

CS-GY 6763: Lecture 4

Linear Sketching, Sparse Recovery, and L_0 Sampling

NYU Tandon School of Engineering, Prof. Rajesh Jayaram

Euclidean Dimensionality Reduction

Lemma (Johnson-Lindenstrauss, 1984)

For any two data points $\mathbf{y}, \mathbf{q} \in \mathbb{R}^d$ there exists a linear map $\Pi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ where $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ such that with probability $1 - \delta$,

$$(1 - \epsilon)\|\mathbf{q} - \mathbf{y}\|_2 \leq \|\Pi\mathbf{q} - \Pi\mathbf{y}\|_2 \leq (1 + \epsilon)\|\mathbf{q} - \mathbf{y}\|_2.$$

Euclidean Dimensionality Reduction

Lemma (Johnson-Lindenstrauss, 1984)

For any set of n data points $\mathbf{q}_1, \dots, \mathbf{q}_n \in \mathbb{R}^d$ there exists a linear map $\Pi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ where $k = O\left(\frac{\log(n/\delta)}{\epsilon^2}\right)$ such that with probability $(1 - \delta)$, for all i, j ,

$$(1 - \epsilon)\|\mathbf{q}_i - \mathbf{q}_j\|_2 \leq \|\Pi\mathbf{q}_i - \Pi\mathbf{q}_j\|_2 \leq (1 + \epsilon)\|\mathbf{q}_i - \mathbf{q}_j\|_2.$$

Extends to approximating all pairwise distances in a set of n vectors via a **union bound**.

JL Application: Nearest Neighbor Search

Definition (Nearest Neighbor Problem)

Given a dataset $\mathcal{D} = \{p_1, p_2, \dots, p_n\} \in \mathbb{R}^d$ of n points in high-dimensional space, preprocess \mathcal{D} so that one can quickly answer the following queries:

1. Insert a point p into \mathcal{D}
2. Delete a point p from \mathcal{D}
3. Given a query point $q \in \mathbb{R}^d$, return a point $p \in \mathcal{D}$ with
$$\|p - q\|_2 \leq (1 + \epsilon) \min_{p' \in \mathcal{D}} \|q - p'\|_2$$

JL Application: Nearest Neighbor Search

Dataset: $\mathcal{D} = \{p_1, p_2, \dots, p_n\} \in \mathbb{R}^d$, query point $q \in \mathbb{R}^d$, want to find $p \in \mathcal{D}$ with $\|p - q\|_2 \leq (1 + \epsilon) \min_{p' \in \mathcal{D}} \|q - p'\|_2$.

First Try: Linear (Brute-force) scan: $O(d \cdot n)$ time per query.

Better with JL:

- Set $\tilde{p}_i = \Pi p_i$ and $\tilde{q} = \Pi q$ where $\Pi \in \mathbb{R}^{k \times n}$ is a random JL matrix with $k = O(\frac{\log n}{\epsilon^2})$.
- Return $p = \arg \min_{p_i} \|\tilde{q} - \tilde{p}_i\|_2$.

Larger $O(dn \frac{\log n}{\epsilon^2})$ preprocessing time, but each query is now faster at time $O(n \frac{\log n}{\epsilon^2})$. After k queries, the total runtime is better!

How many queries can we answer correctly with probability 9/10?

Definition

Given any high-dimensional vector $f \in \mathbb{R}^n$, a *linear sketch* is a matrix-vector product $Sf \in \mathbb{R}^k$, $k \ll n$, where $S \in \mathbb{R}^{k \times n}$ is called a *sketching matrix*.

- Usually, S is a *random* matrix, whose entries can be easily stored (e.g., S is generated by 2-wise independent hash functions).
- **Goal:** from knowledge only of S and Sf , approximate some function of f (i.e. $\|f\|_2^2$, find heavy hitters $f_i > \epsilon \|f\|_2$, etc.)
- **Benefits:** S much smaller to store than f , can be maintained in a stream!

We have seen these before: Count-Min, Count-Sketch, Johnson-Lindenstrauss...

Main Benefit of linear sketches:

Linear sketches are Linear!

$$\mathbf{S}x + \mathbf{S}y = \mathbf{S}(x + y)$$

How is this useful for streaming? Suppose we have $\mathbf{S}f$ stored, and f gets an update $(i, \Delta) \in [n] \times \mathbb{Z}$.

$$f_i \leftarrow f_i + \Delta$$

$$\mathbf{S}f \leftarrow \mathbf{S}f + \Delta \cdot \mathbf{e}_i$$

$$\left(f + \Delta \vec{e}_i \right)$$

Main Benefit of linear sketches:

Linear Sketches are linear!

$$\mathbf{S}x + \mathbf{S}y = \mathbf{S}(x + y)$$

How is this useful for streaming? Suppose we have $\mathbf{S}f$ stored, and f gets an update $(i, \Delta) \in [n] \times \mathbb{Z}$.

$$\mathbf{S}f \leftarrow \mathbf{S}f + \mathbf{S}_i \cdot \Delta$$

Also very useful for **distributed computation**. Machines m_1, \dots, m_k have data $x_1, \dots, x_k \in \mathbb{R}^n$. Each machine can sketch $\mathbf{S}x_i \in \mathbb{R}^k$, send to aggregator, which computes $\sum_i \mathbf{S}x_i = \mathbf{S}(\sum_i x_i)$.

