CS 15-851: Algorithms for Big Data Spring 2025

Lecture 7 — 2/27/2024

Prof. David Woodruff *Scribe: Aksara Bayyapu*

Theorem 1 (Theorem)**.** *With high probability (w.h.p.),*

$$\|Ax\|_1 \leq \|RAx\|_1 \leq d \log d \, \|Ax\|_1.$$

Recall that

$$\|RAx\|_1 = \|Ax\|_1 \cdot \frac{1}{d \log d} \sum_j |Z_j|,$$

where each $|Z_j|$ follows a half-Cauchy distribution. Using Chernoff bounds, we have analyzed the sum

$$\sum_j |Z_j|$$

and concluded that

$$\sum_j |Z_j| = \Omega(d \log d)$$

with probability at least $1 - e^{-d \log d}$. This immediately implies that

$$\|RAx\|_1 \geq \|Ax\|_1,$$

as desired.

Next, we aim to show the upper bound:

$$\|RAx\|_1 \leq d \log d \, \|Ax\|_1.$$

Note that the $|Z_j|$ are heavy-tailed; indeed, each $|Z_j|$ has a cumulative distribution function (c.d.f.) that is asymptotically

$$F(t) \sim 1 - \Theta\left(\frac{1}{t}\right),$$

which implies

$$\mathbb{P}\left[|Z_j| \geq t\right] \sim \frac{1}{t},$$

ignoring constant factors. We cannot use an argument similar to the lower-bound as it will not yield a strong enough result to apply the net argument.

Our strategy for proving the upper bound begins by considering that

$$\|RA_{*i}\|_1 = \|A_{*i1}\|_1 \cdot \frac{1}{d \log d} \sum_j |Z_{i,j}|,$$

where each $Z_{i,j}$ is a half-Cauchy random variable. We introduce the event

$$E_{ij} := \{\, |Z_{i,j}| < d/3 \,\}$$

and note that, due to the behavior of the half-Cauchy distributions cumulative distribution function, the probability of E_{ij} is approximately $1 - \Theta(1/d)$.

Next, define the truncated variable Z'_{ij} by capping $Z_{i,j}$ at $d/3$. Under this truncation, one can show that

$$\mathbb{E}[|Z_{i,j}| \mid E_{ij}] = \mathbb{E}[|Z'_{ij}|] = O(\log d).$$

To see this, observe that the expected value is given by

$$\int_0^{d^3} \frac{2z}{\pi(1 + z^2 \cdot Pr[Z_{ij} \text{ in } [0, d^3]])}\, dz,$$

which is comparable to

$$\int_1^{d^3} \frac{1}{z}\, dz,$$

thereby yielding an order of $O(\log d)$.

Now, let E be the event that every E_{ij} occurs for all indices i and j. By applying a union bound over the $d^2 \log d$ pairs, we obtain

$$\mathbb{P}[E] \geq 1 - \frac{\log d}{d}.$$

Wen bounding $\mathbb{E}[|Z'_{ij}| \mid E_{ij}]$, we use the law of total expectation. Although the columns of RA are independent, the entries in the rows of RA are not necessarily independent. We start by writing the conditional expectation as

$$\mathbb{E}[Z'_{i,j} \mid E_{i,j}] = \mathbb{E}[Z'_{i,j} \mid E_{i,j}, E] \cdot \mathbb{P}(E \mid E_{i,j}) + \mathbb{E}[Z'_{i,j} \mid E_{i,j}, \neg E] \cdot \mathbb{P}(\neg E \mid E_{i,j}).$$

Since all terms are nonnegative, it follows that

$$\mathbb{E}[Z'_{i,j} \mid E_{i,j}] \geq \mathbb{E}[Z'_{i,j} \mid E_{i,j}, E] \cdot \mathbb{P}(E \mid E_{i,j}).$$

Noting that conditioning further on E does not change the expectation when $E_{i,j}$ already occurs, we can write

$$\mathbb{E}[Z'_{i,j} \mid E_{i,j}] \geq \mathbb{E}[Z'_{i,j} \mid E] \cdot \mathbb{P}(E \mid E_{i,j}).$$

