|v_j\|^2 - 2\langle v_i, v_j\rangle$. The first is $2(1 \pm \epsilon)$ and $\|v_i\|^2, \|v_j\|^2 = (1 \pm \epsilon)$, so $|\langle v_i, v_j\rangle| = O(\epsilon)$. Then we can renormalize $v_i$ so that $\|v_i\| = 1$.

Define the matrix $\Pi$ consisting of these vectors. So $\Pi^T\Pi$ has main diagonal 1 and off-diagonal entries with absolute value at most $\epsilon$. Call any such matrix a *$\epsilon$-near identity*. So the statement Alon actually shows is:

**Theorem.** *Any $n \times n$ $\epsilon$-near identity must have rank at least $\Omega\left(\frac{1}{\epsilon^2 \log \frac{1}{\epsilon}} \log n\right)$.*

This isn't tight; we can achieve a rank of $O\left(\frac{1}{\epsilon^2} \min\left\{\log n, \left(\frac{\log n}{\log \frac{1}{\epsilon} + \log\log n}\right)^2\right\}\right)$, which we proved on the homework using Reed-Solomon codes. □

*Proof of Alon's theorem.* We'll prove this lemma for the case of a symmetric matrix, which is what we need. In fact, it's still true for nonsymmetric matrices, since you can average them with their transpose and that'll still be $\epsilon$-near identity with at most double the rank.

**Lemma.** *If $A$ is an $\epsilon$-near identity matrix is symmetric, then its rank $m \geq \frac{n}{1+\epsilon^2(n-1)}$.*

*Proof.* Since $A$ is PSD, its nonzero eigenvalues $\lambda_1, \dots, \lambda_m$ are positive. Then

- $\sum_i \lambda_i^2 = \|A\|_F^2 \leq n + n(n-1)\epsilon^2$.
- By Cauchy-Schwarz, $n^2 = \left(\sum_i \lambda_i\right)^2 \leq m \sum_i \lambda_i^2$.

Combining these, $n^2/m \leq n + \epsilon^2 n(n-1)$, so $m \geq \frac{n}{1+\epsilon^2(n-1)}$, as desired. □

That's great when $\epsilon$ is small, $\epsilon \leq \frac{1}{\sqrt{n}}$. We're going to bootstrap that lemma a bit by taking powers of the matrix.

**Lemma.** *If $A = (a_{ij})$ is of rank $m$ and $p : \mathbb{R} \to \mathbb{R}$ is a degree $k$ polynomial, then $p(A)$ has rank at most $\binom{m+k}{k}$.*

*Proof.* Suppose that $v_1, \dots, v_m$ is a basis for the rowspace of $A$, so any row can be written as $A_i = \sum_{r=1}^m \alpha_r v_r$. Then if $p(z) = \sum_{i=0}^k \beta_i z^i$,

$$p(A)_{ij} = \sum_{q=0}^k \beta_q \left(\sum_r \alpha_r v_{r,j}\right).$$

Therefore, the basis for the rowspace of $p(A)$ is $\left\{(v_{1,t}^{d_1}, \dots, v_{m,t}^{d_t})_{t=1}^n\right\}_{d_i \geq 0, \sum_i d_i \leq k}$. We can count the number of elements here using the "stars and bars" formula, and it's $\binom{m+k}{k}$. We're going to use this on $p(z) = z^k$, so we could extract a little more: $\binom{m+k-1}{k}$, since we'll have $\sum_i d_i = k$, but we don't really need that. □

Now we match our parameters: take $k = \log_{1/\epsilon} \sqrt{n} = \frac{\log n}{2 \log \frac{1}{\epsilon}}$ and $p(z) = z^k$. Look at $p(\Pi^i \Pi)$, so $\frac{n}{2} \leq \operatorname{rank}(P(\Pi^T \Pi)) \leq \binom{m+k-1}{k}$. Use the fact that $\binom{a}{b} \leq (ea/b)^b$ and take logs of both sides. Eventually by rearranging, you get $\frac{1}{\epsilon^2} \leq \frac{e(m+k)}{k}$. □

Finally, recall that JL says that a random sign matrix preserves all vectors in a set $T$ of unit $\ell_2$ vectors simultaneously up to $1 \pm \epsilon$ if $m \gtrsim \frac{\log |T|}{\epsilon^2}$. We had applied this to $T = \left\{ \frac{x_i - x_j}{\|x_i - x_j\|} \right\}$. In general, there's a better bound: you can get $m \gtrsim \frac{g^2(T)+1}{\epsilon^2}$, where $g(T) = \mathbb{E}_g \sup_{x \in T} \langle g, x \rangle$ where $g$ is a random Gaussian vector. Notice that this gives us $\log n$ in the numerator when these vectors are pointed in random directions, but if they're in about the same direction, this supremum might be better.

This was proved by [Gordon '88] for entries $N(0,1)/\sqrt{m}$ and [Klartag-Mendelson '05] for entries $\pm 1/\sqrt{m}$.

# Tuesday, October 8, 2013

Reminder: Project proposals are due October 31. There will be more information on how to find a good project proposal and PSet 5 out soon.

Today, we'll talk about more efficient dimensionality reduction.

Before, the big idea with JL was to transform a high-dimensionality problem to lower dimension so that our algorithms run faster. If our old time was some function $T(N, n)$ where $N$ is the number of vectors and $n$ is the dimension, we can do it in $T(N, O(\frac{1}{\epsilon^2} \log N))$ plus the time required to perform the dimensionality reduction. Whatever algorithm you're using should be okay with some changes.

So if your original dimension was high, this could help a lot, but we might also care about the time required to perform JL, which means finding $\Pi$. That's what we'll be finding this lecture and next.

Recall that we had $\Pi \in \mathbb{R}^{m \times n}$ with $\Pi_{ij} = \pm 1/\sqrt{m}$, independent in some fashion. The time to compute $\Pi x$ is $O(mn)$. More precisely, it's $O(m \|x\|_0)$, where $\|x\|_0 = |\operatorname{supp} x|$. (There are plenty of machine learning applications where $x$ is relatively sparse. For example, some applications to spam recognition think in terms of word frequency, where the dimension is the number of words in your dictionary, and we expect most documents to be sparse.)

This lecture, we'll see two approaches. The first will run fast for sparse vectors and the second will run fast for dense vectors. Bridging that gap is an open research problem. Depending on your circumstance, you might be able to choose ahead of time.

## Sparse Vectors

Our best approach is to make $\Pi$ sparse. If $\Pi$ has exactly $s$ non-zero entries per column, the time to compute $\Pi x$ is $O(s \|x\|_0)$. Let's see how the best approaches compare:

| reference | $m$ | $s$ |
|---|---|---|
| JL, others (84) | $\approx \frac{4}{\epsilon^2} \ln \frac{1}{\delta}$ | $m$ |
| Achtioptas (01) | $\approx \frac{4}{\epsilon^2} \ln \frac{1}{\delta}$ | $m/3$ (exp.) |
| Thorup, Zhang (04) | $\frac{c}{\epsilon^2 \delta}$ | 1 |
| Dasgupta, Kumar and Sarlos (10) | $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ | $\frac{c}{\epsilon} \log^2 \frac{1}{\delta}$ |
| Kane, Nelson (12) | $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ | $\frac{c}{\epsilon} \log \frac{1}{\delta}$ |

We saw Thorup-Zhang on the first problem set, but that's often infeasible because we want to take $\delta < 1/N^2$ or so.

What's the matrix $\Pi$? We don't have so many different ideas. There are two possibilties:

1. We divide the $m$ rows into $s$ blocks and put a random sign $\pm 1/\sqrt{s}$ in each row. So there are $s$ locations with a $\pm 1/\sqrt{s}$.
2. We pick $s$ entries in each column without replacement and put a sign $\pm 1/\sqrt{s}$.

The first instruction is better of implementation purposes because you can write it in terms of hash functions $h : [n] \times [s] \to [m/s]$ and $\sigma : [n] \times [s] \to \{\pm 1\}$. The second is more annoying. In both cases, $\Pi_{ij} = \frac{1}{\sqrt{s}}\delta_{ij}\sigma_{ij}$.

Recall Count Sketch, where we build a $L \times W$ matrix and map $h_1 : [n] \to [w]$ and $\sigma_f : [n] \to \{\pm 1\}$. In our first construction, we'll use $L = S$ and $W = M/S$.

*Claim* (1). With the same notation,

$$m = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}), \quad s = O(\frac{1}{\epsilon}\log\frac{1}{\delta}).$$

Unfortunately, there's some bad news.

*Claim* (Nelson, Ngyuen). There exist $N$ vectors such that any $\Pi \in \mathbb{R}^{m\times n}$ with $m = O(\frac{1}{\epsilon^2}\log N)$ and with $s$ nonzero elements in each column, as long as $m = O(n/\log(1/\epsilon))$.

*Proof of Claim 1.* WLOG $\|x\|_2 = 2$. Then $(\Pi x)_r = \frac{1}{\sqrt{s}}\sum_{i=1}^n \delta_{ri}\sigma_i x_i$, so when we take the norm squared,

$$\|\Pi x\|^2 = \frac{1}{s}\sum_{r=1}^n \left(\sum_{r=1}^n \delta_{ri}x_i^2\right).$$

The diagonal term is $s$, so we want to have $\Pr_\Pi\left(\left|\|\Pi x\|^2 - 1\right| > \epsilon\right) < \delta$. This error term is equal to $\frac{1}{s}\sum_{r=1}^m\sum_{i\neq j}\delta_{ri}\delta_{rj}x_ix_j =: Z$, and with probability $1-\delta$, $\|\Pi x\|^2 - 2 = \sigma^T A_x\sigma - \mathbb{E}\sigma^T A_x\sigma$ where $A_x$ is a block matrix with matrices $x^{(r)}x^{(r)T}$ where $x^{(r)} = (\delta_{ri}x_i)_{i=1}^n$. From Hanson-Wright,

$$\Pr_\sigma(\left|\sigma^T A\sigma - \mathbb{E}\sigma^T A\sigma\right| > \epsilon) \lesssim \exp\left(-\min\left\{\frac{c\lambda^2}{\|A\|_F^2}, \frac{c\lambda}{\|A\|}\right\}\right).$$

We'll define a good event $E$ as when $\forall i \neq j \in [n]$, $\sum_{r=1}^m \delta_{ri}\delta_{rj} = O(s^2/m)$. When this happens,

$$\|A_r\|_F^2 = \frac{1}{s^2}\sum_{r=1}^m\sum_{i\neq j}\delta_{ri}\delta_{rj}x_i^2x_j^2 = \frac{1}{s^2}\sum_{i\neq j}x_ix_j\left(\sum_{r=1}^m\delta_{ri}\delta_{rj}\right) \leq O(1/m)\sum_{i\neq j}x_i^2x_j^2 \leq O(1/m)(\sum_i x_i^2)^2 = O(1/m).$$

We also need to calculate the operator norm $\|A\|$. $A$ is block-diagonal with a factor of $1/s$ outside and blocks $A_1, \ldots, A_m$ and the eigenvalues of $A$ are the eigenvalues of its blocks, so $\|A\| = \max_{r\in[m]}\|A_r\| \cdot \frac{1}{s}$. Since $A_r = S_r - D_r$ where $S_r$ is the matrix $x^{(r)}x^{(r)T}$ and $D_r$ is diagonal with entries $\delta_{r1}x_1^2, \ldots, \delta_{rn}x_n^2$. Therefore, by the triangle inequality, $\|A_r\| \leq \|S_r\| + \|D_r\| \leq 1 + 1 = 2$ since $\|D_r\| = \max_i \delta_{ri}x_i^2 \leq \|x\|_\infty^2 \leq 1$ and $S_r$ is rank 1 with one eivenvector with a nonzero eigenvalue, and $\left\|x^{(r)}\right\|^2 \leq \|x\|^2 = 1$. All of this is conditioned on the event $E$ occurring, though. Putting this into Hanson-Wright,

$$\Pr_\sigma(\left|\|\Pi x\|^2 - 1\right| > \epsilon) \lesssim \max\left\{\exp\left(-\min\left\{\frac{c\lambda^2}{\|A\|_F^2}\right\}, \exp\left(\frac{c\lambda}{\|A\|}\right\}\right)\right\}.$$

So we can take $m = \Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ and $s = \Theta(\frac{1}{\epsilon}\log\frac{1}{\delta})$. It only remains to show that event $E$ occurs.

Fix some $i, j$. Suppose $\delta_{r_1,i}, \ldots, \delta_{r_s,i} = 1$. Define random indicator variables $X_1, \ldots, X_s$ for the event that $\delta_{r_kj} = 1$, so $\sum_{r=1}^m \delta_{ri}\delta_{rj} = \sum_{k=1}^s X_k$. We apply Chernoff and get an error probability $\gamma = \delta/n^2$ that works as long as $s^2/m \geq c\log\frac{1}{\delta}$, i.e.

$$s \gtrsim \sqrt{m\log(n/\delta)} = \frac{1}{\epsilon}\sqrt{\log\frac{1}{\delta}\log\frac{n}{\delta}}.$$

We take the Thorup-Zhang matrix with $s = 1$ and $m = O(\frac{1}{\epsilon^2\delta})$ and let $\Pi$ be the product of our matrix and the Thorup-Zhang one. To remove the $\sqrt{\log 1/\epsilon}$,

$$Z := \|\Pi x\|^2 - 1 = \frac{1}{s}\sum_{r=1}^{m}\sum_{i\neq j}\delta_{ri}\delta_{rj}\sigma_{ri}\sigma_{rj}x_i x_j =: \frac{1}{s}\sum_{r=1}^{m} Z_r.$$

We then apply Markov to $Z^l$, expand, and write it in terms of the variables $Z_r$. Just solve for $\mathbb{E}_{r_j}^{l_j}$ using first principles. □

The moral of this story is that we can use things like Hanson-Wright as a black box. But to improve further, you can argue from first principles as much as possible. If you see a paper using lots of black boxes, maybe you can hope to open the boxes up and improve the results.

# Thursday, October 10, 2013

Projects will be presented on December 3 and 5 instead of November 26 and December 3 because Jelani didn't know that you can in fact have presentations during reading period. E-mail him if you don't

Today is the last day of talking about the Johnson-Lindenstrauss lemma itself. After this we'll mostly use it as a black box. Recall that there were two versions, one that was fast for sparse $x$ and one that was fast for dense $x$.

The big idea is that we want a distribution over $\Pi \in \mathbb{R}^{m\times n}$ such that for all $\|x\| = 1$, $\Pr_\Pi(\left|\|\Pi x\|^2 - 1\right| > \epsilon) < \delta$. We obtained this by picking a spares $\Pi$ with $s$ values of $\pm 1/\sqrt{s}$ chosen in each of the $n$ columns.

We were able to do this for $m \lesssim \frac{1}{\epsilon^2}\log\frac{1}{\delta}$ and $s \lesssim \frac{1}{\epsilon}\log\frac{1}{\delta} = \epsilon m$. We had some bad news that said that if $m \leq \text{poly}\,\frac{1}{\epsilon}\log\frac{1}{\delta}$, then $s = \Omega\left(\frac{1}{\epsilon}\frac{\log(1/\delta)}{\log(1/\epsilon)}\right)$.

Ideally, we'd like to reach a multiplication time for $\Pi x$ that is $O(m + \|x\|_0)$, and the bad news says that we can't do that just via sparse $\Pi$. We might prove that if $\Pi$ is a sign matrix (basically, entries are 0 or $\pm 1/\sqrt{s}$), you can't reach this ideal. It's also true for general matrices, but more technical to prove.

This lecture, we'll investigate what we can do for sparse vectors. The first idea is due to Ailon (NOT Alon) and Chazelle in 2006. They came up with the "Fast Johnson-Lindenstrauss Transform" or FJLT. Before saying what $\Pi$ they chose, we'll explain the intuition to another approach besides these random $\Psi$ matrices. We had this vector $x$ and wanted to create a $m\times n$ matrix such that in each row we pick one random location, and then normalize by a factor of $\sqrt{n/m}$. This means we just sample from different coordinates. Call such a matrix $S$. Then $\|Sx\|^2$ is the scale empirical mean of some random $x_i^2$.

What do we expect from picking a random $x_i^2$? The expected value would be $\frac{1}{m}\|x\|^2$, which is what we want, but it'll have high variance, especially if $x$ is concentrated on a few coordinates. But if it's relatively spread out, then it'll work well. So we just need to randomly precondition so that it becomes well-spread out whp.

There's an idea from physics, the uncertainty principle, which says that a signal and its Fourier transform cannot be both well concentrated. The first thing we might try, then, is to first take the Fourier transform, but we might happen to concentrate it then. You can cook up examples where neither is particularly concentrated or spread, i.e. around $\sqrt{n}$ coordinates, so you'd need to sample about $\sqrt{n}$ coordinates. So we need to do something smarter.

Ailon and Chazelle wrote $\Pi = PHD$ (their notation) where $D$ is a random sign matrix with signs $\pm 1$ on the diagonal, $H$ is the Fourier transform, and $P$ is a random sparse matrix. Their multiplication time for $\Pi x$ is $O(n\log n + \frac{1}{\epsilon^2}\log^2(\frac{1}{\delta})\log\frac{n}{\delta})$. We won't prove this exactly, and we'll prove something slightly weaker of $\frac{1}{\epsilon^4}$. They use something slightly denser than the sampling matrix, which we'll use, but it's slightly denser.

For notation, let $y = HDx$ and $z = Sy$ where $S$ is the sampling matrix with one nonzero entry per row of $\pm\sqrt{n/m}$. To make this work, you don't really need to use the Fourier matrix. What properties do we really want?

- We can multiply $H$ quickly (in $O(n \log n)$ time).
- For all $i, j$, $|H_{ij}| = 1/\sqrt{n}$.
- $H$ is orthogonal, so $HH^T = H^T H = I$.

Instead of the Fourier transform, we'll let $H_{ij} = (-1)^{\langle i,j \rangle}/\sqrt{n}$, wher we consider $i, j$ as being in $\{0, \dots, n-1\}$ and write them in binary as $\{0,1\}^{\log_2 n}$. For this to work, make $n$ a power of 2 by padding some zeros if necessary. Then you can check that $H_n = \begin{pmatrix} H_{n/2} & H_{n/2} \\ H_{n/2} & -H_{n/2} \end{pmatrix}$, so we can multiply it in logarithmic time. This is symmetric, and we can verify that it's orthogonal pretty easily, either from the definition or using the recurrence.

*Claim.* $\Pr_D(\|y\|_\infty \geq e\sqrt{\log(2n/\delta)/n}) < \delta/2$.

*Proof.* We have $D = \operatorname{diag}(\alpha_1, \dots, \alpha_n)$ where $\alpha_i = \pm 1$ random. Then $y_i = \sum_{r=1}^n \alpha_r h_{ir} x_r$. So by Markov,

$$\Pr(|y_i| > \lambda) < \lambda^{-p} \mathbb{E} |y_i|^p ,$$

so let's understand these moments. We'll use the trick from a previous lecture of inserting random Gaussians:

$$\begin{aligned}
\left\| \sum_{r=1}^n \alpha_r h_{ir} x_r \right\|_p &= \sqrt{\frac{\pi}{2}} \left\| \mathbb{E}_g \sum_{r=1}^n \alpha_r |g_r| (h_{ir} x_r) \right\|_p \\
&\leq \sqrt{\frac{\pi}{2}} \left\| \sum_r \alpha_r |g_r| h_{ir} x_r \right\|_p \\
&= \sqrt{\frac{\pi}{2}} \left\| \sum_r g_r (h_i x_r) \right\|_p \\
&= \sqrt{\frac{\pi}{2}} \left( \sum_r (h_{ir} x_r)^2 \right)^{1/2} \|g\|_p \\
&\lesssim \sqrt{p/n} \|x\|_2 .
\end{aligned}$$

By the way, the general fact that if $\sigma_i$ are random signs, $\|\sum_i \sigma_i x_i\|_p \lesssim \sqrt{p} \|x\|_2$ is called Khintchine's inequality. Now back to Markov:

$$\Pr\left( |y_i| > e\sqrt{\log(2n/\delta)/n} \right) < \left( \sqrt{\frac{p}{n}} \frac{1}{e} \sqrt{\frac{n}{\log(2n/\delta)}} \right)^p = \left( \frac{p}{e^2 \log(2n/\delta)} \right)^{p/2} .$$

So we'll choose $p = \log(2n/\delta)$, and let $c = e$. Then this is at most $e^{-p} = \frac{\delta}{2n}$, and by the union bound, the claim holds. □

Now condition on this good event.

