Outline

- Quick recap of ℓ_1-regression, and how to speed it up
- Introduction to the Streaming Model
- Estimating Norms in the Streaming Model
- Heavy Hitters in a Stream
- Estimating Number of Non-Zero Entries (ℓ_0)

Heavy Hitter Guarantees

- l_1 – guarantee
  - output a set containing all items j for which $|x_j| \geq \phi |x|_1$
  - the set should not contain any j with $|x_j| \leq (\varphi - \varepsilon) |x|_1$

- l_2 – guarantee
  - output a set containing all items j for which $x_j^2 \geq \phi |x|_2^2$
  - the set should not contain any j with $x_j^2 \leq (\phi - \epsilon)|x|_2^2$

- l_2 – guarantee can be much stronger than the l_1 – guarantee
  - Suppose x = $(\sqrt{n}, 1, 1, 1, \ldots, 1)$
  - Item 1 is an l_2-heavy hitter for constant φ, ε, but not an l_1-heavy hitter
  - If $|x_j| \geq \phi |x|_1$, then $x_j^2 \geq \phi^2 |x|_1^2 \geq \phi^2 |x|_2^2$

Heavy Hitter Intuition

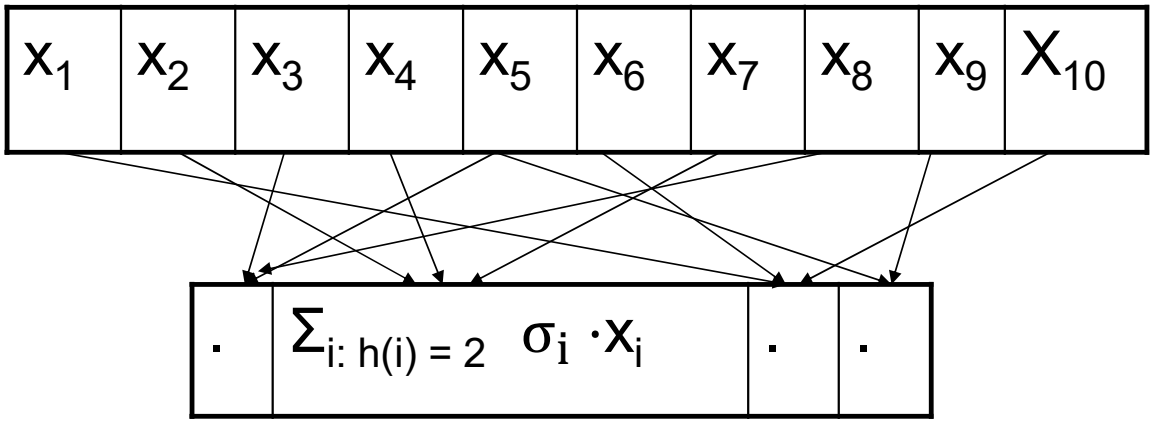
- Suppose you are promised at the end of the stream, $x_i = n$, and $x_j \in \{0,1\}$ for $j \in \{1, 2, \ldots, n\}$ with $j \neq i$
- How would you find the identity i?
- For each j in {1, 2, 3, …, log n}, let $A_j \subset [n]$ be the set of indices with j-th bit in their binary representation equal to 0, and B_j be the set with j-th bit equal to 1
- Compute $a_j = \sum_{i \in A_j} x_i$ and $b_j = \sum_{i \in B_j} x_i$ for each j in {1, 2, …, log n}
- Read off the identity of item i

Heavy Hitter Intuition Continued

- Suppose you are promised at the end of the stream, $x_i = 100\sqrt{n \log(n)}$, and $x_j \in \{0,1\}$ for $j \in \{1, 2, \ldots, n\}$ with $j \neq i$
- How would you find the identity i?
- For each j in {1, 2, 3, …, log n}, let $A_j \subset [n]$ be the set of indices with j-th bit in their binary representation equal to 0, and B_j be the set with j-th bit equal to 1
- Compute $a_j = \sum_{i \in A_j} \sigma_i \cdot x_i$ and $b_j = \sum_{i \in B_j} \sigma_i \cdot x_i$ for each j in {1, 2, …, log n}
- Read off the identity of item i?
- Additive Chernoff bound implies magnitude of "noise" in a count is at most $\sqrt{n \log(n)}$ w.h.p.
- Remove assumptions: (1) $x_i = 100\sqrt{n \log(n)}$ and (2) and $x_j \in \{0,1\}$ for $j \in \{1, 2, \ldots, n\}$ with $j \neq i$