Johnson Lindenstrauss for Streaming

JL-Lemma says that there exists a linear sketch $\mathbf{S} \in \mathbb{R}^{k \times n}$, where $k = \frac{1}{\epsilon^2} \log(1/\delta)$, such that for any fixed $f \in \mathbb{R}^n$, we have $\|\mathbf{S}f\|_2^2 = (1 \pm \epsilon)\|f\|_2^2$ with prob $1 - \delta$. So:

Theorem

There is a turnstile streaming algorithm (in the random oracle model) which estimates the ℓ_2 norm $\|f\|_2^2$ of the frequency vector to $(1 \pm \epsilon)$ -multiplicative error with probability $> 1 - \delta$, using $O(\frac{1}{\epsilon^2} \log(1/\delta))$ bits of space.

Recall: Random oracle model of streaming means we can store random bits for free.

Recall: Count-Sketch

- Choose random hash functions $h_1, h_2, \dots, h_t : [n] \rightarrow [B]$, and $\sigma_1, \sigma_2, \dots, \sigma_t : [n] \rightarrow \{1, -1\}$, where $t = O(\log 1/\delta)$ and $B = O(1/\epsilon^2)$.
- For each update $\ell = 1, \dots, m$
 - Given update (i_ℓ, Δ_ℓ) , for each $j = 1, 2, \dots, t$ set

$$\mathbf{A}_j[h_j(i_\ell)] = \mathbf{A}[h(i_\ell)] + \sigma(i_\ell) \cdot \Delta_\ell$$

A_1	-4	-5	0	20	0	-2	7	-4	-2	0	2	0
-------	----	----	---	----	---	----	---	----	----	---	---	---

A_2	0	5	-20	-10	0	5	0	0	6	7	4	2
-------	---	---	-----	-----	---	---	---	---	---	---	---	---

⋮

A_t	2	0	0	-17	0	0	4	2	1	22	3	-8
-------	---	---	---	-----	---	---	---	---	---	----	---	----

Recall: Count-Sketch

We can stack $\mathbf{A}_1, \dots, \mathbf{A}_t \in \mathbb{R}^B$ together to form a vector $\mathbf{A} \in \mathbb{R}^{Bt}$, such that $\mathbf{A} = \mathbf{S} \cdot \mathbf{f}$ for some sketching matrix \mathbf{S} .

$$\mathbf{A} = \begin{array}{|c|} \hline \mathbf{A}_1 \\ \hline \mathbf{A}_2 \\ \hline \mathbf{A}_3 \\ \hline \vdots \\ \hline \mathbf{A}_t \\ \hline \end{array} = \mathbf{S} \mathbf{f}$$

Count-Sketch is a Linear Sketch

Can define the count-sketch sketching matrix $\mathbf{S} \in \mathbb{R}^{tB \times n}$ via stacking t identically distributed sketching matrices $\mathbf{S}^1, \mathbf{S}^2, \dots, \mathbf{S}^t$

$$\mathbf{S}^i = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & -1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -1 & 0 & 0 & \dots & 1 & 0 & 0 \end{bmatrix}$$

Each column \mathbf{S}_j^i of \mathbf{S}^i has one non-zero entry, placed in a random row $h_i(j)$, and given a random sign $\sigma_i(j)$.

Count-Sketch is a Linear Sketch

Can define the count-sketch sketching matrix $\mathbf{S} \in \mathbb{R}^{tB \times n}$ via stacking t identically distributed sketching matrices $\mathbf{S}^1, \mathbf{S}^2, \dots, \mathbf{S}^t$

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}^1 \\ \mathbf{S}^2 \\ \mathbf{S}^3 \\ \vdots \\ \mathbf{S}^t \end{bmatrix}$$

Then $\mathbf{A} = \mathbf{S} \cdot f$, so count-sketch is linear, where \mathbf{S} has $O(\log 1/\delta)$ non-zero entries per column. This is sparse!

Count-Sketch is a Linear Sketch

Detour on k -wise independent hashing

Note on Hash Functions

We often assume we have truly random hash functions h , can we weaken this?

Definition (Universal hash function)

A ~~random~~ hash function $h : \mathcal{U} \rightarrow \{1, \dots, B\}$ is universal if, for any fixed $x, y \in \mathcal{U}$,

$$\Pr[h(x) = h(y)] \leq \frac{1}{B}.$$

Claim: A uniformly random hash-function is universal.