Next, by applying Bayes' rule we have

$$\mathbb{P}(E \mid E_{i,j}) = \frac{\mathbb{P}(E_{i,j} \mid E) \cdot \mathbb{P}(E)}{\mathbb{P}(E_{i,j})}.$$

Under our assumptions, one can show that

$$\mathbb{P}(E \mid E_{i,j}) \geq 1 - \frac{\log d}{d}.$$

Given that we have already established $\mathbb{E}[Z'_{i,j} \mid E] = O(\log d)$, we obtain

$$\mathbb{E}[Z'_{i,j} \mid E_{i,j}] \geq O(\log d) \cdot \left(1 - \frac{\log d}{d}\right).$$

Rearranging the terms (i.e., moving the $\log d$ factor appropriately) leads us to conclude that

$$\mathbb{E}[Z'_{i,j} \mid E] = O(\log d).$$

Assume that the event E holds. Under this assumption, consider the expression

$$\|RA_{*i}\|_1 = \|A_{*i1}\|_1 \cdot \frac{1}{d \log d} \sum_j |Z_{i,j}|.$$

By taking the sum over the index i, we deduce that

$$\sum_i \|RA_{*i}\|_1 = O(\log d) \sum_i \|A_{*i1}\|_1.$$

Using Markovâs inequality, we can ensure that this bound holds with a constant probability.

To refine our previous polynomial-in-d bound, we use an Auerbach basis. Such a basis, denoted $\{A_{*1}, A_{*2}, \ldots, A_{*d}\}$, satisfies the following properties:

- For any vector x, we have $\|x\|_\infty \leq \|Ax\|_1$.
- The sum of the column norms is exactly d, i.e., $\sum_i \|A_{*i}\|_1 = d$.

Now, for any x, we start with

$$\|RAx\|_1 \leq \sum_i \|RA_{*i}x_i\|_1,$$

where the triangle inequality is applied. Since each scalar x_i satisfies $|x_i| \leq \|x\|_\infty$, we can factor out $\|x\|_\infty$:

$$\|RAx\|_1 \leq \|x\|_\infty \sum_i \|RA_i^*\|_1.$$

Using the bound on $\sum_i \|RA_i^*\|_1$ and the property $\|x\|_\infty \leq \|Ax\|_1$, we obtain

$$\|RAx\|_1 \leq O(d \log d) \, \|Ax\|_1.$$

Thus, we conclude that for all x,

$$\|RAx\|_1 \leq O(d \log d) \, \|Ax\|_1.$$

It remains to show that

$$\|Ax\|_1 \leq \|RAx\|_1 \quad \text{for all } x.$$

Up to this point, we have established this inequality for any fixed x with probability at least $1 - e^{-d \log d}$. To extend this result uniformly over all x, we employ a γ-net argument. In particular, for any x there exists a vector y satisfying

$$\|Ax - y\|_1 \leq \gamma,$$

with $\gamma = \frac{1}{d^3 \log d}$.

Using the triangle inequality, we write

$$\|RAx\|_1 \geq \|Ry\|_1 - \|R(Ax - y)\|_1.$$

The second term is controlled by our previous bound, so that

$$\|R(Ax - y)\|_1 \leq O(d \log d) \, \|Ax - y\|_1 \leq O(d \log d) \, \gamma = O\left(\frac{1}{d^2}\right).$$

Thus, we have

$$\|RAx\|_1 \geq \|Ry\|_1 - O\Big(\frac{1}{d^2}\Big).$$

In addition, by the high-probability result established for fixed vectors, it follows that

$$\|Ry\|_1 \geq \|y\|_1 - O\Big(\frac{1}{d^2}\Big).$$

Combining these estimates yields

$$\|RAx\|_1 \geq \|y\|_1 - O\Big(\frac{1}{d^2}\Big).$$

Recall that the well-conditioned (Auerbach) basis of A satisfies the properties:

- $\|Ax\|_1 \geq \|x\|_\infty$ for all x,
- and since $\|x\|_1 = 1$, we have $\|x\|_\infty \geq \frac{1}{d}$.