CountSketch achieves the l_2–guarantee

- Assign each coordinate i a random sign $\sigma_i \in \{-1,1\}$
- Randomly partition coordinates into B buckets, maintain $c_j = \Sigma_{i:\, h(i) = j} x_i \cdot \sigma_i$ in the j-th bucket



- Estimate x_i as $\sigma_i \cdot c_{h(i)}$

Why Does CountSketch Work?

- $\mathrm{E}\left[\sigma_i c_{h(i)}\right] = \mathrm{E}[\sigma_i \sum_{i':h(i)=h(i')} \sigma_{i'} x_{i'}] = x_i$
- Suppose we independently repeat this hashing scheme O(log n) times
- Output the median of the estimates across the log n repetitions
- "Noise" in a bucket is $\sigma_i \cdot \sum_{i' \neq i, h(i')=h(i)} \sigma_{i'} \cdot x_{i'}$
- What is the variance of the noise?
- $\mathrm{E}\left[\left(\sigma_i \cdot \sum_{i' \neq i, h(i')=h(i)} \sigma_{i'} \cdot x_{i'}\right)^2\right] \leq \frac{|x|_2^2}{B}$
- So with constant probability, the noise in a bucket is $O(\frac{|x|_2}{\sqrt{B}})$ in magnitude
- Since the log n repetitions are independent, this ensures that our estimate $\sigma_i c_{h(i)}$ will equal $x_i \pm O(\frac{|x|_2}{\sqrt{B}})$ with probability 1-1/poly(n)
- Hence, we approximate *every* x_i simultaneously up to additive error $O(\frac{|x|_2}{\sqrt{B}})$

Tail Guarantee

- CountSketch approximates every x_i simultaneously up to additive error $O(\frac{|x|_2}{\sqrt{B}})$
- But what if x_1 is a super large poly(n), and $x_2 = n$ and $x_3 = \ldots = x_n = 1$?
- We get a pretty bad approximation to x_2
- Tail Guarantee: CountSketch approximates *every* x_i simultaneously up to additive error $O(\frac{|x_{-B/4}|_2}{\sqrt{B}})$, where $x_{-B/4}$ is x after zero-ing out its top B/4 coordinates in magnitude
- Proof: with probability at least $3/4$, in each repetition the top B/4 coordinates of x in magnitude do not land in the same hash bucket as x_i
  - Do we need a lot of independence for this?
- What happens if x is B/4-sparse?

Why Care About the ℓ_1-Guarantee?

- l_1 – guarantee
  - output a set containing all items j for which $|x_j| \geq \phi |x|_1$
  - the set should not contain any j with $|x_j| \leq (\phi\text{-}\varepsilon)\ |x|_1$

- l_2 – guarantee
  - output a set containing all items j for which $x_j{}^2 \geq \phi |x|_2^2$
  - the set should not contain any j with $x_j^2 \leq (\phi - \epsilon)|x|_2^2$

- l_2 – guarantee implies the l_1 – guarantee

- So why care about the l_1 – guarantee?

- A nice thing about the l_1-guarantee is that it can be solved deterministically!

Deterministic ℓ_1 Heavy Hitters

- An s x n matrix S is ϵ-incoherent if
  - for all columns S_i, $|S_i|_2 = 1$
  - for all pairs of columns S_i and S_j, $\left|\langle S_i, S_j\rangle\right| \leq \epsilon$
  - entries can be specified with O(log n) bits of space

- Compute $S \cdot x$ in a stream using O(s log n) bits of space

- Estimate $\widehat{x_i} = S_i^T Sx$
  - $\widehat{x_i} = \sum_{j=1,\ldots,n}\langle S_i, S_j\rangle x_j = |S_i|_2^2 x_i \pm \max_{i,j}\left|\langle S_i, S_j\rangle\right||x|_1 = x_i \pm \epsilon|x|_1$
  - Can figure out which $|x_i| \geq \phi|x|_1$ and which $|x_i| \leq (\phi - \epsilon)|x|_1$

- But do ϵ-incoherent matrices exist?