Efficient alternative: Let p be a prime number between $|\mathcal{U}|$ and $2|\mathcal{U}|$. Let a, b be random numbers in $\{0, \dots, p\}$ with $a \neq 0$.

$$h(x) = [a \cdot x + b \pmod{p}] \pmod{B}$$

is universal. Note we only need to store a, b !

Note on Hash Functions

Another definition you might come across:

Definition (Pairwise independent hash function)

A random hash function $h : \mathcal{U} \rightarrow \{1, \dots, B\}$ is pairwise independent if, for any fixed $x, y \in \mathcal{U}, i, j \in \{1 \dots, B\}$,

$$\Pr[h(x) = i \cap h(y) = j] = \frac{1}{B^2}.$$

We can naturally extended to k -wise independence for $k > 2$, which is strictly stronger. Need degree k polynomials, which requires $O(k)$ space to store coefficients.

Note on Hash Functions

Extension to k -wise independence

Definition (k -wise independent hash function)

A random hash function $h : \mathcal{U} \rightarrow \{1, \dots, B\}$ is k -wise independent for $k \geq 2$ if, for any fixed $x_1, x_2, \dots, x_k \in \mathcal{U}$ and $i_1, \dots, i_k \in \{1, \dots, B\}$,

$$\Pr[h(x_1) = i_1 \cap h(x_2) = i_2 \cap \dots \cap h(x_k) = i_k] = \frac{1}{B^k}.$$

k -wise independence implies $k - 1$ -wise independence for all $k' < k$, can you see why?

or

Note on Hash Functions

k -wise independence implies $k - 1$ -wise independence for all $k' < k$, can you see why?

Proof:

$$\begin{aligned} & \Pr [h(x_1) = i_1, \dots, h(x_{k-1}) = i_{k-1}] \\ &= \sum_{i_k} \Pr [h(x_1) = i_1, \dots, h(x_{k-1}) = i_{k-1}, h(x_k) = i_k] \\ & \sum_{i_k} \frac{1}{B^k} = \frac{1}{B^{k-1}} \end{aligned}$$

Note on Hash Functions

Definition (k -wise independent hash function)

A random hash function $h : \mathcal{U} \rightarrow \{1, \dots, B\}$ is k -wise independent for $k \geq 2$ if, for any fixed $x_1, x_2, \dots, x_k \in \mathcal{U}$ and $i_1, \dots, i_k \in \{1, \dots, B\}$,

$$\Pr[h(x_1) = i_1 \cap h(x_2) = i_2 \cap \dots \cap h(x_k) = i_k] = \frac{1}{B^k}.$$

Facts: if $h : \mathcal{U} \rightarrow \{1, \dots, B\}$ is k -wise independent, then for any fixed $x_1, x_2, \dots, x_k \in \mathcal{U}$ we have

$$\mathbb{E}[h(x_1)h(x_2) \cdots h(x_k)] = \mathbb{E}[h(x_1)]\mathbb{E}[h(x_2)] \cdots \mathbb{E}[h(x_k)]$$

More generally, for any powers $p_1, p_2, \dots, p_k \geq 0$:

$$\mathbb{E}[h(x_1)^{p_1}h(x_2)^{p_2} \cdots h(x_k)^{p_k}] = \mathbb{E}[h(x_1)^{p_1}]\mathbb{E}[h(x_2)^{p_2}] \cdots \mathbb{E}[h(x_k)^{p_k}]$$

Revisiting Count-Sketch with pairwise independence.

- Choose random **2-wise independent** hash functions $h_1, h_2, \dots, h_t : [n] \rightarrow [B]$, and $\sigma_1, \sigma_2, \dots, \sigma_t : [n] \rightarrow \{1, -1\}$, where $t = O(\log 1/\delta)$ and $B = O(1/\epsilon^2)$.
- For each update $\ell = 1, \dots, m$
 - Given update (i_ℓ, Δ_ℓ) , for each $j = 1, 2, \dots, t$ set

$$\mathbf{A}_j[h_j(i_\ell)] = \mathbf{A}[h(i_\ell)] + \sigma(i_\ell) \cdot \Delta_\ell$$

A_1	-4	-5	0	20	0	-2	7	-4	-2	0	2	0
-------	----	----	---	----	---	----	---	----	----	---	---	---

A_2	0	5	-20	-10	0	5	0	0	6	7	4	2
-------	---	---	-----	-----	---	---	---	---	---	---	---	---

⋮

A_t	2	0	0	-17	0	0	4	2	1	22	3	-8
-------	---	---	---	-----	---	---	---	---	---	----	---	----

Revisiting Count-Sketch with pairwise independence.

When estimating \tilde{f}_i for f_i , for the error in level $s \in [t]$ we have

$$\mathbb{E} \left[\sum_{j, h_s(j)=h_s(i)} \sigma_s(j) f_j \right] = 0$$

Recall, variance is linear so long as the variables are pairwise independent!

$$\text{Var} \left[\sum_{j, h_s(j)=h_s(i)} \sigma_s(j) f_j \right] = \sum_{j, h_s(j)=h_s(i)} \text{Var}(\sigma_s(j) f_j) \leq \frac{\|f\|_2^2}{B}$$

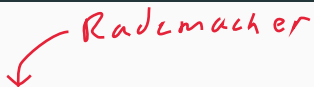
Where we used pairwise independence: $\Pr[h_s(j) = h_s(i)] = 1/B$.