*Claim.* Conditioning on $\|y\|_\infty \lesssim \sqrt{\log(n/\delta)/n}$, $\left| \|Sy\|^2 - 1 \right| \leq \epsilon$ wp $\geq 1 - \delta/2$.

*Proof.* To prove this, we use one form of the Chernoff bound: If $X_1, \dots, X_n$ are independent and $X = \sum_i X_i$ has mean $\mu$ and variance $\sigma^2$ and $|X_i| \leq K$, then

$$\Pr(|X - \mu| > \lambda) \lesssim \max\{\exp(-c\lambda^2/\sigma^2), \exp(-c\lambda/K)\}.$$

Let $S_{ri} = \sqrt{n/m}\delta_{ri}$ where $\delta_{ri} \in \{0,1\}$ is our random sample with $\sum_i \delta_{ri} = 1$. Then $z = Sy$ and define

$$\begin{aligned} q_r = z_r^2 &= \frac{n}{m}\left(\sum_{i=1}^n \delta_{ri} y_i\right)^2 \\ &= \frac{n}{m}\left[\sum_i \delta_{ri} y_i^2 + \sum_{i\neq j} \delta_{ri}\delta_{rj} y_i y_j\right] \\ &= \frac{n}{m}\sum_i \delta_{ri} y_i^2. \end{aligned}$$

We care about $z^2 = \sum_{r=1}^m q_r$. The $q_r$ are independent, so we'll use this form of Chernoff. So we need $K \leq \frac{n}{m}\|y\|_\infty^2 \leq \frac{\log(n/\delta)}{m}$ and

$$\sigma^2 \leq m\mathbb{E}q_1^2 = \mathbb{E}\frac{n^2}{m}\sum_i \delta_{ri} y_i^4 = \frac{n}{m}\sum_i y_i^4 \leq \|y\|_\infty^2 \frac{n}{m}\sum_i y_i^2 \leq \frac{\log(n/\delta)}{m}.$$

Now we apply Chernoff. We get

$$\Pr(\left|\sum q_{ir} - 1\right| > \epsilon) \lesssim \max\left\{\exp(-c\epsilon^2 m/\log(n/\delta)), \exp(-c\epsilon m/\log(n/\delta))\right\}.$$

We want this to be at most $\delta/2$, so we take $m \approx \frac{1}{\epsilon^2}\log(n/\delta)\log(1/\delta)$.

Putting this all together, we write $\Pi = \frac{1}{r}(\text{sign})\sqrt{\frac{n}{m}}SHD$, where sign is a random sign matrix with $\frac{1}{\epsilon^2}\log\frac{1}{\delta}$ rows so that we get the right target dimension. Therefore, the time to multiply $\Pi x$ is $O(n)$ for $D$, $O(n\log n)$ for $H$, and $O(m) = O(n)$ for $S$, and $O(m^3)$ for the sign matrix. □

Later work tried to improve this $O(m^3)$ part.

- Ailon, Liberty '08 got it down to $O_\gamma(m^{2+\gamma})$.
- Ailon, Liberty '11 and Krahmer, Ward '11 got it down to $O(m)$ just using a sampling matrix $\Pi = SHD$. The novelty was a better analysis. Krahmer and Ward essentially found what was going on. Ailon and Chazelle realized that $HD$ is a preconditioner designed so that sampling will work. The 2011 work found that $SH$ has some nice property (which we'll see when we go to compressed sensing) with high probability and that $D$ will make things work. That nice property is known as the restricted isometry property, which is that it preserves the norms of $k$-sparse vectors.

  There's a downside, though: You get a slightly worse dimensionality reduction. You get $m = O(\frac{1}{\epsilon^2}\log N(\log\log N)^3)$ instead of the original $m = O(\frac{1}{\epsilon^2}\log N)$. There's some recent work of Nelson, Price, Wooters '14 that gets it down to $m = O(\frac{1}{\epsilon^2}\log N(\log\log N)^2)$.

For the rest of the class, we'll get to some things we didn't touch on from JL.

**Definition.** Let $T \subseteq \ell_2$. Define the *doubling constant* $\lambda(T)$ as the smallest $\lambda$ such that any $\ell_2$-ball within $T$ of radius $r$ can be covered by at most $\lambda$ balls of radius $r/2$ centered in $T$. The *doubling dimension* of $T$ is $d = \log_2(\lambda(T))$.

You can prove that $d$-dimensional Euclidean space has doubling dimension $d$. But subsets might have a smaller doubling dimension.

**Lemma.** *Any embedding of $T$ into $\ell_2^m$ with distortion at most 2, then $m \gtrsim d(T)$.*

This is a simple proof. Basically, take any ball in the original space. Look at the images of those points and $m$-dimensional Euclidean space can be covered with a few balls and take the inverse image of those balls back and it'll still cover. You might need to adjust the radii of the balls maybe to like $r/4$ so it works. You can do this for any constant bound on the distortion other than 2.

There's a conjecture related to this:

**Conjecture** (Gupta, Krauthgamer, Lee '03; Lang, Plaut '01)**.** *For any $T$, we can embed $T$ into $\ell_2^m$ with distortion $C$ such that $C \leq f(d(T))$ and $m \leq g(d(T))$.*

There's a whole space we haven't considered. We only used linear maps so far, but you can't prove this conjecture using only linear maps. Take the following set of points: $x_{ij0} = A(in+j)e_{n+1} + e_i$ and $x_{ij1} = A(in+j)e_{n+1} + e_j$, where $A$ is large. So there are pairs of points close to a big ray in the $x_{n+1}$ direction. A linear map must make $\Pi(x_{ij0} - x_{ij1}) = \Pi(e_i - e_j)$ have the right norm, but that's the same setting as Alon's lower bound, which gave us $\Omega(\log n)$. But there is a nice embedding of this set: $f(x_{ij0}) = (A(in+j)n, 0, 1)$ and $f(x_{ij1}) = (A(in+j), 1, 0)$.

Next week, we'll do some numerical linear algebra. It'll all be the same matrices, but we'll call it by different names.

# Tuesday, October 15, 2013

Jelani wrote out a guide to picking problems for a project that he'll send out in a couple days. He'll also set up a wiki so that we can find a partner if we want to.

Today we're moving to a new topic, numerical linear algebra. We'll see some of related topics, too. We'll spend about four lectures on this before moving to compressed sensing.

Today, we'll look at approximate matrix multiplication. Jelani planned to get to (oblivious) subspace embeddings, and least squares regression, but we spent the whole lecture on matrix multiplication. For all of these, we'll develop randomized algorithms that compute what they're supposed to compute faster than deterministic algorithms could do.

Let's start by talking about matrix multiplication. We have matrices $A$, which is $n \times r$, and $B$, which is $n \times p$, and we want to compute $A^T B$. (This awkward formulation will make some later lemmas seem natural.)

The standard algorithm takes time $rnp$, which just has three for loops. You can do faster. If $A$ and $B$ are both square, i.e. $r = n = p$, then you can do it in time $O(n^\omega)$, where $\omega < 2.373\ldots$. Strassen was the first one doing better, and he got $O(n^{\log_2 7})$ by some clever recursion. Coppersmith and Winograd in the '80s got a better exponent, and there have been some recent improvements by Vassilevska-Williams '11 and Stothers in '10. No one uses these, and people still only occasionally use Strassen.

That's square matrices and exact computation, so let's look at approximate matrix multiplication. What do we want?

We want to quickly find matrices $\tilde{A}$, which is $m \times r$ and $\tilde{B}$ which is $m \times p$, where $m \ll n$, such that $\left\| \tilde{A}^T \tilde{B} - A^T B \right\|$ is small. In this lecture, that norm will be the Frobenius norm, and small will be $\epsilon \|A\|_F \|B\|_F$. Recall that $\|X\|_F = (\sum_{i,j} X_{ij}^2)^{1/2}$ (treat the matrix as a vector and take its $\ell_2$ norm).

We'll look at two algorithms for matrix multiplication. The first one will use sampling. Here's the approach: Notice that if $A$ consists of rows $x_1^T, \ldots, x_n^T$ and $B$ consists of rows $y_1^T, \ldots, y_n^T$ (so each of these vectors are column vectors), then $A^T B = \sum_{k=1}^n x_k y_k^T$. Indeed,

$$(A^T B)_{ij} = \sum_k A_{ki} B_{kj} = \sum_{k=1}^n (x_k)_i (y_k)_j = (\sum_{k=1}^n x_k y_k^T)_{ij}.$$

The idea is that we have a these $n$ outer products, and we'll only sample $m \ll n$ of them. We won't use the uniform distribution, though: We'll choose $\tilde{A}^T \tilde{B} = \dfrac{1}{m} \displaystyle\sum_{t=1}^m \frac{x_{k_t} y_{k_t}^T}{p_{k_t}}$.

In terms of matrices, $\tilde{A} = \Pi A$ and $\tilde{B} = \Pi B$, where $\Pi$ has exactly one nonzero entry in each row, chosen according to some distribution. If we pick a column $k$ with probability $p_k$, then we'll put a $1/\sqrt{mp_k}$ entry in that row. The probability distribution isn't uniform; we'll take $p_k \propto \|x_k\|_2 \|y_k\|_2$. Notice that if you were to implement this as a streaming algorithm, you'd need two passes over the data, the first one to determine

the $p_k$, and the second to pick them with that probability. Perhaps it could be solved with some variant of reservoir streaming though.

The claim is that this is going to be an unbiased estimator and that its variance is small. We can't exactly improve this probability by doing a bunch of trials and taking the median since these are matrices, but there is a way we can boost it anyways.

So we have $\tilde{A}^T\tilde{B} = \frac{1}{m}\sum_{t=1}^{m}\frac{x_{k_t}y_{k_t}}{p_{k_t}}$. Call the summand $Z_t$. We'll look at $\mathbb{E}(Z_t)_{ij}$, and that'll tell us about the expectation of this sum by linearity of expectation. We have

$$\mathbb{E}(Z_t) = \frac{1}{m}\sum_{k=1}^{n}\frac{\Pr(k_t = k)x_k y_k^T}{p_k} = \frac{1}{m}A^T B,$$

so the expectation is correct. Now the variance calculation. We have

$$\begin{aligned}\mathbb{E}\left\|\tilde{A}^T\tilde{B} - A^TB\right\|_F^2 &= \sum_{i=1}^{r}\sum_{j=1}^{p}\mathbb{E}((\tilde{A}^T\tilde{B})_{ij} - (A^TB)_{ij})^2 \\ &= \sum_{i,j}\operatorname{Var}\left(\sum_{t=1}^{m}(Z_t)_{ij}\right) = \sum_{i,j} m\operatorname{Var}((Z_t)_{ij})\end{aligned}$$

since the $Z_t$ are independent. And we have

$$\begin{aligned}\operatorname{Var}((Z_t)_{ij}) \le \mathbb{E}(Z_t)_{ij}^2 &= \frac{1}{m^2}\sum_{k=1}^{n}\frac{p_k(x_k)_i^2(y_k)_j^2}{p_k^2} \\ \mathbb{E}\left\|\tilde{A}^T\tilde{B} - A^TB\right\|_F^2 &\le \frac{1}{m}\sum_{i,j}\sum_{k=1}^{n}\frac{(x_k)_i^2(y_k)_j^2}{p_k} \\ &= \frac{1}{m}\sum_{k=1}^{n}\frac{1}{p_k}\|x_k\|^2\|y_k\|^2 \\ &= \frac{1}{m}\left(\sum_{k=1}^{n}\|x_k\|\|y_k\|\right)^2 \text{ (substituting the definition of } p_k) \\ &\le \frac{1}{m}\left(\sum_{k=1}^{n}\|x_k\|^2\right)\left(\sum_{k=1}^{n}\|y_k\|^2\right) \text{ (Cauchy-Schwarz)} \\ &= \frac{1}{m}\|A\|_F^2\|B\|_F^2.\end{aligned}$$

Now when we apply Chebyshev, we get

$$\Pr_{\Pi}\left(\left\|\tilde{A}^T\tilde{B} - A^TB\right\|_F > \epsilon\|A\|_F\|B\|_F\right) < \frac{\mathbb{E}\left\|\tilde{A}^T\tilde{B} - A^TB\right\|_F^2}{\epsilon^2\|A\|_F^2\|B\|_F^2} < \frac{1}{\epsilon^2 m}.$$

So in conclusion, we only need to sample $m = O(1/\epsilon^2)$ outer products to get success with probability 9/10. The reference is Drineas, Kannan, and Mahoney, "Fast Monte Carlo...", the first of three papers in a series.

So we said we could bootstrap the failure probability to $\delta$ efficiently. We could just make $m = O(1/\epsilon^2\delta)$ but we can also do it more efficiently, a $\log(1/\delta)$ multiplier.

We'll see a trick due to Clarkson and Woodruff from STOC '09 (quite recent!). We'll pick $\Pi_1, \ldots, \Pi_t$ where $t = O(\log 1/\delta)$ and form $t$ matrix products $\tilde{A}_1^T\tilde{B}_1, \ldots, \tilde{A}_t^T\tilde{B}_t$. In the past, we had elements and we took the median, but these are matrices. We'd like to compute $\left\|\tilde{A}^T\tilde{B} - A^TB\right\|_F$ and see if that's small, but

that would involve computing $A^T B$. So instead, we'll compare them with each other: Pick the first $j$ you find such that

$$\left\|\tilde{A}_j^T \tilde{B}_j - \tilde{A}_i^T \tilde{B}_i\right\|_F < \frac{\epsilon}{2} \|A\|_F \|B\|_F$$

for more than half of the $i$'s. This is something like choosing a median.

Why does this work? Well, with probability $1-\delta$, more than half of the $j$'s have $\left\|\tilde{A}_j^T \tilde{B}_j - A^T B\right\|_F < \frac{\epsilon}{4} \|A\|_F \|B\|_F$, by the Chernoff bound. Then we just use the triangle inequality to know that for all such $i, j$,

$$\left\|\tilde{A}_j^T \tilde{B}_j - \tilde{A}_i^T \tilde{B}_i\right\|_F \leq \left\|\tilde{A}_j^T \tilde{B}_j - A^T B\right\|_F + \left\|\tilde{A}_i^T \tilde{B}_i - A^T B\right\|_F \leq \frac{\epsilon}{2} \|A\|_F \|B\|_F,$$

so any such $j$ will be counted. Could another one trick us? Nope: If more than half of the $i$'s have $\tilde{A}_i^T \tilde{B}_i$ close to some $\tilde{A}_j^T \tilde{B}_j$, then at least one of these will be close to $A^T B$, and this implies by the triangle inequality that any success will be within $\frac{3\epsilon}{4} \|A\|_F \|B\|_F$ of $A^T B$, as desired.

Now let's see another way to do approximate matrix multiplication related to the Johnson-Lindenstrauss (dimensionality reduction) lemma.

**Definition.** A distribution $D$ over $\mathbb{R}^{m \times n}$ is said to have the $(\epsilon, \delta, p)$-*JL moment property* if for every $x \in \mathbb{R}^n$ with $\|x\| = 1$,

$$\mathbb{E}_{\Pi \sim D} \left| \|\Pi x\|^2 - 1 \right|^p < \epsilon^p \delta.$$

Note that by Markov, this gives us

$$\Pr_{\Pi}\left( \left| \|\Pi x\|^2 - 1 \right| > \epsilon \right) < \frac{1}{\epsilon^p} \mathbb{E}_{\Pi} \left| \|\Pi x\|^2 - 1 \right|^p < \delta.$$

This looks a little bit stronger though. Sometimes you prove some tail bound looking like

$$\Pr_{\Pi \sim D}(\left| \|\Pi x\|^2 - 1 \right| > \epsilon) < \exp(-c(\epsilon^2 m + \epsilon m)).$$

However, this statement is actually equivalent to $D$ having the $(\epsilon, \exp(-c\epsilon^2 m), \epsilon^2 m)$-JL moment property. One direction is just Markov as above, but for the other direction, you can use integration by parts:

$$\mathbb{E} Z^p = -\int_0^\infty x^p (-\varphi(x))\, dx = [x^p (1-\Phi(x))|_0^\infty + p \int_0^\infty x^{p-1}(1-\Phi(x))\, dx = p \int_0^\infty x^{p-1}(1-\Phi(x))\, dx.$$

Then you apply the tail bound on that inequality and you get a moment bound back. Notice that for the moment bound, you need a tail bound like this for every $\epsilon$, because you need to integrate. We won't focus on this too much, but just remember that tail bounds for every $\epsilon$ are equivalent to moment bounds (of this form) for every $\epsilon$.

**Theorem.** *Suppose that $D$ satisfies the $(\epsilon, \delta, p)$-JL moment property for some $p \geq 2$. Then for every $A, B$ with matching numbers of rows,*

$$\Pr_{\Pi \sim D}\left( \left\| (\Pi A)^T (\Pi B) - A^T B \right\|_F > 3\epsilon \|A\|_F \|B\|_F \right) < \delta.$$

*Proof.* The big idea is that the JLMP implies that you preserve vectors, and we'll show that this implies you preserve dot products, and then that that implies that you preserve matrix products.

So suppose that for $a, b \in \mathbb{R}^n$, $\|a\| = \|b\| = 1$. Then $\|a-b\|^2 = \|a\|^2 + \|b\|^2 - 2\langle a, b\rangle$ and $\|\Pi a - \Pi b\|^2 = \|\Pi a\| + \|\Pi b\|^2 - 2\langle \Pi a, \Pi b\rangle$. Therefore, under distribution $D$,

$$\begin{aligned}
\|\langle \Pi a, \Pi b\rangle - \langle a, b\rangle\|_p &= \frac{1}{2} \left\| (\|\Pi a\|^2 - 1) + (\|\Pi b\|^2 - 1) - (\|a-b\|^2 - \|\Pi a - \Pi b\|^2) \right\|_p \\
&\leq \frac{1}{2} \left[ \left\| \|\Pi a\|^2 - 1 \right\|_p + \left\| \|\Pi b\|^2 - 1 \right\|_p + \|a-b\|^2 \left\| \left\| \Pi \left( \frac{a-b}{\|a-b\|} \right) \right\|^2 - 1 \right\|_p \right] \\
&\leq \frac{1}{2} [\epsilon \delta^{1/p} + \epsilon \delta^{1/p} + 4\epsilon \delta^{1/p}] = 3\epsilon \delta^{1/p}.
\end{aligned}$$

Now, let's look at $A^TB$. Again write $A$ as having rows $x_1^T, \ldots, x_n^T$ and $B$ having rows $y_1^T, \ldots, y_n^T$. Define $X_{ij} = \frac{1}{\|x_i\|\|y_j\|}(\langle \Pi x_i, \Pi y_j\rangle - \langle x_i, y_j\rangle)$. So we can write

$$\left\|(\Pi A)^T(\Pi B) - A^TB\right\|_F^2 = \sum_{i=1}^{r}\sum_{j=1}^{p} \|x_i\|^2 \|y_j\|^2 X_{ij}^2.$$

Since $p \geq 2$, $\|\cdot\|_{p/2}$ is still a norm, so we apply the triangle inequality to get

$$\begin{aligned}
\left\|\left\|(\Pi A)^T(\Pi B) - A^TB\right\|_F^2\right\|_{p/2} &= \left\|\sum_{i,j} \|x_i\|^2 \|y_j\|^2 X_{ij}^2\right\|_{p/2} \\
&\leq \sum_{i,j} \|x_i\|^2 \|y_j\|^2 \left\|X_{ij}^2\right\|_{p/2} \\
&\leq (3\epsilon\delta^{1/p})^2 \sum_{i,j} \|x_i\|^2 \|y_j\|^2 \\
\left\|\left\|(\Pi A)^T(\Pi B) - A^TB\right\|_F^2\right\|_{p/2}^{p/2} &= (3\epsilon\delta^{1/p} \|A\|_F \|B\|_F)^p.
\end{aligned}$$

Now we just apply Markov to get a tail bound.

$$\Pr\left(\left\|(\Pi A)^T(\Pi B) - A^TB\right\|_F > \epsilon \|A\|_F \|B\|_F\right) \leq \mathbb{E}\left\|(\Pi A)^T(\Pi B) - A^TB\right\|_F (\epsilon \|A\|_F \|B\|_F)^2 \leq \delta. \qquad \square$$

So we've proved that if we have a $(\epsilon, \delta, p)$-JLMP distribution, that's good enough. How do we pick such a distribution? We'll use the $\Pi$ from PSet 1, Problem 3, with a random sign in each column and $m$ rows, where $m = O(1/\epsilon^2\delta)$ to get a success probability $2/3$. This is nice because it's a 1-pass algorithm, and we've reduced the problem to something we already have some ideas for.