Therefore, $\|Ax\|_1 \geq \frac{1}{d}$ and the error term $O\Big(\frac{1}{d^2}\Big)$ is negligible compared to $\|Ax\|_1$. Moreover, since

$$\|y\|_1 \geq \|Ax\|_1 - \gamma - O\Big(\frac{1}{d^2}\Big),$$

for sufficiently large d we conclude that

$$\|y\|_1 \geq \frac{1}{2}\|Ax\|_1.$$

Thus, we obtain

$$\|RAx\|_1 \geq \frac{1}{2}\|Ax\|_1,$$

which shows that the desired inequality $\|Ax\|_1 \leq \|RAx\|_1$ holds uniformly over all x.

ℓ_1-Regression Recap

- **Goal:** Solve an ℓ_1-regression problem efficiently, i.e., find x that minimizes $\|Ax - b\|_1$.
- **Algorithm Outline:**
  1. Compute a polynomial-in-d approximation of the solution.
  2. Compute a well-conditioned basis for A.
  3. Sample rows from both the well-conditioned basis and from the residual of the initial approximation.
  4. Solve the ℓ_1-regression problem on the sampled rows to obtain the final vector x.

Sketching with a Cauchy Matrix: The main computational bottleneck in this method is the multiplication $A \cdot R$, where R is typically a matrix of i.i.d. Cauchy random variables. A naive implementation can cost $O(\text{nnz}(A) \cdot d)$ or more, which becomes expensive for large or sparse matrices.

Faster Sketching via CountSketch + Cauchy Diagonal: The key idea is instead of using a full i.i.d. Cauchy matrix R, we replace it with the product of

$$R \;=\; D \cdot S,$$

where

1. S is a *CountSketch* matrix, which is a sparse embedding reducing the dimensionality, and
2. D is a diagonal matrix whose diagonal entries are i.i.d. Cauchy random variables.

CountSketch can be applied in near input-sparsity time, i.e., $O(\text{nnz}(A))$. Multiplying by a diagonal matrix D is only an elementwise scaling and is therefore efficient. Overall, this yields an embedding step in $O(\text{nnz}(A) + \text{poly}(d/\varepsilon))$ time rather than an expensive dense multiplication. The error guarantees differ slightly from using a fully dense Cauchy matrix, but are still sufficient for ℓ_1-regression. Typically, one obtains with high probability:

$$(1-\delta)\,\|Ax\|_1 \;\;\leq\;\; \|RAx\|_1 \;\;\leq\;\; (1+\delta)\,\|Ax\|_1,$$

for all x, where δ is small or depends on ε.

By using the CountSketch + diagonal Cauchy embedding R, the time to compute RA is dominated by $\text{nnz}(A)$ plus a lower-order polynomial term in d/ε. Thus, the overall time for the ℓ_1-regression algorithm is:

$$O\big(\text{nnz}(A)\big) \;+\; \text{poly}\Big(\frac{d}{\varepsilon}\Big).$$

This improves upon methods requiring dense multiplication with a full Cauchy matrix, which can be much more expensive when A is large or sparse.

A Fun Fact about Cauchy Random Variables

Let $R_1, R_2, \ldots, R_n$ be i.i.d. random variables. If each R_i has a finite mean and variance, then by the Central Limit Theorem (CLT) the average

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

converges in distribution to a normal random variable with mean 0 and variance σ^2/n, where σ^2 is the common variance of the R_i.

Now consider the case where each R_i is a standard Cauchy random variable, which has neither a well-defined mean nor variance. In fact, if we take

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

where each R_i follows a standard Cauchy distribution, the resulting random variable is still a standard Cauchy. This is a striking example of the CLT *failing* because the Cauchy distribution does not satisfy the finite variance condition required for the theorem to hold.

Streaming Model

While sketching has primarily been used to improve the time complexity of algorithms, it also plays a crucial role in reducing memory usage. One important framework for this is the **streaming model**.