Worth going back to check that everything goes through from here with limited independence.

Back to Linear Sketching

AMS Sketch, alternative to JL

Rademacher



Let $h_1, \dots, h_k : [n] \rightarrow \{-1, 1\}$ be 4-wise independent hash functions, and let $\mathbf{S} \in \mathbb{R}^{k \times n}$ be a random matrix define by $\mathbf{S}_{i,j} = h_i(j)$. Note \mathbf{S} can be stored in $O(k)$ words of space.

Theorem (Alon, Matias, Szegedy '96)

If S is the random matrix from above, with $k = O(\frac{1}{\epsilon^2})$, then for any fixed $x \in \mathbb{R}^n$ with probability at least 9/10 we have

$$\left\| \frac{1}{\sqrt{k}} \mathbf{S}x \right\|_2^2 = (1 \pm \epsilon) \|x\|_2^2$$

AMS Sketch, alternative to JL

Let $h_1, \dots, h_k : [n] \rightarrow \{-1, 1\}$ be 4-wise independent hash functions, and let $\mathbf{S} \in \mathbb{R}^{k \times n}$ be a random matrix define by $S_{i,j} = h_i(j)$. Note \mathbf{S} can be stored in $O(k)$ words of space.

Theorem (Alon, Matias, Szegedy '96)

If S is the random matrix from above, with $k = O(\frac{1}{\epsilon^2})$, then for any fixed $x \in \mathbb{R}^n$ with probability at least 9/10 we have

$$\left\| \frac{1}{\sqrt{k}} \mathbf{S}x \right\|_2^2 = (1 \pm \epsilon) \|x\|_2^2$$

This is perhaps **the** seminal paper in streaming algorithms. The authors won the Gödel prize for this work!

AMS Sketch: another linear sketch!

Let $h_1, \dots, h_k : [n] \rightarrow \{-1, 1\}$ be 4-wise independent hash functions, and $\mathbf{S}_{i,j} = h_i(j)$.

Proof: Lets look at a single entry of $(\mathbf{S}x)_1 = \langle \mathbf{S}_1, x \rangle$.

$$\begin{aligned}\mathbb{E} [\langle \mathbf{S}_1, x \rangle^2] &= \sum_i \mathbb{E} \left[\left(\sum_i x_i h_1(i) \right)^2 \right] \\ &= \sum_i \mathbb{E}[x_i^2 h_1^2(i)] + \sum_{i \neq j} \mathbb{E}[x_i h_1(i) x_j h_1(j)] \\ &= \|x\|_2^2\end{aligned}$$

So $\mathbb{E}[\|k^{-1/2} \mathbf{S}x\|_2^2] = k \cdot (1/k) \|x\|_2^2 = \|x\|_2^2$.


AMS Sketch: another linear sketch!

$$\begin{aligned}\text{Var}[\langle S_1, x \rangle^2] &\leq \mathbb{E} \left[\left(\sum_{i,j} x_i x_j h(i) h(j) \right)^2 \right] - \|x\|_2^4 \\ &\leq \mathbb{E}[(\sum_{i,j} x_i x_j h(i) h(j))^2] - \sum_{i,j} x_i^2 x_j^2 \\ &= \sum_{\substack{i \neq j, \text{ or} \\ t \neq \ell}} \mathbb{E}[x_i x_j x_t x_\ell h(i) h(j) h(t) h(\ell)] \\ &= \sum_{\substack{i \neq j \\ t \neq \ell}} \mathbb{E}[x_i x_j x_t x_\ell h(i) h(j) h(t) h(\ell)] \\ &= \sum_{i \neq t} x_i^2 x_t^2 \leq (\sum_i x_i^2)^2 = \|x\|_2^4\end{aligned}$$

AMS Sketch: another linear sketch!

Let $h_1, \dots, h_k : [n] \rightarrow \{-1, 1\}$ be 4-wise independent hash functions, $k = 10/\epsilon^2$ and $\mathbf{S}_{i,j} = h_i(j)$.

Wrapping up: $\mathbb{E}[\|k^{-1/2}\mathbf{S}\mathbf{x}\|_2^2] = \|\mathbf{x}\|_2^2$, and


$$\frac{1}{k} \text{Var}(\|k^{-1/2}\mathbf{S}\mathbf{x}\|_2^2) = \frac{1}{k} \sum_{i=1}^k \text{Var}(\langle S_i, \mathbf{x} \rangle^2) \leq \frac{1}{k} \|\mathbf{x}\|_2^4$$

By Chebyshev's Inequality:

$$\Pr \left[\left| \|k^{-1/2}\mathbf{S}\mathbf{x}\|_2^2 - \|\mathbf{x}\|_2^2 \right| \geq \epsilon \|\mathbf{x}\|_2^2 \right] \leq \frac{1}{\epsilon^2 \|\mathbf{x}\|_2^4} \cdot \frac{\|\mathbf{x}\|_2^4}{k} \leq 1/100$$

AMS Sketch, Summary

Let $h_1, \dots, h_k : [n] \rightarrow \{-1, 1\}$ be 4-wise independent hash functions, and let $\mathbf{S} \in \mathbb{R}^{k \times n}$ be a random matrix define by $\mathbf{S}_{i,j} = h_i(j)$. Note \mathbf{S} can be stored in $O(k)$ words of space.