Next time, we'll look at subspace embeddings.

**Definition.** If $V \subseteq \mathbb{R}^n$ is a dimension-$d$ linear subspace, we say that $\Pi$ is an $\epsilon$ *subspace embedding for* $V$ if for every $x \in V$, $\|\Pi x\| = (1 \pm \epsilon)\|x\|$.

We'll see how subspace embeddings relate to a lot of things we've seen. Problem 1 on the current PSet (due Thursday) will basically prove that JL on $\exp(cd)$ vectors implies subspace embeddings.

# Thursday, October 17, 2013

Today, we'll apply approximate matrix multiplication to do things for us:

- Subspace embeddings.
- Least squares regression.
- Low-rank approximation.

We ended last class with the definition of subspace embeddings:

**Definition.** If $W \subseteq \mathbb{R}^n$ is a linear subspace of dimension $d$. An "$\epsilon$-subspace embedding" is a matrix $\Pi \in \mathbb{R}^{m\times n}$ such that for all $x \in W$,

$$(1-\epsilon)\|x\| \leq \|\Pi x\| \leq (1+\epsilon)\|x\|.$$

We claim that for all such $W$, there exists a 0-subspace embedding $\Pi \in \mathbb{R}^{m\times n}$ with $m = d$, and we can't go below to $m < d$.

This is easy to see. For the minimality, if $m < d$, then there will be a nonzero vector of $W$ in $\ker(\Pi)$, and it won't work. You can do $m = d$ by first rotating the subspace to be the span of $\{e_1, \dots, e_d\}$ then projecting on the first $d$ coordinates.

How are you going to find this rotation, though? We could do Gram-Schmidt, but it'll be expensive. We'll need this important fact from linear algebra:

**Theorem.** *Every matrix $A \in \mathbb{R}^{n\times d}$ has a "singular value decomposition" $A = U\Sigma V^T$, where $U \in \mathbb{R}^{n\times r}$ has orthonormal columns, where $r$ is the rank of $A$; $\Sigma$ is $r \times r$ and diagonal, and $V \in \mathbb{R}^{d\times r}$ has orthonormal columns.*

How quickly can we get this?

**Theorem** (Demmel et al)**.** *We can approximate the SVD in time $\tilde{O}(nd^{\omega-1})$.*

Wading into this gets into messy numerical linear algebra, so we'll just assume you could find the SVD exactly in that time. So you can read off a great subspace embedding using the SVD in time $\tilde{O}(nd^{\omega-1})$.

Meanwhile, let's look at least squares. Given $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$, with $n \gg d$. There probably won't be a solution to the equation $Ax = b$, so we want to compute $x^* = \operatorname{argmin}_x \|Ax - b\|$.

The picture when $d = 2$ is that you have a bunch of data that you collected and you want to find a line that minimize the sum of squares distances to the line.

There are theorems in statistics which explain why some people use least squares. For example, you might assume that some parameter of your system is some linear function of $d$ other parameters and you want to recover those coefficients. There's noise in your experiments, though.

If you make a few assumptions about your errors: they have mean 0, constant variance, and are uncorrelated, then the least squares approximator is best. This is a result known as the Gauss-Markov Theorem.

How do we solve least squares? Well, $\{Ax : x \in \mathbb{R}^d\}$ is the column span of $A$, and we just need to project $b$ onto this. If $A = U\Sigma V^T$ is the SVD, then this projection is $Ax^* = UU^TB$, so if we set $x^* = V\sigma^{-1}U^Tb$, then $Ax^* = U\Sigma V^TV\Sigma^{-1}U^Tb = UU^Tb$ as desired. Therefore, if we can find the SVD, we can solve least squares.

How can we use subspace embeddings? Let $\Pi$ be an $\epsilon$-subspace embedding for the span of $b$ and the columns of $A$, of dimension at most $d+1$. Then compute $\tilde{x}^* = \operatorname{argmin}_{x\in\mathbb{R}^d} \|\Pi Ax - \Pi b\|$, but is this preserved? Well, we have

$$(1-\epsilon)\|A\tilde{x}^* - b\| \le \|\Pi A\tilde{x}^* - \Pi b\| \le \|\Pi Ax^* - \Pi b\| \le (1+\epsilon)\|Ax^* - b\| \implies \|A\tilde{x}^* - b\| \le \frac{1+\epsilon}{1-\epsilon}\|Ax^* - b\|,$$

which is optimal. So we can get within a constant factor of optimal. Informally, it's known that you can use $\epsilon$-subspace embeddings for least square regression.

But we want to do all of this without computing the SVD. Let's apply something from the PSet due this morning. We learned that there exist $m = O(d/\epsilon^2)$ rows that work within a factor of $\epsilon$ with probability $1 - 1/\exp(cd)$.

There's a small issue: We have to multiply $\Pi A$, which takes a bunch of time. Can we use fast matrix multiplication? If we use the $O(n^\omega)$ methods, on blocks of size $d \times d$, we get time $O((n/d)d^\omega(m/d)) = O(mnd^{\omega-2})$. But this is worse than the $O(nd^{\omega-1})$ as $m = O(d/\epsilon^2)$. So we might have this subproblem we can solve faster, but we can't get there quickly enough.

Sarlos '06 was the first person to investigate this, and he used the fast Johnson-Lindenstrauss transform. In FJLT, the time to compute $\Pi x$ is $O(n\log n + m^3) \approx O(n\log n + d^3)$, and we can do $\Pi A$ in time $O(nd\log n + d^4)$. You can get rid of the $d^4$ term utilizing some later results like Ailon and Liberty.

There's another approach. Clarkson and Woodruff (2013) showed that you can get $m = O(d\log^6(d/\epsilon)/\epsilon^2)$ with $s = 1$ number of nonzero entries per column. Two other papers, Mahoney and Meng (2013) and Nelson and Nguyen (2013) achieved $m = O(d^2/\epsilon^2)$ and $s = 1$. The proof we'll see hasn't been presented anywhere yet.

A matrix $\Pi$ is an $\epsilon$-subspace embeedding for $W$ iff $\left\|(\Pi U)^T(\Pi U) - I\right\| \leq \epsilon$, where $U \in \mathbb{R}^{n\times d}$ has columns that span $W$.

We then reviewed some linear algebra. Recall that the operator norm $\|A\| = \sup_{\|x\|=1} \|Ax\|$ by definition. If $A = U\Sigma V^T$, then $\|A\|$ is equivalently the largest entry in $\Sigma$, or $\sqrt{\lambda_{max}(A^TA)} = \sup_{\|x\|=1} \left|x^TAx\right|$, if $A$ is symmetric for the last one.

Applying these facts to this case,

$$\left\|(\Pi U)^T(\Pi U) - I\right\| = \sup_{\|x\|=1} \left|x^T(\Pi U)^T(\Pi U)x - x^Tx\right| = \sup_{\|x\|=1} \left|\|\Pi Ux\|^2 - 1\right|.$$

If this is at most $\epsilon$, then all unit norm vectors in $W$ are preserved by $\Pi$ up to error $\epsilon$.

This might look like approximate matrix multiplication, because we're comparing $(\Pi U)^T(\Pi U)$ to $I = U^tU$. Approximate matrix multiplication was even stronger, though. Recall that in the last lecture, we had

$$\Pr(\left\|(\Pi U)^T(\Pi U) - U^TU\right\|_F > \epsilon \|U\|_F^2) < \delta.$$

Notice that $\|U\|_F^2 = d$, so if you apply this with error $\epsilon/d$ using the Thorup-Zhang sketch from PSet 1, you get $m = O(\epsilon^2/d^2)$, as we said would be possible. You can also relax $s = 1$ to $s = O(1/\epsilon)$ and get $m$ down to $d^{1.01}/\epsilon^2$.

So there are two ways people do subspace embedding these days: Use FJLT or this Thorup-Zhang-type sketch. Later in this class, we'll see that we just need an $1/\sqrt{2}$-subspace embedding that also satisfies approximate matrix multiplication.

There are other ways to use subspace embeddings to solve least squares quickly. They'll be iterative algorithms ([Tygert, Rokhlin], [Avron, Maymounkov, Toledo], both recent). We'll essentially use gradient descent, which depends on the condition number, or ratio of its largest and smallest singular values. Let $\Pi$ be a 1/4-subspace embedding for the column span of $A$. Then let $\Pi A = U\Sigma V^T$ be a SVD, and let $R = V\Sigma^{-1}$. For every $x$, $\|x\| = \|\Pi ARx\| = (1 \pm 1/4)\|ARx\|$. This means that $AR =: \tilde{A}$ has good condition number, so gradient descent works well. This gives us an algorithm:

1. Pick $x^{(0)}$ such that $\left\|\tilde{A}x^{(0)} - b\right\| \leq 1.1 \left\|\tilde{A}x^* - b\right\|$ and so on.
2. Iteratively define $x^{(i+1)} = x^{(i)} + \tilde{A}^T(b - \tilde{A}x^{(i)})$. We claim that we'll get within $1 + \epsilon$ of $\left\|\tilde{A}x^* - b\right\|$.

Let's prove this. We have

$$\tilde{A}(x^{(i+1)} - x^*) = \tilde{A}(x^{(i)} + \tilde{A}^T(b - Ax^{(i)}) - x^*) = (\tilde{A} - \tilde{A}\tilde{A}^T\tilde{A})(x^{(i)} - x^*).$$

Indeed, for the last equality, all of the terms match up except $\tilde{A}\tilde{A}^Tb$ and $\tilde{A}\tilde{A}^T\tilde{A}x^*$ but these are equal since $x^*$ is optimal, i.e. $\tilde{x}^*$ is the projection onto $\tilde{A}$ of $b$.

Now let the SVD of $\tilde{A} = U'\Sigma'V'^T$. We have $\tilde{A} - \tilde{A}\tilde{A}^T\tilde{A} = U'(\Sigma' - \Sigma^3)V'^T$, so

$$\begin{aligned}\left\|\tilde{A}(x^{(i+1)} - x^*)\right\| &= \left\|U'(\Sigma' - \Sigma'^3)V^T(x^{(i)} - x)\right\| \\ &= \left\|I - \Sigma'^2\right\| \left\|U'\Sigma'V'T(x^{(i)} - x^*)\right\| \\ &\leq \frac{1}{2}\left\|\tilde{A}(x^{(i)} - x^*)\right\|.\end{aligned}$$

So in every iteration, your error halves, which means you need $O(\log 1/\epsilon)$ iterations. In each iteration, we have to multiply by $AR$. We can multiply by $A$ in time proportional to the number of nonzero entries in $A$, and by $R$ in $d^2$ time. So the dominant term is $\|A\|_0 \log(1/\epsilon)$ plus the time to compute the SVD.

Here we have the downside that $\|A\|_0$ isn't multiplied by $\log(1/\epsilon)$. Can we avoid this? We can, with a third approach due to Sarlos (2006).

First let's introduce some notation. Write $x^* = \operatorname{argmin} \|Ax - b\|$ and $\tilde{x}^* = \operatorname{argmin} \|\Pi Ax - \Pi b\|$. Let $A = U\Sigma V^T$ be its SVD, so $Ax^* = U\alpha$ and let $Ax^* - b = -w$ and $A\tilde{x}^* - Ax^* = U\beta$ since it will be some linear combination of the columns of $A$, i.e. $U$. The optimal value is $opt = \|w\| = \|Ax^* - b\|$.

We have

$$\begin{aligned}\|A\tilde{x}^* - b\|^2 &= \|A\tilde{x}^* - Ax^* + Ax^* - b\|^2 = \|A\tilde{x}^* - Ax^*\|^2 + \|Ax^* - b\|^2 \text{ since they're orthogonal} \\ &= \|\beta\|^2 + opt^2.\end{aligned}$$

So we want to show that $\|\beta\|^2 \leq 2\epsilon opt^2$. Now we have

$$\begin{aligned}\Pi U(\alpha + \beta) = \Pi A\tilde{x}^* = \text{proj}_{\Pi A}(\Pi b) = \text{proj}_{\Pi U}(\Pi b) = \text{proj}_{\Pi U}(\Pi(U\alpha + w)) = \Pi U\alpha + \text{proj}_{\Pi U}(\Pi w) \\ \Pi U\beta = \text{proj}_{\Pi U}(\Pi w) \\ (\Pi U)^T(\Pi U)\beta = (\Pi U)^T \Pi w\end{aligned}$$

Since $\Pi U$ is a subspace embedding, it preserves the norm of every vector up to an error of $1/4$, and so therefore does $(\Pi U)^T$. Therefore,

$$\|\beta\|^2 / 2 \leq \left\|(\Pi U)^T(\Pi U)\beta\right\|^2 = \left\|(\Pi U)^T \Pi w\right\|^2.$$

If $\Pi$ also does approximate matrix multiplication, then we know that because $w$ is orthogonal to the columns of $U$,

$$\begin{aligned}\Pr_\Pi(\left\|(\Pi U)^T \Pi w - U^T w\right\|_2 > \epsilon' \|U\|_F \|w\|_F) < \delta \\ \Pr_\Pi(\left\|(\Pi U)^T \Pi w\right\|^2 > \epsilon'^2 d \|w\|^2) < \delta.\end{aligned}$$

Now take $\epsilon' = \sqrt{\epsilon/d}$ and putting it all together, we get $\|\beta\|^2 / 2 \leq \epsilon \|w\|^2$, so $\|\beta\|^2 \leq 2\epsilon opt^2$, as we needed.

What's the advantage? In the first Thorup-Zhang sketch, you needed $m = O(d^2/\epsilon^2)$ with $s = 1$, and now we have $m = O(d^2 + d/\epsilon)$ still with $s = 1$, the $d^2$ to get subspace embedding, and $d/\epsilon$ to get matrix multiplication.

# Tuesday, October 22, 2013

Today:

- Low-rank approximation
- Mention some things not covered in class
- Some more subspace embedding stuff

## Low rank approximation

Here's the problem: Given $A \in \mathbb{R}^{n \times d}$, we want to compute $A_k = \operatorname{argmin}_{\text{rank}(B) \leq k} \|A - B\|_X$, where we haven't specified the norm used, usually Frobenius or operator.

**Theorem** (Schmidt approximation, 1907)**.** *If the SVD of $A = U\Sigma V^T$ and $\text{rank}(A) = r$ and $\Sigma$ is positive definite with elements $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$, then under $\|\cdot\|_X = \|\cdot\|_F$, $A_k = U_k\Sigma_k V_k^T$, where $U_k, V_k$ are the first $k$ columns of $U, V$, and $\Sigma_k$ is the first $k$ columns and rows of $\Sigma$ (so still diagonal).*

There was an extension of this theorem discovered later:

**Theorem** (Mirsky 1960)**.** *$A_k = U_k\Sigma_k V_k^T$ is the best approximation under any unitarily invariant norm.*

Let's see some applications of low-rank approximation.

1. Principal Component Analysis.
2. Latent Semantic Indexing.

Let's talk about LSI first. It's a technique used in information retrieval. We have $n$ documents and a dictionary of size $d$, and we can make a giant matrix of a weighted count of the occurrences of word $j$ in document $i$. There are problems with this system:

1. Synonymy. Two authors could write about the same thing, but use a different word, like if one was more formal than the other generally.
2. Multiple definitions. "Python" or "surfing" mean two different things and might look similar because they use the same words.

What does LSI do? It computes the SVD of $A$, and then each document is projected onto the span of the top $k$ singular values. Under some modeling assumptions, LSI "performs well." [Papadimitriou et al, JCSS '00]

Our approach today will use subspace embeddings. Recall that last time, we got our best parameters from a matrix that satisfied subspace embeddings as well as low-rank approximation. We're going to do something similar today. We'll compute $\tilde{A}_k$ with rank at most $k$ such that $\left\|A - \tilde{A}_k\right\|_F \leq (1+\epsilon)\left\|A - A_k\right\|_F$.

We'll take the analysis and approach of Sarlos, FOCS '06. The first paper on this to get some decent error bound (optimal plus $\epsilon \left\|A\right\|_F$) was in the Papadimitriou paper and Frieze, Kannan, and Vempala in JACM, '04.

There's a sampling approach here, but the sampling rate depends on row norms of singular vectors, which depend on knowing the SVD. There's some work around estimating those using subspace embeddings, but once you're doing that, it's almost this method anyways.

**Theorem.** *As long as $\Pi \in \mathbb{R}^{m\times n}$ is a subspace embedding with error $\Theta(1)$ for a certain $k$-dimensional subspace and satisfies AMM with error $\sqrt{\epsilon/k}$, then $\left\|A - \text{proj}_{A\Pi^T,k}(A)\right\|_F \leq (1+\epsilon)\left\|A - A_k\right\|_F$, where $\text{proj}_{V,k}(A)$ is the best rank-$k$ approximation to $\text{proj}_V(A)$, i.e. projecting the columns of $A$ to $V$.*

We'll reduce this back down to regression, i.e. doing this is equivalent to it simultaneously solving $k$ regression problems.

**Definition.** If $A = U\Sigma V^T$, its *Moore-Penrose pseudoinverse* is $A^+ := V\Sigma^{-1}U^T$.

Recall that this is the pseudoinverse we dealt with in least square regression.