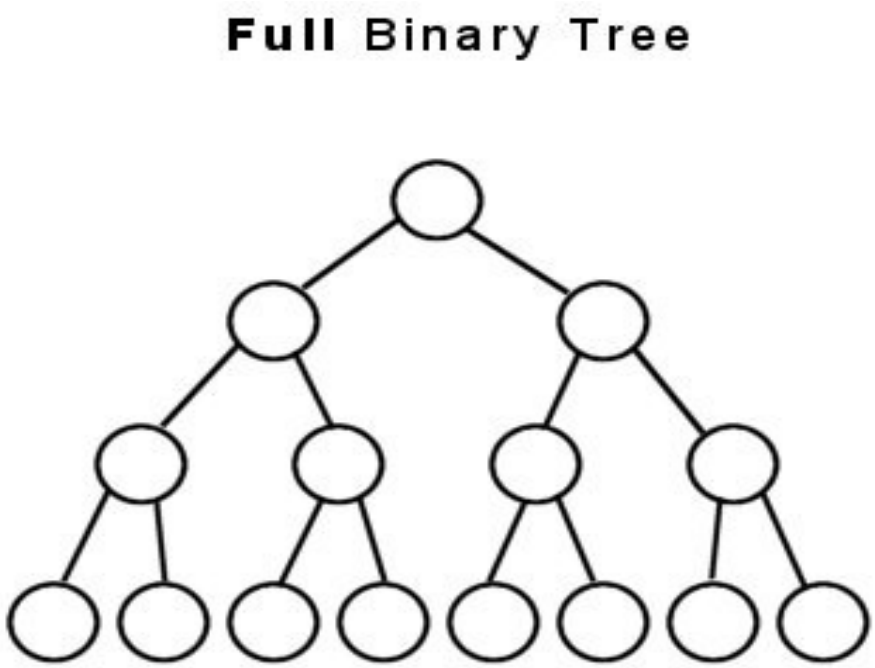
ϵ-Incoherent Matrices

- Consider a prime q = $\Theta((\log n)/\epsilon)$. Let d = $\epsilon \cdot q$ =O(log n)
- Consider n distinct non-zero polynomials $p_1, \ldots, p_n$ each of degree less than d.
  - $q^d - 1 > n$
- Associate p_i with i-th column of S
- Let s = q^2 and group the rows of S into q groups of size q
  - In j-th group, the i-th column has a single non-zero on the $p_i(j)$-th entry
  - $p_{i(j)}$-th entry is equal to $1/q^{1/2}$
- Each column S_i has $|S_i|_2 = 1$
- S_i and S_j each have the same non-zero in the k-th group iff $p_i(k) = p_j(k)$
- Number of such groups k is at most $d \leq \epsilon q$, so $|\langle S_i, S_j \rangle| \leq \epsilon$

How to Find the Top k Heavy Hitters Quickly

- There are 2^i nodes in i-th level of tree
  - Start at the level with 2k nodes
- Each node corresponds to a subset of [n] of size $n/2^i$ with the same i-bit prefix
- In i-th level, for each i, hash to O(k) buckets repeat O(log n) times. Like CountSketch, but in each bucket we run an approximation algorithm to the 2-norm
- In top level our universe has only 2k nodes, so we find top k just by computing estimate for all of them

Main idea: in next level, we only need to consider the left and right child of each of the k nodes we found at the previous level. So only $2k \ll n$ nodes to consider.



Outline

- Quick recap of ℓ_1-regression, and how to speed it up
- Introduction to the Streaming Model
- Estimating Norms in the Streaming Model
- Heavy Hitters in a Stream
- Estimating Number of Non-Zero Entries (ℓ_0)

Estimating the Number of Non-Zero Entries

- $|x|_0 = |\{i \text{ such that } x_i \neq 0\}|$
- How can we output a number Z with $(1-\epsilon)Z \leq |x|_0 \leq (1+\epsilon)Z$ with prob. 9/10?
  - Want $O((\log n)/\epsilon^2)$ bits of space
- Suppose $|x|_0 = O(\frac{1}{\epsilon^2})$. What can we do in this case?
- Use our algorithm for recovering a k-sparse vector from last time, $k = O\left(\frac{1}{\epsilon^2}\right)$
  - What is another way?
- But what if $|x|_0 \gg \frac{1}{\epsilon^2}$?