Theorem (Alon, Matias, Szegedy '96)

If S is the random matrix from above, with $k = O(\frac{1}{\epsilon^2})$, then for any fixed $x \in \mathbb{R}^n$ with probability at least 9/10 we have

$$\left\| \frac{1}{\sqrt{k}} \mathbf{S} x \right\|_2^2 = (1 \pm \epsilon) \|x\|_2^2$$

Note that if we used **fully random** hash functions h_1, \dots, h_k , we could have used Chernoff bounds instead of Chebyshev inequality (why?), to obtain failure probability δ using $k = O(\log(1/\delta)/\epsilon^2)$. This gives an alternative to Gaussian matrices for JL!

Linear Sketching and Turnstile Streaming

An important observation researchers made after 20 years of designing turnstile streaming algorithms: **almost every known turnstile streaming algorithm was a linear sketch!**

It turns out, there is a **formal equivalence**, under some (admittedly non-trivial) assumptions.

Turnstile Streaming Algorithms Might as Well Be Linear Sketches,
Yi Li, Huy L. Nguyễn, David P. Woodruff (STOC 14).

You will not need to know the details of this, or use this connecting, but it is useful to keep in mind, and a beautiful result!

All turnstile streaming algorithms in this course will be linear sketches (almost certainly).

Sampling via Linear Sketches

In the HW, we saw how to estimate the number of non-zero elements $\|f\|_0 = |\{i \in [n] | f_i \neq 0\}|$ in *insertion-only* streams.

Algorithm for Distinct Elements Estimation

1. Select a uniformly random hash function $h : [n] \rightarrow [0, 1]$.
2. For each update $(a_i, \Delta) \in [n]$ in the stream, compute $h(a_i) \in [0, 1]$, and store it if it is one of the the k -smallest unique hash values seen so far.
3. At the end of the stream, let $h_1 < h_2 < \dots < h_k \in [0, 1]$ be the k -smallest hash values seen during the stream (i.e., the k -smallest real numbers in the set $\{h(i) : i \in [n], f_i \neq 0\}$), which the algorithm now has stored. Output the estimator $R = \frac{1}{h_k} \cdot k$.

In the HW, we saw how to estimate the number of non-zero elements $\|f\|_0 = |\{i \in [n] | f_i \neq 0\}|$ in *insertion-only* streams.

What if we just wanted to sample a non-zero f_i ?

Algorithm for sampling non-zero coordinates

1. Select a uniformly random hash function $h : [n] \rightarrow [0, 1]$.
2. For each update $(a_i, \Delta) \in [n]$ in the stream, compute $h(a_i) \in [0, 1]$, and store the tuple $(a_i, h(a_i))$ if it's the smallest hash value seen so far.
3. At the end of the stream, let $(a_i, h(a_i))$ be the stored tuple, and output the sample $a_i \in [n]$.

Algorithm for sampling non-zero coordinates

1. Select a uniformly random hash function $h : [n] \rightarrow [0, 1]$.
2. For each update $(a_i, \Delta) \in [n]$ in the stream, compute $h(a_i) \in [0, 1]$, and store the tuple $(a_i, h(a_i))$ if it's the smallest hash value seen so far.
3. At the end of the stream, let $(a_i, h(a_i))$ be the stored tuple, and output the sample $a_i \in [n]$.

This gives a uniform sample from $\{i \in [n] \mid f_i \neq 0\}$.

Proof: each $h(a_i)$ equally likely to be the smallest hash

Algorithm for sampling non-zero coordinates

1. Select a uniformly random hash function $h : [n] \rightarrow [0, 1]$.
2. For each update $(a_i, \Delta) \in [n]$ in the stream, compute $h(a_i) \in [0, 1]$, and store the tuple $(a_i, h(a_i))$ if it's the smallest hash value seen so far.
3. At the end of the stream, let $(a_i, h(a_i))$ be the stored tuple, and output the sample $a_i \in [n]$.

This gives a uniform sample from $\{i \in [n] \mid f_i \neq 0\}$.

Proof: each $h(a_i)$ equally likely to be the smallest hash

What happens if there are deletions in the stream? Also, is this sketch is linear?

L_0 Sampling via Linear Sketches

Key issue with min-hash sampling: many points may be deleted after inserted – cannot keep track of the smallest hash.