*Proof.* Write $X$ to be the column span of $\text{proj}_{A\Pi^r}(A_k)$. Define $P$ to be the projection operator onto $X$. Then

$$\begin{aligned}\left\|A - \text{proj}_{A\Pi^T,k}(A)\right\|_F^2 &\leq \left\|A - PA\right\|_F^2 = \left\|U\Sigma V^T - PU\Sigma V^T\right\|_F^2 \\ &= \left\|U\Sigma - PU\Sigma\right\|_F^2 = \left\|U_k\Sigma_k - PU_k\Sigma_k\right\|_F^2 + \left\|U'_{r-k}\Sigma'_{r-k} - PU'_{r-k}\Sigma'_{r-k}\right\|_F^2.\end{aligned}$$

Let's analyze these terms, starting with the second one.

$$\begin{aligned}\left\|U'_{r-k}\Sigma'_{r-k} - PU'_{r-k}\Sigma'_{r-k}\right\|_F^2 &= \left\|(I-P)U'_{r-k}\Sigma'_{r-k}\right\|_F^2 \\ &\leq \left\|U_{r-k}\Sigma_{r-k}\right\|_F^2 = \left\|U_{r-k}\Sigma_{r-k}V_{r-k}^T\right\|_F^2 = \left\|A_{r-k}\right\|_F^2 = \left\|A - A_k\right\|_F^2.\end{aligned}$$

Now it suffices to show that

$$\left\|A_k - PA_k\right\|_F^2 \leq 2\epsilon\left\|A - A_k\right\|_F^2.$$

By the definition of $P$, $PA_k = (A\Pi^T)(A\Pi^T)^+A_k$. Since $PA_k$ is the best rank-$k$ approximation to $A_k$ in the subspace spanned by the columns of $A\Pi^T$, we can be a little clever.

$$\begin{aligned}\left\|A_k - (A\Pi^T)(A\Pi^T)^+A_k\right\|_F^2 &\leq \left\|A_k - (A\Pi^T)(A_k\Pi^T)^+A_k\right\|_F^2 \\ &= \left\|A_k^T - A_k^T(\Pi A_k^T)^+(\Pi A^T)\right\|_F^2 \text{ (now for the clever step)} \\ &= \sum_i \left\|(A_k^T)^{(i)} - A_k^T((\Pi A_k^T)^+(\Pi A)^T)^{(i)}\right\|_2^2.\end{aligned}$$

The superscripted $(i)$ means the $i$th column. This is a vector, so we're doing $k$ different regression problems, one per column: $\min_x \left\|A_k^T x - (A^T)^{(i)}\right\|_2^2$. The optimal $x$ to choose gives $A_k^T x = \text{proj}_{A_k^T}((A^T)^{(i)}) = (A_k^T)^{(i)}$. This $x$, for the new regression problem, $\min_x \left\|\Pi A_k^T x - \Pi(A^T)^{(i)}\right\|_2^2$, is $x = (\Pi A_k^T)^+(A^T)^{(i)}$.

Now let's recall the third least squares analysis from last week. We got the optimal $x$ for $\min_x \|Sx - b\|$ was $x^* = S^+b$. We also wrote $x^* = U\alpha$, $w = b - Sx^*$, and $\tilde{x}^* = \text{argmin}\,\|\Pi Sx - \Pi b\|_2$, then $S\tilde{x}^* - Sx^* = U\beta$. Last time, we showed that $(\Pi U)^T(\Pi U)\beta = (\Pi U)^T\Pi w$. We're going to do this with $U = V_k^T$, and then we get $V_k^T\Pi^T\Pi V_k\beta_i = V_k^T\Pi^T\Pi w$.

For low-rank approximation, we have a bunch of such $w_i$, $\beta_i$, and $\alpha_i$. Let $W = (w_i)_i$. Then going through how these relate to our problem,

$$\begin{aligned}\|w_i\|_2^2 = \left\|(A^T)^{(i)} - (A_k^T)^{(i)}\right\|_2^2 &\implies \sum_i \|w_i\|_2^2 = \|A - A_k\|_F^2 \\ \|\beta_i\|_2^2 = \left\|A_k^T - A_k^T(\Pi A_k^T)^+(\Pi A^T)\right\|_F^2 &\end{aligned}$$

Now if all of the singular values of $\Pi V_k$ are at least $2^{-1/4}$, we have

$$\frac{1}{2}\sum_i \|\beta_i\|^2 \leq \sum_i \left\|V_k^T\Pi^T\Pi V_k\beta_i\right\|^2 = \sum_i \left\|V_k^T\Pi^T\Pi w_i\right\|_2^2 = \left\|V_k^T\Pi^T\Pi W\right\|_F^2.$$

What is this? Well, $w_i$ was orthogonal to the columns of $V_k$ (playing the role of $U$ in our original notation), so this is the error term in approximate matrix multiplication for $\Pi$. Recall that AMM says, if $V_kW = 0$,

$$\Pr_\Pi(\left\|(\Pi V_k)^T(\Pi W)\right\|_F^2 > \epsilon'^2 \|V_k\|_F^2 \|W\|_F^2) < \delta.$$

Here, $\|V_k\|_F^2 = k$ since the columns are orthonormal, so we'll take $\epsilon' = \sqrt{\epsilon/k}$. Then $\left\|V_k^T\Pi^T\Pi W\right\|_F^2 \leq \epsilon \|W\|_F^2$. What is $W$? It's $(A'_{r-k})^T = A^T - A_k^T$, so this error is $\epsilon \|A - A_k\|_F^2$, as desired. □

If this was confusing, Problem 2 on this PSet will help you work with these things and become more proficient. Sarlos, FOCS'06, will help if you're confused. That PSet problem also provides you a potential final project topic, since it could possibly be improved from $2 + \epsilon$ to $1 + \epsilon$.

Now we'll talk about some things that we won't talk about in depth. What we just talked about gives a good low-rank approximation, but every column of $\tilde{A}_k$ is a linear combination of potentially all columns of $A$. In applications, you want to project onto this space, and interpret what you get. If the projections involve linear combinations of nearly all the columns of $A$, then this won't be so feasible. (Think of LSI, where you want to pick out some key words, but not all of them.) So there's been work on finding few columns of $A$, call them $C$, so that $\|A - (CC^+A)_k\|^2$ is small.

The current records so far come from [Bootsidis et al, FOCS'11], who showed that we can take $C$ with about $2k/\epsilon$ columns of $A$, getting error $(1 + \epsilon)\|A - A_k\|_F$. Separately, [Guruswami and Sinop in SODA'12] got $C$ with $\leq \frac{k}{\epsilon} + k - 1$ columns (known to be optimal) so that $\|A - CC^+A\|_F \leq (1 + \epsilon)\|A - A_k\|_F$. This isn't just better, though; $CC^+A$ might not have rank $k$, and we're comparing it to the best rank-$k$ matrix.

Another topic we won't talk about is other ways of getting subspace embeddings. Here are the known ways:

1. Johnson-Lindenstrauss
2. Approximate matrix multiplication
3. The moment method, which we'll see on the PSet.

For the last one, $\Pi$ being a subspace embedding for the columns of $U \in \mathbb{R}^{n\times d}$ iff $\left\|(\Pi U)^T(\Pi U) - I\right\| \leq \epsilon$. Now, if we let $S = (\Pi U)^T(\Pi U)$,

$$\Pr_{\Pi}(\|S - I\| \geq \epsilon) < \frac{1}{\epsilon^l}\mathbb{E}\,\|S - I\|^l \leq \frac{1}{\epsilon^l}\mathbb{E}\mathrm{tr}((S-I)^l), \text{ if } l \in 2\mathbb{Z}.$$

By induction, you can get that for $B \in \mathbb{R}^{n\times n}$,

$$(B^l)_{ij} = \sum_{i_1,\ldots,i_{l+1}:i_1=i,i_{l+1}=j} \prod_{t=1}^{l} B_{i_t i_{t+1}}.$$

# Thursday, October 24, 2013

Jelani will be gone this coming Monday in California for FOCS. If people want to talk about their projects, he'll make himself available for Skype, or you can just send an e-mail.

Here are the course topics:

1. Streaming
2. Dimensionality reduction
3. Numerical linear algebra
4. Compressed sensing
5. External memory algorithms
6. MapReduce/Hadoop

We're halfway through these! Compressed sensing starts with a "compressible" signal $x \in \mathbb{R}^n$, and we want to make few linear measurements of $x$ so that we can approximately recover $x$ given the measurements.

We'll organize our measurement vectors as the rows of a matrix $\Pi \in \mathbb{R}^{m\times n}$, for $m \ll n$. Obviously this will have a nontrivial kernel, so we need to have our notion of compressible to make any progress (otherwise there are just things in $\ker\Pi$ that aren't recovered). So for us, compressible will mean that it's (approximately) sparse in some basis.

*Example.* Images. Consider an image as a $2^N \times 2^N$ matrix. Images aren't sparse in the usual basis, but they are in the "Haar wavelet" basis. The big idea is that if you look at the image, each pixel isn't very different from its neighbors.

It's easier to describe the transform than the explicit basis. You start with breaking the matrix into $2\times 2$ blocks with entries $\begin{pmatrix} p_1 & p_2 \\ p_3 & p_4 \end{pmatrix}$. We then make four bit $2^{N-1}\times 2^{N-1}$ matrices, and put these entries in the table: $\frac{1}{4}\begin{pmatrix} p_1+p_2+p_3+p_4 & p_1-p_2+p_3-p_4 \\ p_1+p_2-p_3-p_4 & p_1-p_2-p_3+p_4 \end{pmatrix}$ (each of these represent one of the submatrices). We would expect the upper left would contain the bulk of the image, so we recurse on that block.

We'll get $\ell_p/\ell_q$ guarantees: given $\Pi x$, we recover $\tilde{x}$ such that $\|x-\tilde{x}\|_p \leq C_{k,p,q}\min_{\|y_0\|\leq k}\|x-y\|_q$. Today, we'll prove this:

**Theorem** (Candés, Romberg, Tao '04, independently Donoho '04)**.** *We have a $\ell_2/\ell_1$ guarantee:* $\|x-\tilde{x}\|_2 \leq O(\frac{1}{\sqrt{k}})\left\|x_{tail(k)}\right\|_1$.

First, let's consider the case of exact sparsity, $\|x\|_0 \le k$, which should be exact. We need that for every $x, x'$ that are $k$-sparse, $\Pi x \neq \Pi x' \implies x - x' \notin \ker(\Pi)$. Define $\Pi_S$ be restricted to the columns of $S$. For exact sparsity, it suffices to have that $\forall S \subseteq [n]$ with $|S| = 2k$, $\Pi_S$ has full column rank. Then we can recover $x$ given $y = \Pi x$ by solving $\min \|z\|_0$ s.t. $\Pi z = y$. That's a linear program, but unfortunately, it's probably NP-hard.

Well, $\Pi_S$ has full column rank iff $\Pi_S^T \Pi_S$ has no zero eigenvalues. Let's make this a little more robust: All eigenvalues of $\Pi_S^T \Pi_S$ are approximately equal.

**Definition.** $\Pi$ satisfies the "$(\epsilon, k)$-restricted isometry property" (RIP) if for all $k$-sparse $x$, $\|\Pi x\|^2 = (1 \pm \epsilon)\|x\|^2$, i.e. $\left\|\Pi_S^T \Pi_S - I\right\| < \epsilon$, for $|S| \le k$.

How can we get RIP matrices?

1. JL on $\binom{n}{k} e^{O(k)}$ vectors, which gets $m \lesssim \frac{k \log(n/k)}{\epsilon^2}$. That's good if you don't care about $\epsilon$ much, and we'll see that $\epsilon = \sqrt{2} - 1$ is enough.
2. Incoherent matrices, which will be explicit (that's good), but will give us $m \ge k^2/\epsilon^2$, which isn't as good.
3. From "first principles," e.g. sampling $m$ rows from the Fourier matrix. This is similar to how we can get JL by using the Fourier with a diagonal sign matrix. Here we don't need that because we want to preserve them all anyways. So you could think about it as a modified version of a previous construction.

We've already seen (1) on the PSet, so let's look at incoherent matrices. Recall the definition refers to the columns $\Pi^i$ of $\Pi$.

**Definition.** An matrix $\Pi$ is $\alpha$-incoherent if $\forall i$, $\left\|\Pi^i\right\|_2 = 1$ and $\forall i \neq j$, $\left|\left\langle \Pi^i, \Pi^j \right\rangle\right| \le \alpha$.

**Lemma** (Gershgorin Circle Theorem)**.** *All eigenvalues of a matrix $A = (a_{ij})$ lie in circles about the $a_{ii}$ of radii $\sum_{j \neq i} |a_{ij}|$ in the complex plane.*

*Proof.* Let $x$ be an eigenvector of $A$, so $Ax = \lambda x$. Let $i$ be such that $|x_i| = \|x\|_\infty$. Then

$$\begin{aligned} \lambda x_i &= \sum_{j=1}^{n} a_{ij} x_j \\ |(\lambda - a_{ii}) x_i| &= \left| \sum_{j \neq i} a_{ij} x_j \right| \\ |\lambda - a_{ii}| &\le \sum_{j \neq i} |a_{ij} x_j / x_i| \le \sum_{j \neq i} |a_i j| , \end{aligned}$$

as desired. □

If $\Pi$ is incoherent, applying this to $A = \Pi_S^T \Pi_S$ with $|S| \le k$, $S \subseteq [n]$, all the eigenvalues (which are real) lie in intervals about 1 of radius $\le \sum_{j \neq i} \left|\left\langle \Pi_S^i, \Pi_S^j \right\rangle\right| \le \alpha(k-1)$, which are RIP if $\alpha = \epsilon/k$.

As an aside, finding explicit RIP matrices with $m \ll k^2$ is a challenging open problem. There's one paper making progress on this by Bourgain, Dilworth, Ford, Konyagin, Kutzarova in STOC 2011 get $m \ll k^{2-\gamma}$ for small $\gamma$ only for $k \approx \sqrt{n}$. Their paper isn't easy, either; it uses analytic number theory and additive combinatorics.

Now let's talk about how you solve that linear program that we said was NP-hard.

**Theorem.** *If $\Pi$ has RIP of order $2k$ with $\epsilon_{2k} \le \sqrt{2} - 1$, then if $\tilde{x} = x + h$ is the solution to the following LP: $\|x - \tilde{x}\|_2 \le O(\frac{1}{\sqrt{k}}) \left\|x_{tail(k)}\right\|_1$.*

Given this theorem, we can then use this LP known as Basis Pursuit: $\min \|z\|_1$ such that $\Pi z = \Pi x$. That is, minimize $\sum y_i$ such that $\Pi z = \Pi x$ and $|z_i| \leq y_i$. You can solve this in polynomial time, e.g. with the ellipsoid algorithm (Khachiyan).

*Proof.* Let's say that the largest $k$ coordinates in magnitude of $x$ are in $T_0 \subseteq [n]$. Then let $T_1$ be the largest $k$ coordinates of $h_{T_0^C}$, and $T_2$ the next $k$ largest, and so on. Then by the triangle inequality,

$$\|h\|_2 = \left\|h_{T_0 \cup T_1} + h_{(T_0 \cup T_1)^C}\right\|_2 \leq \|h_{T_0 \cup T_1}\|_2 + \left\|h_{(T_0 \cup T_1)^C}\right\|_2.$$

Our proof strategy will be to show that $\left\|h_{(T_0 \cup T_1)^C}\right\|_2 \leq \|h_{T_0 \cup T_1}\|_2$, then show that $\|h_{T_0 \cup T_1}\|_2$ is small.

For the first of these, we have

$$\left\|h_{(T_0 \cup T_1)^C}\right\|_2 \leq \sum_{j \geq 2} \left\|h_{T_j}\right\|_2 \leq \sqrt{k} \sum_{j \geq 2} \left\|h_{T_j}\right\|_\infty \leq \frac{1}{\sqrt{k}} \sum_{j \geq 1} \left\|h_{T_j}\right\|_1 = \frac{1}{\sqrt{k}} \left\|h_{T_0^C}\right\|_1.$$

This trick is usually called "shelling." Now we need to use that $\tilde{x}$ is the $\ell_1$ minimizer of our Basis Pursuit LP. This means that

$$\begin{aligned}
\|x\|_1 \geq \|x+h\|_1 = \|(x+h)_{T_0}\|_1 + \left\|(x+h)_{T_0^C}\right\| &\geq \|x_{T_0}\|_1 - \|h_{T_0}\|_1 + \left\|h_{T_0^C}\right\|_1 - \left\|x_{T_0^C}\right\|_1 \\
\left\|h_{T_0^C}\right\|_1 &\leq \|x\|_1 - \|x_{T_0}\|_1 + \|h_{T_0}\|_1 + \left\|x_{T_0^C}\right\|_1 \\
&= 2\left\|x_{T_0^C}\right\|_1 + \|h_{T_0}\|_1 \\
&\leq 2\left\|x_{T_0^C}\right\|_1 + \sqrt{k}\,\|h_{T_0}\|_2,
\end{aligned}$$

applying Cauchy-Schwarz to the $k$-sparse vector $h_{T_0}$. Plugging this back into our previous equation, we get $\left\|h_{(T_0 \cup T_1)^C}\right\|_2 \leq \frac{2}{\sqrt{k}} \left\|x_{T_0^C}\right\|_1 + \|h_{T_0 \cup T_1}\|_2$, so it's within our allowed error of the head term, and we're done with the first part.

Now we just need to bound the head. There's an observation we'll need that we've already seen, so we'll save it here:

$$\sum_{j \geq 2} \left\|h_{T_j}\right\|_2 \leq \frac{1}{\sqrt{k}} \left\|h_{T_0^C}\right\|_1 \leq \frac{2}{\sqrt{k}} \left\|x_{T_0^C}\right\|_1 + \|h_{T_0 \cup T_1}\|_2 \quad (*)$$

Now, we'll need a lemma.

**Lemma.** *If $x, x'$ are supported on disjoint sets $T, T'$ with $|T|, |T'| = k'$, then $|\langle \Pi x, \Pi x' \rangle| \leq \epsilon_{k+k'} \|x\|_2 \|x'\|_2$.*

*Proof.* WOLOG $\|x\|_2 = \|x'\|_2 = 1$. We know that

$$\langle \Pi x, \Pi x' \rangle = \frac{1}{4}[\|\Pi x + \Pi x'\|_2^2 - \|\Pi x - \Pi x'\|_2^2].$$

The terms on the right side are each $2(1 \pm \epsilon_{k+k'})$, so their difference is at most $\epsilon_{k+k'}$. □

Now we have

$$\begin{aligned}
\|\Pi h_{T_0\cup T_1}\|^2 &= \langle \Pi h_{T_0\cup T_1}, \Pi h\rangle - \sum_{j\geq 2}\left\langle \Pi h_{T_0\cup T_1}, \Pi h_{T_j}\right\rangle \\
&\leq \sum_{j\geq 2}\left|\left\langle \Pi h_{T_0\cup T_1}, \Pi h_{T_j}\right\rangle\right| \\
&\leq \epsilon_{2k}\sum_{j\geq 2}(\|h_{T_0}\|_2 + \|h_{T_1}\|_2)\left\|h_{T_j}\right\|_2 \\
(1-\epsilon_{2k})\|h_{T_0\cup T_1}\|_2^2 \leq \|\Pi h_{T_0\cup T_1}\|_2^2 &\leq \sqrt{2}\epsilon_{2k}\|h_{T_0\cup T_1}\|_2\sum_{j\geq 2}\left\|h_{T_j}\right\|_2 \\
(1-\epsilon_{2k})\|h_{T_0\cup T_1}\|_2 &\leq \sqrt{2}\epsilon_{2k}\sum_{j\geq 2}\left\|h_{T_j}\right\|_2^2 \\
&\leq \sqrt{2}\epsilon_{2k}\left(\frac{2}{\sqrt{k}}\left\|x_{T_0^C}\right\|_1 + \|h_{T_0\cup T_1}\|_2\right) \text{ by } (*) \\
(1-\epsilon_{2k}-\sqrt{2}\epsilon_{2k})\|h_{T_0\cup T_1}\|_2 &\leq \frac{2\sqrt{2}\epsilon_{2k}}{\sqrt{k}}\left\|x_{T_0^C}\right\|_1.
\end{aligned}$$

And now if $\epsilon < \sqrt{2}-1$, this is a nontrivial improvement. □

# Tuesday, October 29, 2013

Thomas Steinke, the TF for the class, was teaching today. We'll be talking about comparing RIP to JL.