Estimating the Number of Non-Zero Entries

- Suppose we somehow had an estimate Z with $Z \leq |x|_0 \leq 2Z$, what could we do?
- Independently sample each coordinate i with probability $p = 100/(Z\,\epsilon^2)$
- Let Y_i be an indicator random variable if coordinate i is sampled
- Let y be the vector restricted to coordinates i for which $Y_i = 1$
- $E[|y|_0] = \sum_{i \text{ such that } x_i \neq 0} E[Y_i] = p|x|_0 \geq \frac{100}{\epsilon^2}$
- $Var[|y|_0] = \sum_{i \text{ such that } x_i \neq 0} Var[Y_i] \leq \frac{200}{\epsilon^2}$
- $Pr\left[\left||y|_0 - E[|y|_0]\right| > \frac{100}{\epsilon}\right] \leq \frac{Var[|y|_0]\epsilon^2}{100^2} \leq \frac{1}{50}$
- Use sparse recovery or CountSketch to compute $|y|_0$ exactly
- Output $\frac{|y|_0}{p}$

But we don't know Z…

Estimating the Number of Non-Zero Entries

- Guess Z in powers of 2
- Since $0 \leq |x|_0 \leq n$, there are $O(\log n)$ guesses
- The i-th guess Z = 2^i corresponds to sampling each coordinate with probability $p = \min(1, \frac{100}{2^i \epsilon^2})$
- Sample the coordinates as nested subsets $[n] = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{\log}$
- Run previous algorithm for each guess
- One of our guesses Z satisfies $Z \leq |x|_0 \leq 2Z$ and we should use that guess

- *But how do we know which one?*

Estimating the Number of Non-Zero Entries

- Use the largest guess $Z = 2^i$ for which $\frac{400}{\epsilon^2} \leq |y|_0 \leq \frac{3200}{\epsilon^2}$
- If $\frac{800}{\epsilon^2} \leq E[|y|_0] \leq \frac{1600}{\epsilon^2}$, then $\frac{400}{\epsilon^2} \leq |y|_0 \leq \frac{3200}{\epsilon^2}$ with probability at least 49/50
- If $\frac{100}{\epsilon^2} \leq E[|y|_0] \leq \frac{200}{\epsilon^2}$, then $|y|_0 < \frac{400}{\epsilon^2}$ with probability at least 49/50
- So with probability 98/100, we choose an i for which $\frac{200}{\epsilon^2} \leq E[|y|_0] \leq \frac{1600}{\epsilon^2}$
- There are only 4 such indices i, and all 4 of them satisfy $|y|_0 = (1 \pm \epsilon)E[|y|_0]$ simultaneously with probability 1-4/50. So doesn't matter which i we choose

- Overall, our success probability is 1-2/50-4/50 > 4/5

What is Our Overall Space Complexity?

- If we use our k-sparse recovery algorithm for $k = O\left(\frac{1}{\epsilon^2}\right)$, then it takes $O(\frac{\log n}{\epsilon^2})$ bits of space in each of log n levels, so $O(\frac{\log^2 n}{\epsilon^2})$ total bits of space ignoring random bits
  - How much randomness do we need?
  - Pairwise independence is enough for Chebyshev's inequality
  - Implement nested sampling by choosing a hash function $h: [n] \to [n]$, checking if first i bits of h(j) = 0
  - O(log n) bits of space for the randomness

- Can improve to $O\left(\frac{\log n\ (\log\left(\frac{1}{\epsilon}\right)+\log\log n)}{\epsilon^2}\right)$ bits. How?

- Just need to know number of non-zero counters, so reduce counters from log n bits to $O(\log\left(\frac{1}{\epsilon}\right) + \log\log n)$ bits

Reducing Counter Size

- In sampling levels that we care about, we have $O\left(\frac{1}{\epsilon^2}\right)$ counters, each of O(log n) bits
- At most $O\left(\frac{\log n}{\epsilon^2}\right)$ prime numbers dividing any of these counters
- Choose a random prime q = $O\left(\frac{\log n \log\log}{\epsilon^2}\right)$. Unlikely that q divides any counters
- Just maintain our sparse recovery structure mod q, so $O\left(\frac{(\log\log n + \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2}\right)$ bits per each of O(log n) sparse recovery instances