Turnstile Streaming Model

- We start with an n-dimensional vector $x \in \mathbb{R}^n$ that is initially set to all zeros.
- The vector x undergoes a long sequence of updates. In each update, a coordinate x_i is modified as:
$$x_i \leftarrow x_i + \delta_i,$$
where δ_i is either $+1$ or -1.
- After all updates, the final vector x lies in a bounded set, i.e.,
$$x \in \{-M, -M+1, \ldots, M\}^n,$$
for some bound $M \leq \text{poly}(n)$.
- The goal is to output an approximation to a function $f(x)$ (such as a norm or other statistic) with high probability.
- The main objective in this model is to use as little memory (measured in bits) as possible.

Is the final vector x (after all updates) the zero vector? A straightforward solution would be to maintain the vector x throughout the stream, which requires $O(n)$ space. However, we can achieve this with much less space by using sketching techniques.

Using CountSketch

Assume we use a CountSketch matrix S with $O(1/\varepsilon^2)$ rows. With this sketch, the squared ℓ_2 norm of the sketched vector Sx satisfies
$$\|Sx\|_2^2 \in (1 \pm \varepsilon)\|x\|_2^2$$
with probability at least $9/10$. In particular, if $x = 0^n$, then $\|x\|_2 = 0$ and by the subspace embedding property, Sx will also be 0 with high probability.

For the purpose of testing whether $x = 0^n$, we set $\varepsilon = \frac{1}{2}$. With this setting, we only need to store $O(1)$ entries of Sx, each of which can be stored using $O(\log n)$ bits (since we only need to maintain the hash function and sign function values). This is a significant reduction from the $O(n)$ space required to store the entire vector x.

Deterministic Lower Bound

Could one design a *deterministic* algorithm for testing if $x = 0^n$ using fewer than $\Omega(n \log n)$ bits of space? It turns out that this is impossible.

Consider the following argument based on the pigeonhole principle:

- Let a denote the state of the vector after the first half of the stream updates. There are $\text{poly}(n)^n$ possible vectors a, which implies that any deterministic algorithm must use at least $\Omega(n \log n)$ bits to uniquely represent all possibilities.
- If an algorithm uses fewer than $\Omega(n \log n)$ bits, then by the pigeonhole principle, there must be two different vectors a_1 and a_2 that are mapped to the same representation.
- Now, consider the case when the second half of the stream applies updates corresponding to $-a_1$. Ideally, when the initial state is a_1, the updates would cancel a_1 to yield 0. However, if the initial state was a_2 (which shares the same representation as a_1), then after applying $-a_1$, the resulting vector would be $a_2 - a_1 \neq 0$.
- Since the algorithm cannot distinguish between a_1 and a_2 (because they share the same representation), it would produce the same output in both cases, causing an error on at least one of them.

This reasoning shows that any deterministic algorithm must use at least $\Omega(n \log n)$ bits to correctly test if $x = 0^n$ in the turnstile streaming model.

Example: Recovering a k-Sparse Vector

- Sparsity Promise: We assume that the vector $x \in \mathbb{R}^n$ ends up with at most k nonzero entries after all updates in the stream.
- Motivation: Often, k is small. For instance, consider two vectors a and b such that b is âsimilarâ to a, so $a - b$ has at most k nonzero coordinates.
- Recovery Goal: We want to identify both the indices and values of those k nonzero entries with high probability.
- Space Complexity: Is it possible to accomplish this using $k \cdot \text{poly}(\log n)$ bits of space?
- Deterministic vs. Randomized: Another question is whether this recovery can be done deterministically under similar or slightly higher space bounds.

Suppose A is an $s \times n$ matrix such that any $2k$ columns are linearly independent. We maintain the product Ax throughout the stream. The key claim is that from Ax, one can recover the subset S of k nonzero entries of x and their values.

To see why this is true, assume there are two vectors x and y, each having at most k nonzero entries, and suppose $Ax = Ay$. Then $A(x - y) = 0$. Since $x - y$ has at most $2k$ nonzero entries and any $2k$ columns of A are linearly independent, it must follow that $x - y = 0$. Hence, $x = y$.

This argument shows that the algorithm is deterministic once A is chosen. However, an open question is whether there exist matrices A with a sufficiently small number of rows s that still satisfy this linear independence property for any $2k$ columns.