**Theorem** (PSet 5, Problem 1)**.** *If $D$ is a distribution on $\mathbb{R}^{m\times n}$ satisfies $(\epsilon,\delta)$-JL property, i.e.*

$$\forall x : \|x\|_2 = 1, \Pr_{\Pi\sim D}\left[\left|\|\Pi x\|_2^2 - 1\right| > \epsilon\right] < \delta$$

*for $\delta \ll \binom{n}{k}^{-1}e^{-O(k)}$, then whp, $\Pi\sim D$ satisfies the $(O(\epsilon),k)$ RIP, i.e.*

$$\forall x : \|x\|_0 \leq k, \|\Pi x\|_2^2 = (1+O(\epsilon))\|x\|_2.$$

This shows that JL implies RIP. Can we go the other way? Yes, by adding a bit of randomness, as we'll see today:

**Theorem** (Khramer, Ward '11)**.** *Let $\Pi\in\mathbb{R}^{m\times n}$ satisfy $(\epsilon,2k)$ RIP. Let $\sigma\in\{-1,1\}^n$ uniformly random and $D_\sigma$ be the matrix with $\sigma$ on the diagonal. Then $\Pi D_\sigma$ satisfies the $(O(\epsilon),2^{-\Omega(k)})$ JL property.*

*Example.* If $\Pi = SHD$ where $S$ is a $m\times n$ sample matrix, $H$ Is $n\times n$ Hadamard (or Fourier transform) and $D$ is a diagonal matrix with random $\pm 1$ on the diagonal, we saw that writing $y = HDx$, $\|y\|_\infty$ is small whp, and $\|Sy\|_2 \approx \|y\|_2 = \|x\|_2$. We split it up as $\Pi = (SH)D = S(HD)$. This is a cool theorem because:

*Theorem* (Cheraghchi, Guruswami, Velingker '13). *If $m \gtrsim k/\epsilon^2\log n\log^3 k$, then whp $SH$ is $(\epsilon,k)$ RIP.*

Now, let's prove the theorem.

*Proof.* Let $x\in\mathbb{R}^n$ with $\|x\|_2 = 1$. We have

$$\|\Pi D_\sigma x\|_2^2 = \sum_{i=1}^m\left(\sum_{j=1}^n \Pi_{ij}\sigma_j x_j\right)^2 = \|\Pi D_x\sigma\|_2^2 = \sigma^T(D_x^T\Pi^T\Pi D_x)\sigma = \sigma^T X\sigma,$$

where $X = D_x^T \Pi^T \Pi D_x$. We'd like to apply Hanson-Wright, but we can't do that immediately. We'll apply the shelling trick from before first. WOLOG arrange the coordinates so that $|x_i| \geq |x_{i+1}|$. Break up the matrix $X$ so that the upper $k \times k$ block is $A_1$, then $A_2, \ldots, A_{n/k}$ are further diagonal blocks of size $k \times k$. Let $B$ be the rest of the first $k$ rows, so $B^T$ is the rest of the first $k$ columns, and let $C$ be the rest. We'll analyze each of these in turn.

For $A_i$, we have

$$\begin{aligned}\sigma^T A \sigma &= \sum_{i=1}^{n/k} \sigma_{(i)}^T A_{(i)} \sigma_{(i)} = \sum_{i=1}^{n/k} \sigma_{(i)}^T D_{x_{(i)}}^T \Pi_{(i)}^T \Pi_{(i)} D_{x_{(i)}} \sigma_{(i)} \\ &= \sum_{i=1}^{n/k} \left\| \Pi_{(i)} D_{\sigma_{(i)}} x_{(i)} \right\|_2^2 \\ &= \sum_{i=1}^{n/k} (1 \pm \epsilon) \left\| x_{(i)} \right\|_2^2 = (1 \pm \epsilon) \left\| x \right\|_2^2 = 1 \pm \epsilon.\end{aligned}$$

Great, now we just have to analyze the rest. The $(i)$ subscript means the $i$th block, and $(-i)$ will mean everything but the $i$th block below. We have

$$B = D_{x_{(1)}} \Pi_{(1)}^T \Pi_{(-1)} D_{x_{(-1)}}.$$

Let $u, v$ be unit vectors. Then

$$\begin{aligned}uBv &= \sum_{i>1} u D_{x_{(1)}} \Pi_{(1)}^T \Pi_{(i)} D_{X_{(i)}} v \\ &\leq \sum_{i>1} \left\| u \right\|_2 \left\| D_{x_{(1)}} \right\| \left\| \Pi_{(1)}^T \Pi_{(i)} \right\| \left\| D_{x_{(i)}} \right\| \left\| v_{(i)} \right\|_2 \\ &\leq \sum_{i>1} 1 \cdot \left\| x_{(1)} \right\|_\infty \cdot 2\epsilon \cdot \left\| x_{(i)} \right\| \left\| v_{(i)} \right\|_2.\end{aligned}$$

Now let's use the blocking property to get

$$\left\| x_{(i)} \right\|_\infty \leq \frac{1}{k} \left\| x_{(i-1)} \right\|_1 \leq \frac{1}{\sqrt{k}} \left\| x_{(i-1)} \right\|_2 \leq \frac{2\epsilon}{\sqrt{k}} \sum_{i>1} \left\| x_{(i-1)} \right\|_2 \left\| v_{(i)} \right\|_2.$$

So we have

$$uBv \leq \frac{2\epsilon}{\sqrt{k}} \sum_{i>1} \frac{\left\| x_{(i-1)} \right\|_2^2 + \left\| v_{(i)} \right\|_2^2}{2} \leq \frac{2\epsilon}{\sqrt{k}}.$$

Now we'll use this to handle $B$. We won't use Hansen-Wright, but a similar Hoeffding bound, which is basically another version of Chernoff:

**Theorem** (Hoeffding)**.** *Let $\sigma \in \{\pm 1\}^n$ be a random sign vector and $x \in \mathbb{R}^n$. Then*

$$\Pr(\left|\sigma^T x\right| > \lambda) \leq 2^{-\Omega(\lambda^2 / \|x\|_2^2)}.$$

We wanted to use this to get that $\sigma_{(i)}^T B \sigma_{(-1)}$ is large with probability $2^{-\Omega(k)}$, but apparently there was a mistake which will be clarified over e-mail.

Let $\Pi = (\Pi_{(1)}, \ldots, \Pi_{(n/k)})$. For $i \neq j$, $\left\| \Pi_{(i)}^T \Pi_{(j)} \right\| \leq \epsilon$. Let $x_{(i)}, x_{(j)}$ be unit vectors. Let $x$ be the matrix with these as the $i$th and $j$th rows. Then we can use the standard trick with bounds on $\|\Pi x\|_2^2$, $\left\| \Pi_{(i)} x_{(i)} \right\|_2^2$ and $\left\| \Pi_{(j)} x_{(j)} \right\|_2^2$ to get a bound on $x_{(i)}^T \Pi_{(i)}^T \Pi_{(j)} x_{(j)} = \pm 2\epsilon$. So we've successfully bounded $B$.

For $C$, which we'll take to be an $n \times n$ matrix with the diagonal blocks and first row and column zeroed out. Then

$$\begin{aligned}\|C\|_F^2 &= \sum_{i,j>1, i\neq j} \left\| D_{x_{(i)}} \Pi_{(i)}^T \Pi_{(j)} D_{x_{(j)}} \right\|_F^2 \\ &\leq \sum_{i,j>1,i\neq j} \left( \|D_{x_{(i)}}\| \left\| \Pi_{(i)}^T \Pi_{(j)} \right\| \|D_{x_{(j)}}\|_F \right)^2 \\ &= \sum_{i,j>1,i\neq j} (\|x_{(i)}\|_\infty 2\epsilon \|x_{(j)}\|_2)^2 \\ \leq \frac{2\epsilon}{k} \sum_{i,j>1} \|x_{(i-1)}\|_2^2 \|x_{(j)}\|_2^2 &\leq \frac{2\epsilon}{k},\end{aligned}$$

where we used that the Frobenius norm satisfies $\|AB\|_F \leq \|A\| \|B\|_F$ early in that. Then with a board of computation that I don't want to copy down, $\|C\| \leq \frac{2\epsilon}{k}$. We've been bounding these norms to use Hanson-Wright. Setting $\lambda = \epsilon$, we get

$$\Pr[\left|\sigma^T C\sigma - \mathbb{E}[\sigma^T C\sigma]\right| > \epsilon] \leq 2^{-\Omega(k)}.$$

Putting ti all together,

$$\sigma^X \sigma = \sigma^T A\sigma + 2\sigma_{(1)}^T B\sigma_{(-1)} + \sigma^T C\sigma = (1 \pm \epsilon) + (\pm 2\epsilon) + (\pm\epsilon) = 1 \pm 4\epsilon$$

with probability $1 - 2^{-\Omega(k)}$. □

Let's look back on this proof in general. We analyzed the diagonal with the restricted isometry property. To bound the rest of the first row and column, we used Hoeffding's bound, and for the rest, we used Hanson-Wright. Question: Did we use the full power of the RIP? We used it twice, to bound $A$, and that the off-diagonal blocks have norm at most $2\epsilon$. Did we really need the full restricted isometry property? No, we only used the subspaces corresponding to the blocks, a small set of subspaces. Where did we use randomness? We used it to bound the off-diagonal entries close to 0. We could have used the $(\epsilon O(n/k+\sqrt{n}), 1)$-JL property.

We finished early today, at about 12:40.

# Thursday, October 31, 2013

We're talking about compressed sensing. Before, we recovered $\tilde{x}$ from $\Pi x$ using $\ell_1$-minimization. Unfortunately, solving a linear program is slow ($n^{O(1)}$ time). Today, we'll do iterative recovery instead of $\ell_1$-minimization. There are many iterative algorithms, like Orthogonal Matching Pursuit (OMP), rOMP, StOMP, CoSamp, and we'll see Iterative Hard Thresholding (IHT).

Blumensath and Davies analyzed Iterative Hard Threshold; while they might not have invented it, they applied it to compressed sensing.

Our setup is that $y = \Pi x + e$, where $e$ is the error (not the number). We'll assume it's some vector of bounded norm, and our error guarantee will be in terms of $e$.

The algorithm initializes $x^{[0]} = 0$, and for $i = 0$ to $t$, assigns $x^{[i+1]} \leftarrow H_k(x^{[i]} + \Pi^T(y - \Pi x^{[i]}))$. At the end, output $x^{[t+1]}$.

**Definition.** $H_k$ is called the "hard threshold operator" which keeps the $k$ largest entries of the vector in magnitude, and zeroes out the rest.

If $H_k$ wasn't there, then this would be a gradient descent step for least squares. We then just zero out the unimportant coordinates.

**Theorem.** *Suppose* $\Pi$ *satisfies* $3k$*-RIP with error* $\epsilon_k < \frac{1}{4\sqrt{2}}$*. Then if* $\tilde{x} = x^{[t+1]}$*, we have*

$$\|x - \tilde{x}\|_2 \lesssim 2^{-t} \|H_k(x)\|_2 + \|e\|_2 + \|x - H_k(x)\|_2 + \|x - H_k(x)\|_1 / \sqrt{k}.$$

*So after* $t = \Theta(\log(\left\|x^k\right\|_2 / \epsilon))$*, the error will be* $\lesssim \|x - H_k(x)\|_2 + \frac{1}{\sqrt{k}} \|x - H_k(x)\|_1 + \|e\|_2$.

This is a little weird, since we haven't seen the $\|x - H_k(x)\|_2$ term in the error before. But we claim that this implies a bound in terms of $\|x - H_k(x)\|_1 / \sqrt{k}$. Why? You can use sparsity $2k$ and then a shelling argument to get $\|x - H_{2k}(x)\|_2 \le \frac{1}{\sqrt{k}} \|x - H_k(x)\|_1$.

*Proof of Theorem.* First we will prove when $x = x^k$, i.e. $x$ is $k$-sparse. For notation, let $y = \Pi x^k + e$, $r^{[i]} = x^k - x^{[i]}$, $a^{[i+1]} = x^{[i]} + \Pi^T(y - \Pi x^{[i]})$, so $x^{[i+1]} = H_k(a^{[i+1]})$. Define $\Gamma_k^*$ to be the support of $x^k$, so $|\Gamma_k^*| \le k$, and $\Gamma^{i+1} = \text{supp}(x^{[i+1]})$, so $\left|\Gamma^{i+1}\right| \le k$ as well. Finally, let $B^{i+1} = \Gamma_k^* \cup \Gamma^{i+1}$.

Now

$$\begin{aligned}
\left\|x^k - x^{[t+1]}\right\|_2 = \left\|x^k_{B^{i+1}}\right\| &\le \left\|x^k_{B^{t+1}} - a^{[t+1]}_{B^{t+1}}\right\|_2 + \left\|a^{[t+1]}_{B^{t+1}} - x^{[t+1]}_{B^{t+1}}\right\|_2 \\
&\le 2\left\|x^k_{B^{t+1}} - a^{[t+1]}_{B^{t+1}}\right\|_2 = 2\left\|x^k_{B^{t+1}} - x^{[t]}_{B^{t+1}} - \Pi^T(y - \Pi x^{[t]})\right\|_2 \\
&= 2\left\|r^{[t]}_{B^{t+1}} - (\Pi^T \Pi r^{[t]})_{B^{t+1}} - (\Pi^T e)_{B^{t+1}}\right\|_2 \\
&= 2\left\|r^{[t]}_{B^{t+1}} - \Pi^T_{B^{t+1}} \Pi r^{[t]} - \Pi^T_{B^{t+1}} e\right\|_2 \\
&= 2\left\|r^{[t]}_{B^{t+1}} - \Pi^T_{B^{t+1}} \Pi (r^{[t]}_{B^{t+1}} + r^{[t]}_{B^t \setminus B^{t+1}} - \Pi^T_{B^{t+1}} e\right\|_2 \\
&\le 2\left\|(I - \Pi^T_{B^{t+1}}(\Pi^T_{B^{t+1}})^T) r^{[t]}_{B^{t+1}}\right\|_2 + 2\left\|\Pi^T_{B^{t+1}}(\Pi^T_{B^T \setminus B^{t+1}})^T r_{B^t \setminus B^{t+1}}\right\|_2 + 2\left\|\Pi^T_{B^{t+1}} e\right\|_2 \\
&\le 2\left\|I - \Pi^T_{B^{t+1}}(\Pi^T_{B^{t+1}})^T\right\| \left\|r^{[t]}_{B^{t+1}}\right\|_2 + 2\left\|\Pi^T_{B^{t+1}}(\Pi^T_{B^t \setminus B^{t+1}})^T\right\| \left\|r^{[t]}_{B^t \setminus B^{t+1}}\right\|_2 + 2\left\|\Pi^T_{B^{t+1}}\right\|_2 \|e\|_2 \\
&\le 2\epsilon_{2k}\left\|r^{[t]}_{B^{t+1}}\right\|_2 + 2\sqrt{1 + \epsilon_{2k}} \|e\|_2 + 2\epsilon_{3k}\left\|r^{[t]}_{B^t \setminus B^{t+1}}\right\|_2 \\
&\le 2\epsilon_{3k}(\left\|r^{[t]}_{B^{t+1}}\right\|_2 + \left\|r^{[t]}_{B^t \setminus B^{t+1}}\right\|_2) + 2\sqrt{1 + \epsilon_{3k}} \|e\|_2 \\
&\le 2\sqrt{2}\epsilon_{3k}\left\|r^{[t]}\right\|_2 + 2\sqrt{1 + \epsilon_{3k}} \|e\|_2 .
\end{aligned}$$

So set $\epsilon_{3k} < \frac{1}{2} \cdot \frac{1}{2\sqrt{2}} = \frac{1}{4\sqrt{2}}$, and it will be $\frac{1}{2}\left\|r^{[t]}\right\| + 3\|e\|_2$.

The caveat is that we assumed $x$ was $k$-sparse. Earlier, we said that $y = \Pi x + e = \Pi x^k + \tilde{e}$, where $\tilde{e} = e + \Pi(x - x^k)$. We get an error of $\|\tilde{e}\|_2 \le \left\|\Pi(x - x^k)\right\|_2 + \|e\|_2$, and then we use shelling:

$$\left\|\Pi(x - x^k)\right\|_2 \le \left\|\sum_{r=0}^{k} \Pi(x - x^k)_{S^r}\right\|_2 \le \sum_{r=1}^{n/k} \left\|\Pi(x - x^k)_{S^r}\right\|_2 \le (1 + \epsilon_k) \sum_{r=1}^{n/k} \left\|(x - x^k)_{S_r}\right\|_2 .$$

The conclusion is that we get $\|\tilde{e}\|_2 \le \|e\|_2 + \left\|x - x^k\right\|_2 + \frac{1}{\sqrt{k}} \left\|x - x^k\right\|_1$. That's the entire analysis. □

Algorithmically, what do you have to do at every iteration of IHT? You have to multiply by $\Pi$ and $\Pi^T$, and be able to multiply matrices times vectors quickly. You can do this for, say, sampling rows of the Fourier transform.

Now we're going use a variant on RIP matrices.

**Definition.** $\Pi$ satisfies $(\epsilon, k)$-RIP$_1$ if $\forall$ $k$-sparse $x$, $\|\Pi x\|_1 = (1 \pm \epsilon) \|x\|_1$.

We will construct $\text{RIP}_1$ matrices and show how to use them for $k$-sparse recovery, getting $\|x - \tilde{x}\|_1 \leq C \left\|x - x^k\right\|_1$. Problem 2 on PSet 7 will show that the old guarantee was strictly stronger, so this is a weaker guarantee, but the advantage is the following: We can get away with very good sparsity in $\Pi$. Also, we can achieve $C = 1 + \delta$ for arbitrarily small $\delta > 0$.

How will we get these $\text{RIP}_1$ matrices. For the first time, we're not going to use random sign matrices!

**Definition.** A $d$-left-regular bipartite graph $G = (U, V, E)$ is an *expander* is called a $(k, \epsilon)$-expander if for every $S \subset U$ with $|S| \leq k$, the neighborhood $|\Gamma(S)| \geq (1 - \epsilon) d |S|$.

Notice that the maximum size of the neighborhood is $d|S|$, so this is almost achieving the maximum.

*Claim.* There exist $d$-regular $(k, \epsilon)$ expander graphs with $|U| = n$, $|V| = m = O(\frac{1}{\epsilon^2} k \log(n/k))$ and $d = O(\frac{1}{\epsilon} \log(n/k))$.

The proof is randomized and you show that it happens with positive probability. Explicitly, you can achieve $d = O((\frac{1}{\epsilon} \log k \log n)^{1+1/\alpha})$ and $m = O(d^2 k^{1+\alpha})$ (Guruswami, Umans, Vadhan).

How do you get a matrix from such a graph? It's just $\frac{1}{d}$ times the bipartite adjacency matrix of $G$, an $m \times n$ matrix. This way, every column has $d$ nonzero entries and sum 1. Call this matrix $\Pi_G$.

*Claim.* If $G$ is a $d$-regular $(k, \epsilon)$-expander, then $\Pi_G$ is $(2\epsilon, k)$-$\text{RIP}_1$.

**Definition.** $\Pi$ satisfies the *$C$-restricted nullspace property* if $\forall \eta \in \ker(\Pi)$ and for all $S$ of size $k$, $\|\eta\|_1 \leq C \|\eta_{\bar{S}}\|_1$.

In other words, not too much of the $\ell_1$ mass of $\eta$ is contained in any set of size $k$.

*Claim.* If $\Pi$ satisfies $(3\epsilon, k)$-$\text{RIP}_1$ then $\Pi$ satsifies restricted nullspace of order $2k$ with $C = 1 + \sqrt{2}(1 - \epsilon^2)$.

Here's the punchline.

**Theorem.** *If $\Pi$ satisfies $C$-restricted nullspace of order $2k$, and $\tilde{x}$ is the result of $\ell_1$-minimization, then $\|x - \tilde{x}\|_1 \leq \frac{2C}{2-C} \left\|x - x^k\right\|_1$.*

We'll see next time how to prove these things and how iterative algorithms can get close to $1 + \epsilon$ in these bounds.

# Tuesday, November 5, 2013

Here's a general overview of upcoming classes:

- Today: Finish $\ell_1/\ell_1$ recovery. Introduce matrix completion.
- Thursday: Finish matrix completion.
- Next week: External-memory algorithms.
- Afterward: MapReduce/Hadoop.

Today, we'll finish up $\ell_1/\ell_1$ minimization. We finished last class establishing that there were these things called expanders, and we could use their adjacency matrix, scaled by the degree, to get the $\text{RIP}_1$ property. Now we'll see an iterative algorithm.

