CS 15-859: Algorithms for Big Data	Fall 2020
Lecture 9 — 11/5/2020	
Prof. David Woodruff	*Scribe: Misha Khodak*

1 Recap: Estimating p-norms in a data stream

Recall that our goal is to estimate $\|y\|_p$ in a data stream up to a constant factor. We have sketching matrices P and D, where D is a diagonal matrix such that $D_{ii} \sim 1/E_i^{\frac{1}{p}}$, E_i ∼Exponential, and P is CountSketch, with sketching dimension $s = n^{1-2/p} \log n$. We showed that $\|Dy\|_\infty = \Theta(\|y\|_p)$. Suppose $(Dy)_j = \|Dy\|_\infty$, i.e. j is the coordinate realizing the maximum. This coordinate will be hashed to a bucket by CountSketch, and our estimator uses the max bucket value; we thus needed to show that the noise in the buckets is fairly small. To do this we needed to separate the coordinate j' of Dy into heavy ($(Dy)_j > \alpha\|y\|_p/\log n$) and light (else). These heavy coordinates do not concentrate, but our analysis showed that the number of heavy items is $O(\log^p n)$ w.p. 9/10, and so they would likely go to separate buckets, i.e. they would be perfectly hashed. For the remaining (non-heavy) items we were able to bound the bucket noise in all buckets w.h.p. to be less than $\|y\|_p/100$ using Bernstein's inequality.

2 This lecture: Heavy hitters in a stream

We have a problem in which we want to find items that occur very frequently in a stream, e.g. some destination in the web receiving heavy traffic. There are several desired guarantees; for example, the **ℓ_1-guarantee** asks for a set containing all items j for which $|x_j| \geq \phi\|x\|_1$, where $\phi \in (0, 1)$, and that the same set does not contain any item j with $|x_j| \leq (\phi - \varepsilon)\|x\|_1$ for $\varepsilon \in (0, \phi)$. This relaxation is required because there is a lower-bound via reduction to the indexing problem, in which we have a stream in which whether or not an item is a heavy hitter depends on whether it occurred in a long span of the stream.

A different guarantee is the **ℓ_2-guarantee**, where we desire a set containing all items j such that $x_j^2 \geq \phi\|x\|_2^2$, with a similar slack allowance that there should be no element with $x_j^2 \leq (\phi - \varepsilon)\|x\|_2^2$. Notably, the ℓ_2-guarantee can be much stronger than the ℓ_1-guarantee. For example, if we have a vector of dimension n with one value of $\sqrt{n}$ and $n - 1$ values of ones, this vector will require $\phi \approx 1/\sqrt{n}$ to be recovered using the ℓ_1-guarantee, which is a problem since the algorithms we show will have memory $1/\phi$. However, in the ℓ_2-guarantee we only require $\phi = 1/2$. More formally, if we have a ℓ_1-heavy-hitter, i.e. $|x_j| \geq \phi\|x\|_1$, then squaring both sides yields

$$x_j^2 \geq \phi^2\|x\|_1^2 \geq \phi^2\|x\|_2^2$$

i.e. the item is also an ℓ_2-heavy-hitter for ϕ^2. In principle we could go beyond to ℓ_p for $p > 2$, but as we will see the memory will grow as $n^{1-2/p}$.

2.1 Intuition via special cases

Suppose we are promised that at the end of a stream we have some coordinate $x_i = n$ and $x_j \in \{0, 1\}$ for $i \neq j$. For each $j \in [\log n]$ let $A_j \subset [n]$ be the set of indices with the jth bit in their binary representation being 0 and $B_j \subset [n]$ be the set where it is 1. We can compute $a_j = \sum_{i \in A_j} x_i$ and $b_j = \sum_{i \in B_j} x_i$ for each $j \in [\log n]$, which requires $\log n$ counters with $\log n$ bits each, i.e. total memory $\log^2 n$. Then we recover the index i by returning the number with binary representation with 1 if $a_j > b_j$ and 0 otherwise.

Now consider a different stream in which we are promised $x_i = 100\sqrt{n \log n}$ and $x_j \in \{0, 1\}$ for $j \neq i$. The previous approach does not work here since the value is too small. However, we can conduct a similar sketch: compute $a_j = \sum_{i \in A_j} \sigma_i x_i$ and $b_j = \sum_{i \in B_j} \sigma_i x_i$ for each $j \in [\log n]$ for $\sigma_i \sim$Uniform$\{\pm 1\}$, which requires $\log n$ counters with $\log n$ bits each, i.e. total memory $\log^2 n$. Using Chernoff bounds we have that the noise in the count is bounded by $\sqrt{n \log n}$ w.h.p., and so we can again recover the index i by returning the number with the binary representation with 1 if $|a_j| > |b_j|$ and 0 otherwise.

2.2 Achieving the ℓ_2-guarantee

Assign each coordinate i a random sign $\sigma_i \in \{-1, 1\}$. Randomly partition the coordinates into B buckets, maintaining a counter $c_j = \sum_{i: h(i) = j} x_i \sigma_i$ in the jth bucket, where $h : [n] \mapsto [B]$ is a hash function. We can then estimate x_i as $\sigma_i c_{h(i)}$, which has expectation

$$\mathbb{E}(\sigma_i c_{h(i)}) = \mathbb{E}\left(\sigma_i \sum_{i': h(i) = h(i')} \sigma_{i'} x_{i'}\right) = \mathbb{E}\left(\sigma_i^2 x_i + \sum_{i': h(i) = h(i'), i' \neq i} \sigma_{i'} x_{i'}\right) = x_i \tag{1}$$