Solution: We follow a two-level approach:

1. First, we guess the value of $\|f\|_0$ up to a factor of 2 error, by making the guesses $r = 1, 2, 4, 8, \dots, n$.
2. Next, for each guess r , we try to sample a uniformly random coordinate of $\|f\|_0$ by subsampling the coordinates of f at rate $\frac{1}{r}$.
3. If our guess was correct, and $\frac{1}{4}\|f\|_0 \leq r < \frac{1}{2}\|f\|_0$, then we expect at most $O(1)$ -coordinates to survive the sampling!

This approach of "guessing" an unknown value to a factor of 2 in geometric intervals is extremely common in algorithm design!

Sparse Recovery

Suppose we had the right guess of r , with $\frac{1}{4}\|f\|_0 \leq r < \frac{1}{2}\|f\|_0$.
Once we subsample coordinates at rate $1/r$, how do we find the $O(1)$ -expected coordinates that survive?

Warning! Cannot just try to store them as they arrive!

Example: Suppose we had $\|f\|_0 = 1$ at the end of the stream, which consisted of first inserting $+1$ to each coordinate, so $f = (1, 1, \dots, 1)$, and then deleting all 1's except one random index f_i , so $f = (0, 0, \dots, 0, 1, 0, \dots, 0)$ at the end of the stream

If we sampled items at rate $1/\|f\|_0 = 1$, what would our space be?

Sparse Recovery

To solve this task, we introduce the central notation of *sparse recovery*.

Definition

Given a unknown vector $f \in \mathbb{R}^n$ with at most k non-zero coordinates (i.e. $\|f\|_0 \leq k$), design a randomized linear sketch $\mathbf{S} \in \mathbb{R}^{t \times n}$ such that from $\mathbf{S}f, \mathbf{S}$, one can *exactly* recover f .

In the streaming version of the problem, the entries of f are seen in a (turnstile) stream, and we are not restricted to using a linear sketch.

- Could take $\mathbf{S} = I_n \in \mathbb{R}^{n \times n}$, but this is very inefficient.

Since linear sketches imply streaming algorithms, useful to start by thinking about it from a streaming perspective.

Definition

Given a unknown vector $f \in \mathbb{R}^n$ with at most k non-zero coordinates (i.e. $\|f\|_0 \leq k$), design a turnstile streaming algorithm to *exactly* recover f .

Simplification: Suppose we got **two passes*** over the stream. In other words, we see the stream of updates $(i_1, \Delta_1), (i_2, \Delta_2), \dots, (i_m, \Delta_m)$ twice, in the same order. Also assume that f has non-negative coordinates.

*This is actually an important streaming model: the *multi-pass streaming model*

Sparse Recovery: A two pass algorithm

$f \in \mathbb{R}^n$ a non-negative k -sparse vector, given by a stream $(i_1, \Delta_1), \dots, (i_m, \Delta_m)$ which we see twice.

Solution #1: Think about the covid testing scheme from the HW. Can you modify the first scheme from there to design a two-pass streaming algorithm using space $O(\sqrt{kn})$?

$C := \# \text{ of Buckets}$

Sparse Recovery: A two pass algorithm

Pass 1:

$$\frac{C}{c} = \frac{\sum_n}{\sum_k}$$

1. Partition the coordinates into $C = O(\sqrt{kn})$ buckets of equal size. Keep a counter c_j for each $j \in [C]$. Let $b(i) \in [C]$ be the bucket containing coordinate $i \in [n]$.
2. Each update (i, Δ) , update the counter $c_{b(i)} \leftarrow c_{b(i)} + \Delta$.

Pass 2:

1. Let $S \subset [C]$ be the set of non-zero counter ($c_j \neq 0$) from the first pass. Note that $|S| \leq k$. Store the exact value of all coordinates f_i with $b(i) \in S$.
2. At the end of the second pass, we will have recovered all k non-zero coordinates of f exactly. **Why?**

Sparse Recovery: A two pass algorithm

Space Complexity: First pass, we store $C = O(\sqrt{kn})$ counters, so $O(\sqrt{kn})$ space. Second pass, we “retest” the whole set of coordinates that belong a bucket j with $c_j \neq 0$. Since each bucket contains $n/C = O(\sqrt{n/k})$ coordinates, we store at most $kn/c = O(\sqrt{nk})$ coordinates on the second pass.

Correctness:

Sparse Recovery: A two pass algorithm

Space Complexity: First pass, we store $C = O(\sqrt{kn})$ counters, so $O(\sqrt{kn})$ space. Second pass, we “retest” the whole set of coordinates that belong a bucket j with $c_j \neq 0$. Since each bucket contains $n/C = O(\sqrt{n/k})$ coordinates, we store at most $kn/c = O(\sqrt{nk})$ coordinates on the second pass.

Correctness: Since $f_i \geq 0$ for all $i \in [n]$, we have $c_j > 0$ at the end of the stream for every bucket j containing a non-zero coordinate.

For small k , this is significantly better than $O(n)$ space!