The best known iterative algorithm is by Indyk and Ruzic in FOCS '08, known as EMP (Expander Matching Pursuit). We won't be looking at it, though, covering a more recent algorithm of Berinde, Indyk and Ruzic called Sparse Matching Pursuit or SMP.

Let's recall the setup. $G = (U, V, E)$ is a bipartite expander with left degree $d$, $|U| = n$, $|V| = m$, and for all subsets $S \subseteq |U|$ expands if $|S| \leq s =: 2k$, where expands means $|\Gamma(S)| \geq (1 - \epsilon) d |S|$.

Recall that $\ell_1/\ell_1$ says we recover $\tilde{x}$ such that $\|x - \tilde{x}\|_1 \leq C \left\|x - x_{tail(k)}\right\|_1$. For EMP, $C = 1 + \epsilon$, but for SMP, $C = O(1)$, which is why SMP is worse.

For us, $\Pi \in \{0,1\}^{m \times n}$ is the bipartite adjacency matrix of $G$. Last time, we stated that $\frac{1}{d}\Pi$ satisfies RIP. We're going to analyze SMP without using this fact at all; you may or may not be able to use RIP as a black box. The paper used expander properties directly.

It's an iterative algorithm for estimating $b = \Pi x + e$.

1. Initialize $j = 0$, $x^j = 0$.
2. Repeat $T$ times the following: (a) $j = j + 1$; (b) $c = b - \Pi x^{j-1}$; (c) $u^* = u^*(c)$, $(u^*)_i$ is the median of $c_{\Gamma(i)}$; (d) $u^j = H_{2k}(u^*)$; (e) $x^j = x^{j-1} + u^j$; (f) $x^j = H_k(x^j)$.

The picture to keep in mind is the CountMedian sketch, which consisted of a $z \times w$ table $(C_{ij})$. We had $z$ hash functions $h_1, \dots, h_z : [n] \to [w]$ and added $x_i$ to $C_{h_r(i),r}$ for $r \in [z]$. Then we looked at the median over $r$ of these counters, and we didn't have a $\ell_1/\ell_1$ guarantee, but a $\ell_\infty/\ell_1$ guarantee: $\|x - \tilde{x}\|_\infty \leq O(1/w) \left\|x - x^{w/2}\right\|_1$ as long as $z = \Omega(\log n)$.

Instead of thinking of the hash functions as being specified by a pairwise independent family, we should think of them as determined by the edges of the expander. It doesn't have a grid structure like this, where each item is hashed to one bucket in each row, but it'll work similarly.

Back to SMP. We will assume $x = x^k$ (i.e. $x$ is $k$-sparse). That is, if not, $\Pi x + e = \Pi x^k + \Pi(x - x^k) + e$, and we'll let $\tilde{e} = e + \Pi(x - x^k)$. Then $\|\tilde{e}\|_1 \leq \|e\|_1 + \left\|\Pi(x - x^k)\right\|_1 \leq \|e\|_1 + d\left\|x - x^k\right\|_1$, which is a bound we're fine with.

*Claim.* After $T$ iterations, $\left\|x - x^T\right\|_1 \lesssim \frac{1}{2^T}\|x\|_1 + \frac{1}{d}\|\tilde{e}\|_1$.

That's what we'll show. Now let's look at the algorithm again. We start off with a guess of $x$ (we don't know anything, so it's just 0). We then estimate $c$ as $\Pi(x - x^{j-1}) + \tilde{e}$, and we want to estimate the residual $x - x^{j-1}$ itself, but we don't think that's going to work so we threshold by the first $2k$ terms. Then we add this error and threshold so it's $k$-sparse again. That's the big idea.

*Proof of Claim.* We will show that after step (d), $\left\|u^j - (x - x^{j-1})\right\|_1 \leq \frac{1}{4}\left\|x - x^{j-1}\right\|_1 + \frac{C}{d}\|\tilde{e}\|_1$. Then after (e), $\left\|x - x^j\right\|_1$ will also be less than this (since they're equal). So after (f), we'll use this to show $\left\|x - x^j\right\|_1 \leq \frac{\left\|x - x^{j-1}\right\|_1}{2} + 2\frac{C}{d}\|\tilde{e}\|_1$. So we can see how this implies the lemma by a geometric sequence and series. First let's show that second claim.

**Lemma.** *Let $x$ be $k$-sparse, $x'$ arbitrary. Then $\|H_k(x') - x\|_1 \leq 2\|x' - x\|_1$.*

It should be clear how this would be enough, so let's prove it.

*Proof.* Let $S = \text{supp}(x)$ and $T = \text{supp}(H_k(x')) \subseteq S$. Then $|S - T| = |T - S|$ so $\left\|x'_{S-T}\right\|_1 \leq \left\|x'_{T-S}\right\|_1$. We have

$$\|H_k(x') - x\|_1 = \|x'_T - x\|_1 = \|x_{S-T}\|_1 + \left\|x'_{T-S}\right\|_1 + \|(x - x')_{S\cap T}\|_1 \leq \|(x - x')_{S\cap T}\|_1 + 2\left\|x'_{T-S}\right\|_1 \leq 2\|x - x'\|_1,$$

as desired. □

Finally, we need to prove the claim about after (d). We'll need a couple lemmas for this. In what follows, let $u = x - x^{j-1}$, and note that $u$ is $2k$-sparse with terms $|u_1| \geq \cdots \geq |u_{2k}|$, so WOLOG $\text{supp}(u) = S = [2k]$.

**Lemma** (A)**.** $\|(u^* - u)_S\|_1 \lesssim \epsilon\|u\|_1 + \|\tilde{e}\|_1/d$ *(the expander parameter).*

**Lemma** (B)**.** *Let $B \subseteq \bar{S}$ with $|B| \leq 2k$. Then $\|u^*_B - u_B\|_1 \leq \|u^*_B\|_1 \lesssim \epsilon\|u\|_1 + \|\tilde{e}\|_1/d$.*

For the proof of the claim using these lemmas, we want to bound $\|H_{2k}(u^*) - u\|_1$ since we just thresholded $u^*$. Let $T = \text{supp}(H_{2k}(u^*))$, so of size $\leq 2k$. Then

$$\begin{aligned}
\|H_{2k}(u^*) - u\|_1 &= \|u^*_T - u\|_1 = \|(u^* - u)_{S\cap T}\|_1 + \left\|u^*_{T-S}\right\|_1 + \|u_{S-T}\|_1 \\
&\leq \|(u^* - u)_{S\cap T}\|_1 + \left\|u^*_{T-S}\right\|_1 + \left\|u^*_{S-T}\right\|_1 + \|(u - u^*)_{S-T}\|_1 \\
&\leq \|(u^* - u)_S\|_1 + 2\left\|u^*_{T-S}\right\|_1.
\end{aligned}$$

Apply Lemma A to the first term and Lemma B to the second term, with $B = T - S$. Take $\epsilon$ in your expander small enough that these terms are less than what we need, and that finishes the job. □

*Proofs of Lemmas A and B.* Now we finally use the expander property. We had $m$ counters $b_1, \ldots, b_m$, and each will get some $u_i$ which are hashed there. For each bucket, we'll select the $u_i$ of largest magnitude in that bucket. Collect all of these into $v \in \mathbb{R}^m$. Then write $\Pi u = v + v''$ and $c = v + v'' + \tilde{e}$. We claim that $\|v''\|_1 \leq 2\epsilon d \|u\|_1$ (see proof of Theorem 1 of [Berinde, Gilbert, Indyk, Karloff, Strauss]).

It would be nice to have the medians line up, but we don't have that. Fortunately, it's easy to show that the median of $a + b$ is at most the sum of the upper quartiles of $a$ and $b$. For another easy claim, suppose $a_1, \ldots, a_s, b_1, \ldots, b_t > 0$, where $\{a_1, \ldots, a_s\} \subseteq \{b_1, \ldots, b_t\}$, and define $c(i) = |\{j : a_j \geq b_i\}|$. Then $\forall i$, $c(i) \leq \alpha i$ implies $\|a\|_1 \leq \alpha \|b\|_1$. Both of these claims are easy and you can look them up in the paper if you want.

Now for the proof of Lemma A, $u_i^*$ is the median of $v_{\Gamma(i)} + v''_{\Gamma(i)}$. Then

$$\begin{aligned}
\|(u - u^*)_S\|_1 &= \sum_{i \in S} \left| \text{median}(v_{\Gamma(i)} + v'_{\Gamma(i)}) - u_i \right| \\
&= \sum_{i \in S} \left| \text{median}(v_{\Gamma(i)} - u_i^d + v'_{\Gamma(i)}) \right| \\
&\leq \sum_{i \in S} \text{median}(\left| v_{\Gamma(i)} - u_i^d \right| + \left| v'_{\Gamma(i)} \right|) \\
&\leq \sum_{i \in S} \text{quant}_{1/4}(|w'|) + \sum_{i \in S} \text{quant}_{1/4}(\left| v'_{\Gamma(i)} \right|).
\end{aligned}$$

Now let $P$ be any set of $s \leq 2k$ coordinates. We're finally going to be using the expander property! We claim that $\sum_{i \in P} \text{quant}_{1/4}(\left| v'_{\Gamma(i)} \right|) \lesssim \frac{\|v'\|_1}{/} d$. We proved this claim using the expaner property and the previous unproved claim, but I was running behind and didn't copy this down. All of this bounds the second term by $\|v'\|_1 / d$.

The proofs of Lemma B and bounding the first term are similar, but we won't see it. Hopefully you can see where expansion fit into the proof, though. □

In the last 7 minutes, we'll preview what matrix completion is, a new topic.

The idea is that we have a matrix $M$ which is $n_1 \times n_2$. Entry $M_{ij}$ corresponds to a user's rating of product $j$. (Netflix is the classic example.) The idea is that there is some true rating matrix out there, but every user doesn't rate every product. So only some entries of $M$ are revealed to us, and we have to fill out the rest of the matrix. This is impossible as stated, so you need some assumptions.

Formally, you have $M_{ij}$ for $(i, j) \in \Omega$ with $|\Omega| \ll n_1 n_2$. The kinds of assumptions we'll make are:

1. $M$ is (approximately) low-rank.
2. The column- and row-spaces of $M$ are incoherent. (Any standard basis vector has small $\ell_2$-norm.)
3. Random entries of $M$ are revealed. Yeah, this is a little weird.

So it's not just a case that you have a complete block of the matrix; it's a random sampling. What we'll show next time is that you can recover the matrix with $|\Omega| \leq (n_1 + n_2) r \, \text{polylog}(n_1 n_2)$(incoherence terms).

The natural recovery method would be to minimize the rank of $X$ such that for all $(i, j) \in \Omega$, $X_{ij} = M_{ij}$. Unfortunately, this is NP-hard. On the PSet, you'll show a semidefinite program to minimize the trace norm of $X$, the sum of the singular values of $X$, such that $X_{ij} = M_{ij}$, and SDP's can be solved in polynomial time.

# Thursday, November 7, 2013

Today: Matrix completion.

We have some rank-$r$ matrix $M \in \mathbb{R}^{n_1 \times n_2}$ such that $(M_{ij})_{(i,j)\in\Omega}$ are revealed, for $|\Omega| = m \ll n_1 n_2$.

Our model is that we sample $m$ times $(i,j)$ uniformly at random and insert them into $\Omega$. We'll recover $M$ by solving this problem (NNM):

$$\text{minimize } \|X\|_* \text{ such that } \forall (i,j) \in \Omega, X_{ij} = M_{ij}.$$

Under some conditions, $M$ will be the unique solution to the above program. The history:

- [Recht, Razel, Parrilo '10] was the first to give some rigorous guarantees for NNM
- [Candès, Recht '09] applied this to matrix completion.
- Improvements in [Candès, Tao '09], [Keshavan, Montanori, Oh '09]
- Today's lecture will come from [Recht, '11].

Let's set up our notation. Suppose $n_1 \le n_2$.

**Definition.** Let the SVD of $M$ be $M = U\Sigma V^T$. Define $\mu(U) = \frac{n_1}{r} \max \|P_U e_i\|^2$, known as the incoherence in $U$, and $\mu(V) = \frac{n_2}{r} \max_i \|P_V e_i\|^2$. Then $\mu_0 = \max\{\mu(U), \mu(V)\}$ and $\mu_1 = \left\|UV^T\right\|_\infty$.

We'll show that $m \gtrsim \max\{\mu_1^2, \mu_0\} \cdot n_2 r \log^2(n_2)$. Note that $1 \le \mu_0 \le n_2/r$.

Let's say some things about dual norms and the trace norm.

**Definition.** $\langle A, B\rangle = \operatorname{tr}(A^* B) = \sum_{i,j} A_{ij} B_{ij}$.

*Claim.* $\|A\|_* = \sup_{\|B\|\le 1} \langle A, B\rangle$.

**Lemma.** $\|A\|_* = \min_{X,Y : A = XY^*} \|X\|_F \|Y\|_F = \min_{X,Y:A=XY^*} \frac{1}{2}(\|X\|_F^2 + \|Y\|_F^2)$.

*Proof of Lemma.* Call these terms (1), (2), (3). Then (2) $\le$ (3) by AM-GM, since $xy \le \frac{1}{2}(x^2+y^2)$. We have (3) $\le$ (1) as we can take $X = Y^* = A^{1/2}$. Finally, for (1) $\le$ (2), let $X, Y$ be such that $A = XY^*$. Then

$$\begin{aligned}
\|A\|_* = \|XY^*\|_* &\le \sup_{\substack{\{a_i\},\{b_i\} \\ \text{o.n. bases}}} \sum_i \langle XY^* a_i, b_i\rangle \\
&= \sup \sum_i \|Y^* a_i\| \, \|X^* b_i\| \\
&\le \sup \left(\sum_i \|Y^* a_i\|^2\right)^{1/2} \left(\sum_i \|X^* b_i\|^2\right)^{1/2} = \|X\|_F \|Y\|_F,
\end{aligned}$$

so (1) $\le$ (2) and the lemma is proved. □

*Proof of Claim.* We'll prove both directions.

1. We'll show that $\|A\|_* \le \sup_{\|B\|=1} \langle A, B\rangle$. To do so, just take $B = \sum_i u_i v_i^*$, from the SVD of $A$.
2. We'll show that $\|A\|_* \ge \langle A, B\rangle$ for all $B$ with $\|B\| = 1$. Write $A = XY^*$ such that $\|A\|_* = \|X\|_F \|Y\|_F$. Write $B = \sum_i \sigma_i a_i b_i$ for all $i$, $\sigma_i \le 1$. Then

$$\langle A, B\rangle = \left\langle XY^*, \sum_i \sigma_i a_i b_i \right\rangle = \sum_i \sigma_i \langle Y^* a_i, X^* b_i\rangle \le \sum_i |\langle Y^* a_i, X^* b_i\rangle| = \|X\|_F \|Y\|_F = \|A\|_*. \quad \square$$

Define $T$ and $T^\perp$ in terms of $U, V$ from the SVD. For notation, define the projections $P_{T^\perp}(Z) = (I - P_U)Z(I - P_V)$ and $P_T(Z) = Z - P_T^\perp(Z)$. Finally, let $R_\Omega(Z)$ only keep entries in $\Omega$, multiplied by multiplicity in $\Omega$ (since we allowed us to choose entries multiple times).

Now we claim that a bunch of good things all happen with high probability (i.e. $1 - 1/\operatorname{poly}(n_2)$).

1. $\frac{n_1 n_2}{m}\left\|P_T R_\Omega P_T - \frac{m}{n_1 n_2} P_T\right\| \lesssim \sqrt{\frac{\mu_0 r(n_1+n_2)\log(n_2)}{m}} \ll \frac{1}{2}$ with our choice of $m$. This is a deviation inequality, since $\frac{m}{n_1 n_2} P_T$ is the expectation of $P_T R_\Omega P_T$. The way we'll prove them is something like the Chernoff bound for matrices.
2. $\left\|\left(\frac{n_1 n_2}{m} R_\Omega - I\right) Z\right\| \lesssim \sqrt{\frac{n_1 n_2^2 \log(n_1+n_2)}{m}} \|Z\|_\infty$. This is also a concentration bound.
3. If $Z \in T$, then $\left\|\frac{n_1 n_2}{m} P_T R_\Omega(Z) - Z\right\|_\infty \lesssim \sqrt{\mu_0 r n_2 \log(n_2)} m \|Z\|_\infty$.
4. $\|R_\Omega\| \lesssim \log(n_2)$. This is just balls and bins. In fact, balls and bins are at most log/loglog, but we won't complicate things.
5. There exists $y \in \text{range}(R_\Omega)$ such that (a) $\|P_T(Y) - UV^*\|_F \le \sqrt{\frac{r}{2n_2}}$, and (b) $\|P_{T^\perp}(Y)\| < \frac{1}{2}$.

Now we'll use these things to prove that trace-norm minimization is what we want. Technically, we're proving that $\operatorname{argmin}_{X: R_\Omega(X) = R_\Omega(M)} \|X\|_*$ is unique and equal to $M$. Alternatively stated, for $Z \in \ker(R_\Omega)$, and we want to show that $\|M + Z\|_* \ge \|M\|_*$.

First, we'll argue that $\|P_T(Z)\|_F$ cannot be big. We have

$$0 = \|R_\Omega(Z)\|_F \ge \|R_\Omega(P_T(Z))\|_F - \|R_\Omega(P_{T^\perp}(Z))\|_F .$$

Meanwhile,

$$\begin{aligned}
\|R_\Omega(P_T(Z))\|_F^2 &= \langle R_\Omega P_T Z, R_\Omega P_T Z\rangle \ge \langle P_T Z, R_\Omega P_T Z\rangle = \langle Z, P_T R_\Omega P_T Z\rangle = \langle P_T Z, P_T R_\Omega P_T Z\rangle \\
&= \left\langle P_T Z, \frac{m}{n_1 n_2} P_T Z\right\rangle - \left\langle P_T Z, \left(P_T R_\Omega P_T - \frac{m}{n_1 n_2}\right) P_T Z\right\rangle \\
&\ge \frac{m}{n_1 n_2}\|P_T Z\|_F^2 - \left\|P_T R_\Omega P_T - \frac{m}{n_1 n_2} P_T\right\| \|P_T Z\|_F^2 \ge \frac{m}{2 n_1 n_2}\|P_T Z\|_F^2 ,
\end{aligned}$$

where we used statement 1 from our list in the last line. Stepping back, we've lower bounded $\|R_\Omega(P_T(Z))\|_F^2$, and now we need to upper bound $\|R_\Omega(P_{T^\perp}(Z))\|_F^2$:

$$\|R_\Omega(P_{T^\perp}(Z))\|_F^2 \le \|R_\Omega\|^2 \cdot \|P_{T^\perp}(Z)\|_F^2 \lesssim \log^2(n_2)\|P_{T^\perp}(Z)\|_F^2 ,$$

by statement 4. Therefore, we've proved that

$$\|P_T(Z)\|_F \le \sqrt{\frac{n_1 n_2 \log^2(n_2)}{m}}\|P_{T^\perp}(Z)\|_F \lesssim \sqrt{\frac{n_2}{r}}\|P_{T^\perp}(Z)\|_F .$$

Now pick $U_\perp$ and $V_\perp$ such that $\langle U_\perp V_\perp^*, P_T^\perp(Z)\rangle = \|P_{T^\perp}(Z)\|_*$ (which exists by the duality fact we proved earlier) and such that $[U, U_\perp]$ and $[V, V_\perp]$ are orthogonal matrices. Now

$$\|M + Z\|_* \ge \langle UV^* + U_\perp V_\perp^*, M + Z\rangle = \|M\|_* + \langle UV^* + U_\perp V_\perp^*, Z\rangle ,$$

and we just need to show that this error is positive. Subtracting $Y \in \text{range}(\Omega)$, this is also

$$\begin{aligned}
\langle UV^* + U_\perp V_\perp^* - Y, Z\rangle &= \|UV^* - P_T(Y)\| P_T(Z) + \langle U_\perp V_\perp^* - P_{T^\perp}(Y), P_{T^\perp}(Z)\rangle \\
&= -\|UV^* - P_T(Y)\|_F \|P_T(Z)\|_F + \|P_{T^\perp}(Z)\|_* - \|P_{T^\perp}(Y)\| P_{T^\perp}(Z)_F .
\end{aligned}$$

The big positive term here is $\|P_{T^\perp}(Z)\|_* \geq \|P_{T^\perp}(Z)\|_F$, and we have to show the negative stuff doesn't cancel it out. By statement 5b, the last term only cuts it in half. The other term is bounded by $-\sqrt{\frac{r}{2n_2}}\|P_T(Z)\|_F$, and that's where the inequality we got before comes in.