Suppose we independently repeat this hashing scheme $O(\log n)$ times and output the median of the estimates across the $\log n$ repetitions. We have noise variance in each bucket

$$\mathbb{E}\left(\sigma_i \sum_{i': h(i) = h(i'), i' \neq i} \sigma_{i'} x_{i'}\right)^2 \leq \|x\|_2^2 / B \tag{2}$$

Then by Chebyshev's Inequality we have for a single bucket that

$$P\left(|\sigma_i c_{h(i)} - x_i| \geq \frac{10\|x\|_2}{\sqrt{B}}\right) \leq 1/100 \tag{3}$$

i.e. the noise is a bucket is $O(\|x\|_2/\sqrt{B})$ with constant probability. By independence of the different hashing schemes, since we have $\log n$ of them the median of the $\sigma_i c_{h(i)}$ across them will be $x_i \pm O(\|x\|_2/\sqrt{B})$ w.p. $1 - 1/poly(n)$. Thus we approximate every x_i simultaneously with error $O(\|x\|_2/\sqrt{B})$, so using $B = O(1/\phi)$ buckets we can distinguish the cases $|x_i| \leq \sqrt{\phi/2}\|x\|_2$ and $|x_i| > \sqrt{\phi}\|x\|_2$, solving the ℓ_2-heavy-hitters problem. Note that the memory used is $B \log^2 n$ since we repeat B buckets $\log n$ times and each bucket is a counter requiring $\log n$ bits.

While the above guarantee approximates all x_i simultaneously with additive error $O(\|x\|_2/\sqrt{B})$, this can be a problem if our stream has $x_1 = poly(n), x_2 = n, x_3 = \cdots = x_n = 1$ and we wish to recover both x_1 and x_2. This is because the resulting norm of $\|x\|_2$ is $poly(n)$ and so n gets poorly approximated. However, we can use the same data structure above to get error bounded by $O(\|x_{-B/4}\|_2/\sqrt{B})$, where $x_{-B/4}$ is the vector x with its top $B/4$ coordinates set to 0. This is because w.p. 3/4, in each bucket repetition the top $B/4$ coordinates of x do not land in the same hash bucket as x_i. Note that this result requires only pairwise independent hash functions, which can be specified with $O(\log n)$ bits. Furthermore, if x is $B/4$-sparse this result implies that all x_i are estimated perfectly.

2.3 The ℓ_1-guarantee

As discussed before, the ℓ_2-guarantee implies the ℓ_1-guarantee; however, we can attain ℓ_1-guarantees deterministically, unlike our CountSketch-based data structure above. To do so, we consider matrices S satisfying the following quality:

Definition. An $s \times n$ matrix S is **ε-incoherent** if all columns S_i have norm $\|S_i\|_2 = 1$ and any pair of columns has dot product $|\langle S_i, S_j\rangle| < \varepsilon$.

We further want all entries of S to be specified with $O(\log n)$ bits. Then we consider an algorithm that maintains Sx in a stream using $O(s \log n)$ bits of space and outputs an estimate $\hat{x}_i = S_i^T Sx$. To see that this works, note that by ε-incoherence we have

$$\hat{x}_i = \sum_{j=1}^{n} \langle S_i, S_j\rangle x_j = \|S_i\|_2^2 x_i \pm \max_{i,j} |\langle S_i, S_j\rangle| \|x\|_1 = x_i \pm \varepsilon\|x\|_1 \tag{4}$$

This solves the ℓ_1-heavy-hitters problem by finding $|x_i| \geq \phi\|x\|_1$.

We now turn to constructing ε-incoherent matrices. Consider a prime $q = \Theta(\frac{1}{\varepsilon}\log n)$ and $d = \varepsilon q = \Theta(\log n)$. Let $p_1, \ldots, p_n$ be n distinct non-zero polynomials $p_i = \sum_{j=0}^{d-1} a_{i,j} x^j \mod q$, for each of which there are $q^d - 1$ possibilities. Choosing q such that $q^d - 1 > n$, associate each polynomial to a column of S and set its row dimension to be $s = q^2$. Partition S into q groups of q rows and populate S by setting the $p_i(j)$th entry of the ith column in the jth group to have entry $1/\sqrt{q}$, with all other entries being 0. Each column S_i thus has $\|S_i\|_2 = 1$, and the dot product between S_i and $S_{i'}$ is the number of entries j such that $p_i(j) = p_{i'}(j)$ divided by $1/q$. Since $p_i - p_{i'}$ is a polynomial of degree at most $d-1$, it has at most $d-1$ zeros and so the dot product is bounded by $\frac{d-1}{q}$; since $d = \varepsilon q$ we have that each dot product is bounded by ε. Thus we can achieve an ε-incoherent matrix with $O(\frac{1}{\varepsilon^2}\log^2 n)$ rows.

2.4 Finding heavy-hitters quickly

The previous algorithms have been constructed by getting estimates of all x_i and cycling through them to determine heavy-hitters; this is memory-efficient but requires $O(n)$ computation after the stream concludes. We can accelerate this using a binary-tree-like data structure, in which we partition the coordinates i into $2k$ groups of size $\frac{n}{2k}$. Using a CountSketch matrix S at each level and setting y^i to be the vector whose entries are those of the ith of $2k$ groups, we can hash to each $j \in O(k)$ buckets the values $S\sum_{h(i)=j} y^i$. The norm of the result will have error $O(\|x\|_2/\sqrt{k})$, so at each level of the binary tree can have at most k heavy hitters. Thus starting from the top of the tree, we determine the groups containing the heavy hitters in each level, which takes $O(k)$ time, and there are $\log n$ levels, so to find the final bucket containing the heavy hitters takes $O(k \log n)$ time. This sketch takes $\log^4 n$ memory due to $\log n$ storage of CountSketch sketches in this binary tree.