Definition

Given a unknown vector $f \in \mathbb{R}^n$ with at most k non-zero coordinates (i.e. $\|f\|_0 \leq k$), design a turnstile streaming algorithm to *exactly* recover f .

We just showed that there is a **two-pass** streaming algorithm using $O(\sqrt{kn})$ space, so long as f has non-negative coordinates.

Removing the assumptions: Let's now try to adapt the second part of the Covid Testing problem to obtain a single pass algorithm for *any* f !

Sparse Recovery

Definition

Given a unknown vector $f \in \mathbb{R}^n$ with at most k non-zero coordinates (i.e. $\|f\|_0 \leq k$), design a turnstile streaming algorithm to *exactly* recover f .

Intuitively: What did the second Covid testing scheme do...split $[n]$ into $O(k)$ buckets, test everyone in a bucket, and repeat $O(\log n)$ times.

- For covid testing, we wanted negative patients not to collide with positive patients at least once.
- For k -sparse recovery, we just need positive patients not to collide with other positive patients *most of the time*, so we can tell their values apart.
- Does this sound like another algorithm we've seen?

We proved in the homework:

Theorem (Count-Sketch with tail error)

There is a linear sketch $\mathbf{S} \in \mathbb{R}^{t \times n}$ with $t = O(B \cdot \log n)$, such that, given $\mathbf{S} \cdot f$ and \mathbf{S} we can produce an estimate $\tilde{f} \in \mathbb{R}^n$, such that with probability $1 - 1/n$ for every $i \in [n]$ we have:

$$|\tilde{f}_i - f_i| \leq \frac{1}{\sqrt{B}} \|f_{-B}\|_2$$

Setting $B = k$, for the k -sparse recovery problem $\|f_{-k}\|_2 = \bigcirc$

We proved in the homework:

Theorem (Count-Sketch with tail error)

There is a linear sketch $\mathbf{S} \in \mathbb{R}^{t \times n}$ with $t = O(B \cdot \log n)$, such that, given $\mathbf{S} \cdot f$ and \mathbf{S} we can produce an estimate $\tilde{f} \in \mathbb{R}^n$, such that with probability $1 - 1/n$ for every $i \in [n]$ we have:

$$|\tilde{f}_i - f_i| \leq \frac{1}{\sqrt{B}} \|f_{-B}\|_2$$

Setting $B = k$, for the k -sparse recovery problem $\|f_{-k}\|_2 =$

- In $O(k \log n)$ words of space, we recover a k -sparse vector f exactly from \mathbf{S} with probability $1 - \frac{1}{n}$!

Back to L_0 Sampling: The Full Algorithm

Set $k = 100 \log n$.

guess $1/f_0 \sim 2^i$

1. For each $i = 1, 2, \dots, \log n$:

2^i

- Sample each $j \in [n]$ independently with probability $2^i/n$ (can be done via truly random hash function $h : [n] \rightarrow [n/2^i]$). Let $A_i \subset [n]$ be the set of sampled coordinates
- Instantiate Count-Sketch \mathbf{S}_i for k -sparse recovery.
- Feed updates to coordinates in A_i to sketch \mathbf{S}_i .
- Get estimate vector \tilde{f}^i from \mathbf{S}_i .

*$5[1A.]$
 $= 2^i$*

2. Find the smallest i such that $\tilde{f}^i \neq 0$, and return a uniformly random $u \sim \{j \in [n] \mid \tilde{f}_j^i \neq 0\}$.

Claim: u is drawn uniformly from $\{j \in [n] \mid f_j \neq 0\}$.

Back to L_0 Sampling: The Full Algorithm

Set $k = 100 \log n$.

1. For each $i = 1, 2, \dots, \log n$:
 - Sample each $j \in [n]$ independently with probability $2^j/n$ (can be done via truly random hash function $h : [n] \rightarrow [n/2^j]$). Let $A_i \subset [n]$ be the set of sampled coordinates
 - Instantiate Count-Sketch \mathbf{S}_i for k -sparse recovery.
 - Feed updates to coordinates in A_i to sketch \mathbf{S}_i .
 - Get estimate vector \tilde{f}^i from \mathbf{S}_i .
2. Find the smallest i such that $\tilde{f}^i \neq 0$, and return a uniformly random $u \sim \{j \in [n] \mid \tilde{f}_j^i \neq 0\}$.

Claim: u is drawn uniformly from $\{j \in [n] \mid f_j \neq 0\}$.

Space $O(\log^3 n)$ words: $\log n$ independent copies of Count-Sketch for k -sparse recovery, with $k = O(\log n)$

Claim: With probability $1 - 1/n$, u is a uniformly random non-zero coordinate of f .