We only had 7 minutes left, so we just stated how one might prove the 5 claims. Statements 1-3 were deviation bounds, using things like Chernoff. Behind 1 and 2, technically, we need

**Theorem** (Non-Commutative Bernstein Inequality)**.** *Suppose $X_1, \ldots, X_n$ are random matrices of the same dimensions such that (1) $\mathbb{E}X_i = 0$ for all $i$, (2) $\|X_i\| \leq M$ a.s., (3) $\sigma_i^2 = \max\{\|\mathbb{E}X_iX_i^*\|, \|\mathbb{E}X_i^*X_i\|\}$ (like the variance). Then*

$$\Pr\left(\left\|\sum_{i=1}^N X_i\right\| > \lambda\right) \lesssim (n_1+n_2)\max\left\{e^{-C\lambda^2/\sum_i \sigma_i^2}, e^{-C\lambda/M}\right\}.$$

We won't prove these, but you can think of $R_\Omega$ as a sum of a bunch of random matrices with expectation $\frac{1}{n_1n_2}I$. You need statements 2 and 3 to get statement 5, which we'll prove next time.

# Tuesday, November 12, 2013

Today, we're starting a new topic: The External Memory Model, also called the Disk Access Model (DAM).

All our data is stored on disk, but to access it, we have to move it to a smaller memory (RAM) of size $M$. Everything is stored in blocks of size $B$ which we can bring into the RAM at a cost of 1. All operations in RAM are free.

Some easy examples:

*Example.* 1. Scanning an array: $N/B + 1$ I/O's.

2. Multiplying two $N \times N$ matrices. One thing you can do is to split them up into $\sqrt{B} \times \sqrt{B}$ blocks and store those blocks and multiply them in the standard way. This takes $O((N/\sqrt{B})^3) = O(N^3/B^{3/2})$ time. But you can do better: Use blocks of size $\sqrt{M/2}$, using the whole memory. This takes $O\left(\frac{M}{B}\left(\frac{N}{\sqrt{M}}\right)^3\right) = O\left(\frac{N^3}{B\sqrt{M}}\right)$ I/O's.

3. Linked list. Not everyone has seen them, so there's a head with a pointer to the next element, and so on. This supports insertion, deletion, and traversing $k$ elements. How do we implement this with good performance in the I/O model?

   We want to take at most (possibly amortized) $k/B$ I/O's. So we'll do the same thing as above, except that each node is a block, with the invariant that each block (except the last one) is at least half full.

   What is the cost to maintain this invariant? Insertion is easy: Add the new element to the last block, and split this block in half if it's overfull now. For deletions, we can delete it unless the block is exactly half-full. In that case, look at the next node and move an item back to the previous block, unless the other one is exactly half-full. In that case, merge them.

4. Binary search. We have $N$ items, sorted. Someone tells us to find an item in the list. If we laid them out sequentially, it would take $O(\log(N/B))$ steps to get down to a window of size $B$, so from there on, we just need one or two blocks.

   In the DAM model, though, reading a block gives us information about the relative positioning within the entire block, rather than having to do $\log B$ comparisons. So we should find our item in $\log N/\log B = \log_B N$ I/O's.

   We're going to store our array in a B-tree, which is an example of an $(a,b)$-tree. In an $(a,b)$-tree, every internal node has between $a$ and $b$ children, and leaves have between $a$ and $b$ elements. A common

example is a $(2,4)$-tree, that corresponds completely to red-black trees. We'll explain this in terms of $(2,4)$-trees, which are also sometimes known as $(2,3,4)$ trees since you can have 2, 3, or 4 children.

We illustrated how this works. When a block gets $b+1$ children, we split that node into two halves and promote a central element to split it.

Usually when you're dealing with binary search trees, you care a lot about balance. But here every leaf is at the same depth since the tree only grows upwards. As each leaf has between $B/2$ and $B$ leaves, the tree is of height $\log_{B/2}(2N/B) = O(\log_B N)$.

These are used everywhere. There will be a homework problem about how you can beat B-trees. And there's a startup based on beating B-trees in another way, which we'll see in another class.

The problem with B-trees is that insertions are pretty slow. If we just had a log of changes, queries would take $O(N/B)$ time, but amortized insertion would be $O(1/B)$ time, which is a lot better than $O(\log N/\log B)$.

Here's the improvement: The Buffer Repository tree (2000). It's an augmented $(2,4)$-tree with a size $B$ repository at each node and blocks of size $B$ at each node. Insertion goes into the root's buffer. When it fills up, you can flush it to its children and recursively flush them.

How does this do? Queries are $O(\log \frac{N}{B})$ I/O's, which is worse, but we'll fix that in the Problem Set. Amortized insertion is $O\left(\frac{1}{B}\log \frac{N}{B}\right)$, and worst-case is $O(\log \frac{N}{B})$.

5. Sorting. We'll do a variant of merge-sort that's multi-way. Instead of splitting into $B$ piles, we split into $M/B$ piles. Recursively sort these, then merge them. The cost of merging is $N/B + M/B$, accounting for potentially small blocks. Then $T(N) \le O(N/B) + (M/B)T(N/(M/B))$. Solving this, we get $T(N) \le O\left(\frac{N}{B}\log_{M/B}(N/B)\right)$.

   We claim that this is optimal in the comparison model. Suppose you had $N/B$ blocks of size $B$ in sorted order. The number of permutations yielding this is $N!/(B!)^{N/B}$. Each comparison costs one bit of information, and so the number of bits is $N\log N - \frac{N}{B}\cdot B\log B = N\log(N/B)$. Whenever we bring in a block, we learn at most the positioning of those $B$ items with respect to the $M$ in memory, or $\log\binom{M+B}{B} \approx B\log(M/B)$ bits. So in total, the number of I/O's we need is the number of bits we need divided by the number of bits per I/O, so $\Omega\left(\frac{N\log(N/B)}{B\log(M/B)}\right) = \Omega\left(\frac{N}{B}\log_{M/B}(N/B)\right)$ I/O's.

Next lecture, we'll talk about how to make performance guarantees even if you don't know $B$ or $M$. We might want to run code on different machines, or reduces tuning considerations, and these algorithms will be pretty cool.

# Thursday, November 14, 2013

Last time, we started external memory algorithms, covering linked lists, $B$ trees and $(a,b)$-trees. We were operating on the disk access model (DAM). Today, we're looking at the cache-oblivious model: The data structure is not allowed to know $M$ (the memory size) or $B$ (the block size).

Notice that the B-tree and $M/B$-merge sort don't work since we need to set them up. But there is one algorithm that was cache oblivious: Traversing a list/array of size $N$. It takes at most $\lceil N/B\rceil + 1$ I/O's.

In the disk access model, we had to tell it which blocks to evict and so on. In the cache-oblivious model, we assume that it takes care of that for us and does so optimally. That might sound problematic, but we'll cite a result showing us that it's not so problematic. But first, we need to talk about competitive ratios and online algorithms.

In an online algorithm, think of the input as a stream, and at the end of the day, you compare the cost of the decisions you made and the decisions that an optimal algorithm would make if it could see into the future.

That is, OPT sees the whole input in advance and makes the best decisions based on that. The algorithm must make decisions at each step without knowing the future. If you think about the paging problem, where you have memory organized into blocks, if you knew what the entire sequence of blocks to be read was, then you could optimally choose which blocks to keep and which to evict back to memory.

The canonical example of online algorithms is the ski rental problem. You can rent skis for \$1 or buy them for \$10, but you don't know how many times you'll ski. Each day, your friend either says, (A) Let's go skiing! or (B) I'm done skiing for the rest of my life.

An all-knowing algorithm would simply count how many times we should ski, and would buy skis if you're going to ski 10 times or more. What can you do otherwise? Well, you can rent for the first 10 days, then buy. This way, you're only off by a factor of 2 from optimal. This means that your algorithm is 2-competitive.

Back to external memory. What does OPT do?

*Claim.* The omniscient algorithm would evict the block that will be used the farthest in the future.

Commonly used algorithms that don't know the future include:

- First In, First Out (FIFO)
- Least Recently Used (LRU)

These evict what it sounds like. The nice thing about these algorithms is the following theorem:

**Theorem** (Sleator, Tarjan '85)**.** *(1) LRU and FIFO are $M$-competitive, but (2) LRU and FIFO are 2-competitive against OPT for a memory of size $M/2$.*

The second statement is the more important one. So we're not directly comparing your performance with an optimal algorithm with the same size cache, but half as big. This says something nice for our algorithms. Even if you don't have any power over how the blocks are evicted, you can do fine. This gives us some kind of regularity condition: As long as $T(N, M, B) = O(T(N, M/2, B))$ when analyzing our CO algorithm, then [ST] implies no asymptotic loss in performance. So we're not going to worry about how to evict from memory anymore.

Let's list things we already know about cache-oblivious algorithms, which were introduced by Frigo, Leiserson, Prokop, Ramachandra at FOCS '99:

- Array traversal takes $\leq 1 + \lceil N/B \rceil$ I/O's.
- Array reversal: $O(N/B + 1)$. Start at both ends and swap elements as you move along.
- Matrix multiplication. We can't block it as we did last time into $\sqrt{M} \times \sqrt{M}$ because we don't know how big $M$ is. We'll do something that doesn't affect the time at all but helps this situation. Break up our two matrices $A$ and $B$ into

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}.$$

  We'll do the matrix multiplication recursively, and store the output recursively. So we'll store $A$ as $(A_{11}|A_{12}|A_{21}|A_{22})$, and store each of those in the same fashion recursively. We do this so that we get locality, which means that at some level, we can multiply by simply bringing the whole thing into memory.

  Let's analyze the number of I/O's. We get a recurrence: $T(n) \leq 8T(n/2) + O(n^2/B + 1)$. The base case will be $T(\sqrt{M}) = O(M/B)$ since we can bring in the chunks together. Solving this gives $T(N) = O\left(\frac{n^2}{B} + \frac{n^3}{M\sqrt{B}}\right)$.
- Linked list. This idea is in a survey by Demaine called "Cache Oblivious Algorithms and Data Structures." We're shooting for $O(1)$ for insertion and deletion. We already know it takes $O(1 + k/B)$ to

traverse $k$ elements (amortized). Here's what we'll do: For insertion, we just append an element to the end of the array. For deletion, we'll just mark its array location as empty. This will possibly leave a lot of elements blank. Whenever we traverse it, we'll clean up the blanks. Here's how we'll do that: When we traverse some $k$ elements, we collect those elements and move them to one big block at the end.

Suppose we have $n$ items in our data structure so far. Then every $n/2$ updates, we'll completely rebuild the data structure. This avoids the problem of ending up with too much space. This guarantees that we'll never use too much memory, and the cost is $n/B$ but there were $n/2$ updates, so amortized, this only adds $1/B$ to each update cost.

So let's analyze the traversals. Suppose we touched $r$ runs of contiguous elements. So the number of I/O's is $O(r + k/b)$. Actually, we should clarify that we maintain pointers (in both directions) to where the next element actually is. The runs might be in different blocks, which is how we get $r$. But these runs only got created because of at least $r$ deletions. We can charge 1 to each of those updates to make that update pay for this cost. So in an amortized sense, the $O(r)$ can be taken care of.

To clarify, by saying the amortized cost is $C$ means that any sequence of $t$ operations has total cost at most $tC$. Think of it like a bank: When we do a deletion, we'll take one coin to make it, then we also store one coin in our bank, and we cash those out when we need to jump over those jumps.

- Static B-trees. It seems odd that you'd be able to do a B-tree without knowing $B$, but you can. What static means is that someone doesn't want to insert or delete, just query. It uses something known as the Van Ende Boas layout. We'll imagine our giant tree having height $\log_2 n$ and divide it in half. Each of the $\sqrt{n}$ nodes in this level will be the root of a tree in the bottom half. (If $\log_2 n$ is not even, round so that the top is smaller.) So the top tree is $T_0$ and the trees in the bottom are $T_1, T_2, \ldots, T_{\sqrt{n}}$. On disk, we'll lay out the trees $T_0, T_1, \ldots, T_{\sqrt{n}}$ each recursively.

  Again, we get locality. When we get a query, we do the same thing as with a regular binary search tree. How do we analyze this? Look at the first scale of recursion when trees have at most $B$ elements. Then at this level, we'll have at least $\sqrt{B}$ nodes in our tree. Because of the locality, reading any of those trees takes 1 I/O. So we can traverse from root to leaf in $O(\log N/\log\sqrt{B}) = O(\log_B N)$ I/O's, as we got earlier.

  On PSet 9 Problem 2, you'll see how to combine this data structure with another one to get queries to still be $\log_B N$ and updates a little worse. There are further tricks you can apply.

  This is a common thing you'll do with cache-oblivious algorithms; you use some common data structure but just store it in memory in a special way that preserves locality.

- Sorting. This was another thing we did last time where we needed $M$ and $B$ things, because it was a $M/B$-mergesort. We'll do something called Lazy Funnelsort. Funnelsort was done by FLPR '99, simplified to the lazy version later by Brodal and Fagerberg. It'll be similarly recursive, but it's even funkier.

  Suppose we had an object called a "$K$-funnel" which had the following properties analogous to mergesort:

  - Uses $O(K^2)$ space.
  - Can merge $K$ sorted lists of total size $K^3$ in $O\left(\frac{K^3}{B}\log_{M/B}\frac{K^3}{B} + K\right)$ I/O's.

  How will we sort? We'll divide our memory into $\sqrt[3]{n}$ blocks of size $n^{2/3}$. We'll recursively sort each block and merge it using a $k$-funnel with $K = n^{1/3}$.

  We'll first make the "tall cache assumption." You assume that $M = \Omega(B^2)$. This can be relaxed to $M = \Omega(B^{1+\gamma})$ for arbitrarily small $\gamma > 0$. Unfortunately, you can't do without this; it's known that

this is required to get the right number of I/O's, which is $O\left(\frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)$ I/O's. This was proved by Brodal and Fagerberg in STOC '03.

First, what is the $k$-funnel? We'll again build it recursively, and use the static B-tree VEB layout. We build a giant binary tree with $\sqrt{K}$ leaves where we'll feed in the $K$ sorted lists, and put a $K^3$ size funnel on the top. Since we're doing this recursively, we'll also put buffers in the edges of the tree halfway down the tree. These buffers will have size $K^{3/2}$, and there are $\sqrt{K}$ of them, so $K^2$ space at this level. So technically the topmost buffer will be read out onto whereever we need the output of the algorithm, but the ones in the middle are read out in some intermediate location.

Space used by funnels is recursive, with the space being $S(K) = (1+\sqrt{K})S(\sqrt{K}) + O(K^2)$ and if you solve this, you get $S(K) = O(K^2)$. Now, every edge has some buffer on it. The root node asks its two edges to empty their buffers to be merged into its output buffer. If those buffers are empty, recursively as the subtrees to fill their buffers.

The major theorem says that Lazy Funnelsort as described, with the tall cache assumption takes $O\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ I/O's. We'll just sketch the proof. For the first property, we already wrote the recurrence. For the second, look at the coarsest scale of recursion where we have $J$ funnels with $J \ll \sqrt{M}$. The rest of the analysis is involved, but if you want to see all the details, look in the survey by Demaine (the original papers are too complicated).

Pset 9 is out, due Tuesday morning. It's the last PSet.

## Tuesday, November 19, 2013

Today: (1) Finish cache-oblivious, (2) MapReduce/Hadoop.

We'll start with a cache-oblivious version of the Buffered Repository Tree, known as the "Cache-Oblivious Lookahead Array" (COLA) introduced by Bender, Farach-Colton, Fineman, Fogel, and Nelson in SPAA '07.

First recall the performance of the buffered repository tree: queries take $O(\log N)$, and updates take $O(\frac{1}{B}\log N)$. Problem 1 on PSet 9 (due this morning) was similar with a branching factor of $\Theta(B^\epsilon)$ and gave us an example of $O(\frac{1}{\epsilon}\log_B n)$ query and $O(\frac{1}{\epsilon B^{1-\epsilon}}\log_B n)$ updates. That's due to Brodal and Fagerberg '03.

Today, we'll achieve $O(\log n)$ queries cache-obliviously. It's also possible to get the $B^\epsilon$ tradeoff cache-obliviously (Brodal, Demaine, Fineman, Iacono, possibly more, SODA some year).

So let's see what COLA does. Keep a list of arrays sorted of sizes $2^k$ laid out contiguously. We'll maintain that each array is either completely full or completely empty. Therefore, insertion will involve merging an inserted item and the longest prefix of full arrays into the next array. A query just conducts a binary search in each array separately.

Let's analyze this. Query takes $\log\frac{N}{B}$ I/O's per array, and there are $\log N - \log B$ arrays. Let's assume that $M/B \gtrsim \log N$, so we can merge $k$ items with $O(k/B)$ I/O's. Now let's amortize. The first $\log B$ arrays fit in $O(1)$ blocks. Each item participates in at most $\log N - \log B$ merges that cost I/O's, so we'll give each item $O(\log(N/B)/B)$ credits upon insertion. Spend $O(1/B)$ credits for each item.

There are a couple things to fix here. First, we needed to assume $M/B \gtrsim \log N$. There's a way to fix this by doubling the space. Instead of merging the whole array, merge them one by one and store the results in an auxiliary table of the same sizes.

Second, our query time is $\log^2(N/B)$, not logarithmic. To fix this, we'll use a concept known as *fractional cascading*. The idea is to sprinkle "lookahead pointers" all over the place. For instance, every 32nd item in array $i$ will also appear in array $i-1$ with pointers between them. Also, every 32nd item in array $i-1$ will not be an item (depicted as a dot) but have two pointers to the closest lookahead spots to the left and right.

We drew an example to understand it. The repeated elements and the "dots" will be offset so they never fall on each other. In all, this will use only a constant factor of extra space. Then in scanning an array, the pointers tell us that we only need to look at 32 points in each vertex, or $O(\log(N/B))$ query time. As an exercise, you can explain how to insert in only $O(1/B \log N/B)$ time. The paper also gives a way to

get $O(\log(N/B))$ worst-case insertion. Basically, you make two copies of the structure and a little bit of background merge work.

Some version of COLA was the basis of a company, Tokutek. Michael Bender had a quote about why people care about fast inserts. There are some scenarios where you log everything that happens, but don't query it all that often. That's the kind of scenario where you'd want to use. If you're cache-aware, $B^\epsilon$ is probably better.

How about dynamic CO B-trees? The original CO B-tree is due to Bender, Demaine, Farach-Colton in FOCS '00. We will get $O(\log_B N)$ query worst-case and $O(\log_B N)$ update amortized; the paper also gets this. In PSet 9, Problem 2, it's also possible to make that amortized bound worst-case. This is a BCR '02 approach called exponential trees.

It was developed to do faster sorting by Andersson and Thorup in various works. Speaking of which, sorting can do better than $O(n \log n)$ because all of the classic arguments are based on the comparison model, but real computers can do bitwise xor and so on. You can actually get down to $O(n \log \log n)$ deterministically and $O(n\sqrt{\log \log n})$ deterministically. There's a whole area of algorithms that uses the RAM model to get faster algorithms. The Van Embe Boas layout is motivated by faster algorithms than binary heaps. You can often get loglog performance as opposed to log using bit tricks. One surprising thing is that you can compute the index of the least significant bit in constant time.

Anyways, the idea is to have a tree of "fat nodes" of depth $\log \log N$ where each subtree at level $i$ has a total number of nodes $\approx 2^{2^i}$. (Leaves at depth 0 have just one node.) The exponential tree is like a 2,3,4-tree except the branching factor changes at each level. The first level will have branching $\Theta(\sqrt{N})$ and the next level has branching $\Theta(N^{1/4})$, and so on. Therefore, the height is about $\log \log N$. The volume of a root will then be about $2^{2^{l(v)}}$ where $l(v)$ is the level of $v$.

To do a query, you just find where it is in the root, then descend. Store each fat node using the static cache-oblivious B-tree with items at the leaves (the Van Emde Boas layout). If a node ends up with too many children, do the usual splitting, promoting an element to the previous layer. Except you can't do that because the parent is static, so what do you do? Just rebuild the entire parent from scratch. Split it in two only if it gets too big itself. You'd only do this if a child doubled in size, which requires the same cost as rebuilding.

Let's analyze this. Query takes $\sum_{i=0}^{\log \log n} \log_B(2^{2^i}) + \log \log N$. The sum will be dominated by its last term, $\log_B N$, so this is $O(\log_B N + \log \log N)$. Update takes $O(\log_B N + \log \log N)$ to find the item, plus you might have to split, but this will be $O(\log \log n)$ amortized cost.

We'd like to remove the $\log \log N$, and there's a trick. We'll require that a fat node and its entire subtree are stored contiguously in memory. If you see the details in the paper, the exact branching factor at each level $i$ is $\leq 2(2^{2^{i-1}} + 2^{2^{i-2}} + 2)$. Really it's an interval between half that and that. The total space we'll need is therefore $\leq \prod_{j=0}^{i} 2(2^{2^{j-1}} + 2^{2^{j-2}} + 2) = O(2^i 2^{2^i})$. So the whole tree takes $O(N \log N)$ space.

But because we stored it contiguously in memory, any subtree rooted at level $\log \log B$ has volume $\lesssim B$, so it fits into $O(1)$ blocks. This gives us query in $O(\log_B N + \log \log N - \log \log B) = O(\log_B N + \log \log_B N) = O(\log_B N)$. To fix the $N \log N$ space, instead build it over $\Theta(N/\log N)$ leaves, and make each leaf not a single item but an array of $\Theta(\log n)$ items that you rebuild each time there's an insertion.

By the way, the Array Insertion Problem from the problem set is also called the Packed Memory Problem (Jelani made that name up to make it less Googleable).

## MapReduce

This is a system developed at Google for massive parallel processing. Hadoop is an open-source variant made by Apache and Yahoo has its own version. Facebook apparently was runnign Hadoop on 21 petabytes of data 3 years. However, Jelani read some funny quotes from a blogpost (Dewitt, Stonebreaker on January 17, 2008) about how MapReduce is a "step backwards." However, the reality still is that many people are using it, so people want to model it and come up with provably good algorithms.

The big problem is that we have a bunch of machines, each with some limited memory. In all these algorithms, we'll assume no machine has enough memory to store the whole input. Items are key-value pairs

and are processed/created in 3 phases:

- Map. The machine gets a list of items $(k_1, v_1), \ldots, (k_n, v_n)$. Map takes in one item and spits out some number of key-value pairs.
- Shuffle: For each key, takes all the items with this key and puts them on the same machine.
- Reduce: Takes as input all the items with some key: $(k, (v_1, \ldots, v_t))$ and produces some new outputs based on this list.

The algorithms we'll study proceed in rounds, with each round consisting of all three of these steps. We want algorithms with small memory, few machines, few rounds to complete, and small total work (summed across all machines). We'll start looking at that next time.

# Thursday, November 21, 2013

I overslept today, so these notes start half an hour in. See the official scribe notes for this period of time.

Our goals with MapReduce are to use few machines ($n^{1-\epsilon} \ll n$, the size of the input), for each machine to have $\ll n$ memory (e.g. again $n^{1-\epsilon}$), the number of rounds $R$ small, and the total work small.

Let's see an example of an algorithm for MST (minimum spanning tree) in dense graphs. Suppose the number of edgse is $m = N^{1+c}$ and $N$ is the number of vertices, where $c$ is bounded away from 0. Here's the algorithm:

1. Pick $k = N^{c/2}$, and randomly partition vertices $V$ into equal-sized sets $V_1, \ldots, V_k$.
2. Let $G_{ij} = (V_i \cup V_j, E_{ij})$ be the induced graph on $V_i \cup V_j$.
3. Compute using the sequential algorithm $M_{ij}$, the minimum spanning forest on $G_{ij}$.
4. Send $H = \cup_{i,j} M_{ij}$ to a single reducer and output $M$, the minimum spanning tree of $H$.

The number of machines is $k^2 = N^c < N < m^{1-\epsilon}$, so few.

*Claim.* With high probability, for all $i, j$, $|E_{ij}| \leq \tilde{O}(N^{1+c/2})$.

*Proof.* We have $|E_{ij}| \leq \sum_{v \in V_i} \deg(v) + \sum_{v \in V_j} \deg(v)$, so we care about the degrees. Split up the vertices into $W_t = \{v \in V : 2^{t-1} \leq \deg(v) < 2^t\}$. For each $t = 1, \ldots, \log N$, separately, if $|W_t| \leq N^{c/2} \log N$, then it's okay if all of $W_t$ maps to $G_{ij}$.

Because no degree is bigger than $N$, $W_t$ contributes at most $N^{1+c/2} \log N$ to the degree sum. So in total, they contribute $\ll N^{1+c/2} (\log N)^2$.

Otherwise, suppose $|W_t| \geq N^{c/2} \log N$. Then the expected number of vertices from $W_t$ landing in $G_{ij}$ is $2 |W_t| / k = 2 \log N$. So by Chernoff, this will be close to expectation (within a constant factor) with probability $1/\operatorname{poly}(N)$. So we can union bound over $t = 1, \ldots, \log N$ and $i, j = 1, \ldots, k$, which gives us at most $N^c \log N$ things to union bound over, and that's fine. □

The downside to this algorithm is that the total memory accesses of the whole system can be up to $m^{2-2\epsilon}$, so the space required is almost quadratic. However, there's another algorithm that's better.

Another thing we can do with this is triangle counting / clustering coefficients. The input is an undirected graph and we want to count the number of triangles, i.e. $|\{u < v < w \in V : (u, v), (v, w), (u, w) \in E\}|$. Alternatively, you might want the clustering coefficient at a particular vertex: $cc(V) = \frac{|\{u, w \in \Gamma(v) : (u,w) \in E\}|}{\binom{\deg(v)}{2}}$.

First, what are the standard algorithms for these things? A simple algorithm is three for-loops. Just loop over $v$, then over neighbors $u$ and $w$, which gives you both clustering coefficients and triangles in time $O\left(v \in V d_v^2\right)$.

Slightly cleverer: Loop over $v, u \in \Gamma(v), w \in \Gamma(v)$ such that $\deg(w) \geq \deg(u) \geq \deg(v)$. The claim is that the running time is $O(m^{3/2})$. This isn't a tight analysis, but it's the best we could do just in terms of $m$.

*Proof of Claim.* Analyze $d_v < t$ and $d_v \geq t$ separately. Then $\sum_{v \in V, d_v \leq t} d_v^2 \leq t \sum d_v \leq mt$. For $d_v > t$, there will be at most $m/t$ such vertices, so the total time is $(m/t)^3$. Putting these together, we get $(m/t)^3 + mt$ and you should set $t = \sqrt{m}$. □

You can implement this in MapReduce. We claim that no reducer will get more than $O(\sqrt{m})$ items to process, that the total work will be at most $O(m^{3/2})$, and the number of rounds is 2. Here's the algorithm:

Map: The input is some edge $\langle (u, v); 1 \rangle$. Let $\rho$ be some parameter. Hash $h : V \to [\rho]$ and $i \leftarrow h(u)$, $j \leftarrow h(v)$. For $a \in [\rho]$, for $b \in [\rho]$ with $b > a$ and $c \in [\rho]$ with $c > b$, if $\{i, j\} \subset \{a, b, c\}$, emit $\langle (a, b, c); (u, v) \rangle$.

Reduce: Input $\langle (i, j, k); E_{ijk} \rangle$. Use any algorithm to enumerate the triangles in the set of edges $E_{ijk}$. For each triangle $(u, v, w)$, set $z \leftarrow 1$ then if $h(u) = h(v) = h(w)$, $z \leftarrow \binom{h(u)}{2} + h(u)(\rho - h(u) - 1) + \binom{\rho - h(u) - 1}{2}$. Otherwise, if $|\{h(u), h(v), h(w)\}| = 2$, then set $z = \rho - 2$. (These two formulas accounting for overcounting.) Output $\left\langle (u, v, w), \frac{1}{z} \right\rangle$. At the very end, there needs to be one more phase where everything is aggregated.

So what this is doing is partitioning $V$ into $V_1, \ldots, V_\rho$ and defining $V_{ijk} = V_i \cup V_j \cup V_k$. Define $E_{ijk}$ to be the edges induced on $V_{ijk}$, $G_{ijk}$ the corresponding graph, and each $V_{ijk}$ has expected size $3N/\rho$. So $|E_{ijk}| = O(m/\rho^2)$.

This is an example of a general approach of using hashing. Suppose we have frequency moments $\langle 1, x_1 \rangle, \ldots, \langle n, x_n \rangle$ where $x_i \in [N]$ and I want to compute $\sum_{i=1}^{N} (\#\text{ of occurences of } i \text{ as a value})^k =: F_k$.

The naive approach is to map $\langle i, x_i \rangle$ to $\langle x_i, 1 \rangle$ and the reducer outputting the number of $x_i$'s to the $k$th power, and finally aggregate. This is a problem because all of the $x_i$ could go to the same machine.

An even simpler example is that you might want to add up a bunch of numbers, and what you'd want to do is send chunks of them to different machiens and have them add those chunks up, then aggregate the totals. We'll do something similar here.

**Definition.** Say a function $f$ is MapReduce-parallelizable if there exist $g, h$ such that for all partitions $T_1, \ldots, T_k$ of $S$, $f(S) = h(g(T_1), \ldots, g(T_k))$ and we can communicate descriptions of $g, h$ efficiently (say, in $O(\log n)$ bits).

*Claim.* Given a universe $U$ of size $n$ and $S_1, \ldots, S_k \subseteq U$ (which may overlap), $k \leq n^{2-c\epsilon}$, $\sum_{i=1}^{k} |S_i| \leq n^{2-c'\epsilon}$, and $f_1, \ldots, f_k$ are MR parallelizable, then we can compute $f(S_1), \ldots, f(S_k)$ using $O(n^{1-\epsilon})$ reducers, each with $O(n^{1-\epsilon})$ memory.

*Proof Sketch.* We have $M = n^{1-\epsilon}$ reducers which we break up into $T = n^{1-2\epsilon}$ blocks of equal size $n^\epsilon$. Then for each $i = 1, \ldots, k$, hash $i$ to some $S_j$ with $j \in [t]$ and for each item in $S_i$, process it using a random reducer in that block. □

Now to compute frequency moment $F_k$, $U$ is just the set of input pairs, and for each $l \in [N]$, $S_l$ is the indices with $x_i = l$. Then compute $f(S_l) = |S_l|^k$ using $h(i_1, \ldots, i_m) = (\sum_{j=1}^{m} i_j)^k$ and $g(T_j) = |T_j|$. Then you do need to aggregate everything in the end.

# Tuesday, November 26, 2013

Today: More MapReduce, particularly "Solve and Sketch" [Andoni, Nikalov, Onak, Yaraslavtrev, '14?] Then we'll say a few words about $k$-means [Bahmani, Kumar, Moreley, Vassilvitskii, Vattani VLDB '12].

Solve-and-Sketch is a method to solve some approximate low-dimensional computational geometry problems. It might solve

1. Minimum Spanning Tree
2. Min-cost Bipartite Matching
3. Earthmover Distance (EMD): Given two sets of points $A, B$, each point has a pile of dirt (weight) on it. The total amount of dirt $A$ and on $B$ is the same. Moving $m$ mass of dirt from $x$ to $y$ costs $m \cdot d(x, y)$, and the EMD is the least cost to move dirt from $A$ around to get into configuration $B$. This is used

in vision to compare two grayscale images. After normalizing each one to have the same total amount of blackness. The set of points is the set of pixels and dirt is the darkness. EMD then says that local small modifications are cheap to change.

Recall that MapReduce only works on MST for sparse graphs. For all three of these, the points are in $\mathbb{R}^d$ and it's a complete graph on these points where edgeweights are Euclidean distance. (The algorithms aren't sensitive to the particular $\ell_p$ distance, though.)

We want to get $(1+\epsilon)$ times the optimal for any of these. The Solve-and-Sketch approach idea is to use the geometry to partition the data and send it to machines. This partition will be hierarchical: Starting with your initial set $T$, build a tree with branching factor $c$ to partition the data to singletons at height $L$, so we'll say that $T$ is at level $L$ and the leaves are at level 0.

It's like a quadtree, which most of the class hadn't heard of. Imagine you're in $\mathbb{R}^d$ and you partition the set of points by orthant (signs of the coordinates). Do that recursively for different origins until you get small enough boxes. For minimum spanning tree, we'll find the minimum spanning tree on each box and then combine them at each stage. So we'll commit to keeping some edges we pick at a smaller stage.

But we don't want to commit too early. We drew out an example where we'd be forced to make commitments that made our MST worse by a factor of almost 2. The way they fix this is that they use a randomly shifted and rotated quadtree instead, so these scenarios are rare. Then each node locally computes some partial solution on the set of points it has, compresses this output, and sends it upwards in the tree.

The basic operation is called a *unit step*. On an input of size $n_u$, this produces an output of size $p_u(n_u)$ and runs in time $t_u(n_u)$ and uses total space at most $s_u(n_u)$. This takes place at every internal node

**Theorem.** *Fix $s = (\log n)^{\Omega(d)}$. Suppose there is a unit step algorithm with $s_u(p_u(S)) \le s^{1/3}$ and $p_u(S) \le s^{1/3}$. Then for $c = \mathrm{poly}(s) \le s$, $L = O(d \log_s n)$. We can implement Solve-and-Sketch in MapReduce in $\mathrm{poly}(\log_s n)$ rounds. Each node's runtime is $st_u(s)\,\mathrm{polylog}(n)$.*

In general, we'll use one of these:

**Definition.** For $a \in (0,1)$ and $b > 0$, a *randomized hierarchical partition* $B$ is "$(a,b)$-distance preserving with approximation $\gamma > 1$" if for $\Delta_l = \gamma \cdot a^{L-l}\mathrm{diam}(S)$ and for every partition $P = (P_0, \dots, P_L)$ of the support of the distribution, (i) $\forall l \ge 0$, $\mathrm{diam}(P_l) \le \Delta_l$ and $\forall x, y \in T$, $\Pr_p(C_l(x) \ne C_l(y)) \le b\frac{\ell_2(x,y)}{\Delta_l}$.

Then we have a theorem out of Section 5 of this paper:

**Theorem** (Aiger, Kaplan, Sharir SODA '13)**.** *A randomly shirted/rotated "c-ary quadtree is $(c^{-1/d}, O(d))$-distance preserving with approximation $O(1)$.*

What will our unit step be? Consider a given internal node. This is responsible for some set of points $C$ in its subtree. The input is a partition of $V(C) \subseteq C$ into $\{Q_1, \dots, Q_k\}$ into connected components based on edges that were decided on lower levels.

While $\exists i, j \in [k]$ such that $\ell_2(Q_i, Q_j) \le \epsilon\Delta_l$:

1. Pick $u \in Q_i$ and $v \in Q_j$ such that $\ell_2(u,v)$ is minimal.
2. Output the edge $(u,v)$ as an edge in the final MST
3. Merge $Q_i, Q_j$.

After this is done, output $V' \subseteq V(C)$ that is an $\epsilon^2\Delta_l$-cover of $V$ and an induced partition $Q'$ of $V$'. Recall that if $(X,d)$ is a metric space then $X'$ is a "$\epsilon$-cover" of $X$ iff $\forall x \in X$, $\exists x' \in X'$ with $d(x,x') \le \epsilon$.

Define $L_{cut}(u,v)$ to be the highest level of the hierarchy where $C_l(u) \ne C_l(v)$. What does each node do? It gets $V'_1, \dots, V'_c$ as well as partitions $Q'_1, \dots, Q'_C$. It sets $V = \cup_{i=1}^k V'_i$ and $Q = \cup Q'_i$. Run the unit step on $V, Q$. At leaves, $V$ is just the set of points and $Q$ is a partition into singletons.

We've finally finished defining the algorithm, so next we should show that it gets a $1+\epsilon$-approximation to OPT. Then show that there are weights $w(u,v)$ such that this algorithms outputs MST wrt $w$ (as opposed to $\ell_2$).

**Lemma.** *There is a way to define the weights in terms of the partition tree such that* $\forall u, v$, $\ell_2(u,v) \leq w(u,v) \leq (1+\epsilon)\ell_2(u,v) + \alpha\Delta_{l_{cut}}(u,v)$.

At the end of the day, we also want to bound that extra term. Let's take expectations: $\mathbb{E}w(u,v) = (1+O(\epsilon))l(u,v) + \alpha\mathbb{E}\Delta_{l_{cut}(u,v)}$. Then

$$\mathbb{E}\Delta_{l_{cut}(u,v)} = \sum_{l=0}^{L} \Pr_p(l_{cut}(u,v) = t)\Delta_l = \sum_l \Pr_p(C_l(u) \neq C_l(v), C_{l+1}(u) = C_{l+1}(v))\Delta_l \leq \sum_l \Pr_p(C_l(u) \neq C_l(v)) \leq bL\ell_2(u,v).$$

So take $\alpha = O(\epsilon)$ and the "$\epsilon$" here to be $\frac{\epsilon}{bL}$.

There are many other good MapReduce papers: [Goodrich, Sitchinava, Zhang ISAAC '11] gave MR-efficient sorting (amongst other) algorithms using $O(\log_M n)$ rounds of MapReduce and $O(N \log_M N)$ total communication. Here, $M$ is the I/O bound on mappers and reducers. This sorting is used as a subroutine here.

Finally, let's recall $k$-means clustering from two PSet problems. Given $x_1, \ldots, x_n$ the problem is to find $k$ centers $c_1, \ldots, c_k$ such that $\sum_{i=1}^n (x_i, \{c_1, \ldots, c_k\})^2$ is minimized.

A popular method is Lloyd's algorithm: Start with some initial set of centers $c_1, \ldots, c_k$. Iteratively cluster points to their closest $c_i$, then move the $c_i$'s to be centroids of each cluster. This has no guarantees in general, but there are heuristics for initailizing it appropriately.

Here's one with provable guarantees: $k$-means++, introduced by [Arthur, Vassilvitskii SODA '07]. This starts with a random data point to the set of centers $C$, and while $|C| < k$, sample a data point $x$ with probability proportional to $\ell_2^2(x, C)$. Then when you have $k$ points, do Lloyd's algorithm.

**Theorem.** *The initial* $C$ *already provides* $O(\log k)OPT$ *with good probability.*

The problem with implementing this in MapReduce is that it's inherently sequential. [BKM+, VLDB '12] give a different algorithm $k$-means$\|$. They also start with a random data point, then set $\psi$ to be the cost of $C$ (the sum of the distances). For $O(\log \psi)$ iterations, sample each point independently with probability $\Theta(k)\ell_2^2(x, C)/\psi$, and add all of these to $C$. This gives you $O(k \log \psi)$ points, and then run $k$-means++ or whatever on those.