Proof: Each count-sketch is correct with probability $1 - 1/n^2$, (making failure probability slightly smaller). By a union bound, all $\log n$ are correct with probability greater than $1 - 1/n$. Condition on this now, and let i^* be such that $k/8 \leq \frac{2^{i^*}}{n} \cdot \|f\|_0 \leq k/4$

Proof: Each count-sketch is correct with probability $1 - 1/n^2$, (making failure probability slightly smaller). By a union bound, all $\log n$ are correct with probability greater than $1 - 1/n$. Condition on this now, and let i^* be such that $k/8 \leq \frac{2^{i^*}}{n} \cdot \|f\|_0 \leq k/4$

Claim

With probability $1 - 1/n^2$ we have $|A_j| \leq k$ for each $j = 1, 2, \dots, i^*$. ^ 156.

Proof: Each count-sketch is correct with probability $1 - 1/n^2$, (making failure probability slightly smaller). By a union bound, all $\log n$ are correct with probability greater than $1 - 1/n$. Condition on this now, and let i^* be such that $k/8 \leq \frac{2^{i^*}}{n} \cdot \|f\|_0 \leq k/4$

Claim

With probability $1 - 1/n^2$ we have $|A_j| \leq k$ for each $j = 1, 2, \dots, i^$.*

Proof: a coordinate t is sampled into $\cup_{j=1}^{i^*} A_j$ with probability at most $\sum_{j=1}^{i^*} 2^j/n \leq 2^{i^*+1}/n$. So the expected number of coordinates in $\cup_{j=1}^{i^*} A_j$ is at most $2^{i^*+1}/n \|f\|_0 \leq k/2$. By Chernoff:

$$\Pr \left[\left| \cup_{j=1}^{i^*} A_j \right| > \frac{k}{2} \left(1 + \frac{1}{2} \right) \right] < e^{-\frac{k/2}{3.4}} < e^{-(100 \log n)/48} < 1/n^2$$

We now have that $|A_j| \leq k$ for each $j = 1, 2, \dots, i^*$ with probability $1 - 1/n$, where i^* is the index such that $k/8 \leq \frac{2^{i^*}}{n} \cdot \|f\|_0 \leq k/4$.

Claim

With probability $1 - 1/n^2$ we have $k/16 \leq |A_{i^}|$.*

We have $\mathbb{E}[|A_{i^*}|] = \frac{2^{i^*}}{n} \cdot \|f\|_0 > k/8 = 100 \log n/8$. Claim follows by another Chernoff bound

L_0 Sampling Putting everything together

The last two claims together (with a union bound) yield:

Lemma

There exists an index $i^ \in [\log n]$ such that with probability $1 - 2/n^2$ we have $|A_j| \leq k$ for each $j = 1, 2, \dots, i^*$, and $k/16 \leq |A_{i^*}|$.*

L_0 Sampling Putting everything together

The last two claims together (with a union bound) yield:

Lemma

There exists an index $i^ \in [\log n]$ such that with probability $1 - 2/n^2$ we have $|A_j| \leq k$ for each $j = 1, 2, \dots, i^*$, and $k/16 \leq |A_{i^*}|$.*

It follows by the correctness of Count-Sketch for k -sparse recovery that $\tilde{f}^j = f^j$ for each $j = 1, 2, \dots, i^*$. It also follows that $\tilde{f}^j \neq 0$ for at least one $j \in [i^*]$. Fix the first such $j \leq i^*$.

The non-zero coordinates of f^j are uniformly drawn from the non-zero coordinates of f . Thus, a random non-zero from f^j is a uniformly random non-zero from f . ■

L_0 Sampling: The Full Algorithm

Set $k = 100 \log n$.

1. For each $i = 1, 2, \dots, \log n$:
 - Sample each $j \in [n]$ independently with probability $2^j/n$ (can be done via truly random hash function $h : [n] \rightarrow [n/2^j]$). Let $A_i \subset [n]$ be the set of sampled coordinates
 - Instantiate Count-Sketch \mathbf{S}_i for k -sparse recovery.
 - Feed updates to coordinates in A_i to sketch \mathbf{S}_i .
 - Get estimate vector \tilde{f}^i from \mathbf{S}_i .
2. Find the smallest i such that $\tilde{f}^i \neq 0$, and return a uniformly random $u \sim \{j \in [n] \mid \tilde{f}_j^i \neq 0\}$.

Claim: u is drawn uniformly from $\{j \in [n] \mid f_j \neq 0\}$.

L_0 Sampling: The Full Algorithm

We have proven the following theorem (in the random oracle streaming model):

Theorem

There is a turnstile streaming algorithm that, on a stream with frequency vector $f \in \mathbb{R}^n$, with probability $1 - O(1/n)$, samples a coordinate i uniformly from the non-zero coordinates of f . The space required is $O(\log^3 n)$ words.

Actually, we can do better for the k -sparse recovery problem. Namely, there is a *deterministic* linear sketch $\mathbf{S} \in \mathbb{R}^{t \times n}$ for k -sparse recovery, using $t = O(k)$ rows! \mathbf{S} is known as a *Vandermonde Matrix*. Using this \mathbf{S} instead of count-sketch gives $O(\log^2 n)$ space.

It turns out this is optimal!