CS 368 (Stanford, Spring 2022) Homework 2

Due: **May 1, 2022, 11:59pm**

Policy: You are permitted to discuss and collaborate on the homework but you must write up your solutions on your own or as a group of two. Furthermore, you need to cite your collaborators and/or any sources that you consulted. No late submissions are allowed. There will be no late days. All homework submissions are subject to the Stanford Honor Code. For all assignments, we are allowing group submissions for groups of 1 or 2.
Submission: We will use Gradescope for homework submissions. You must typeset your write-up (we recommend LaTeX via Overleaf). If you are working as a group of two, only one group member needs to submit the assignment. When submitting, please remember to add all group member names on Gradescope.
Length of submissions: Please include as much of the calculations that show that you understand everything that is going through the answer. As a rule of thumb after you have solved the problem, try to identify what are the main steps taken and critical points of a proof and include them. Unnecessary long answers to questions will be penalized. The points next to each question are indicative of the hardness/length of the proof.

1 Sketching for Faster Updates [40 points]

The AMS sketch from class for F_2 moment estimation can be thought of as picking a random $m \times n$ matrix A with entries $\pm 1/\sqrt{m}$ for $m = O(\epsilon^{-2})$, and estimating $\|f\|_2^2$ as $\|Af\|_2^2$. One can show that with at least 2/3 probability,

$$(1-\epsilon)\|f\|_2^2 \leq \|Af\|_2^2 \leq (1+\epsilon)\|f\|_2^2 \tag{1}$$

Here we're not doing the full median-of-means estimate. Instead, we merely compute the mean of m independent estimates, so we don't drive the error probability down.

In this problem you will explore a different way of estimating F_2, inspired by Count-Sketch. Imagine picking $A \in \{\pm 1, 0\}^{m \times n}$ differently, for each $i \in \{1, \ldots, n\}$ we pick a row $h_i \in [m]$ uniformly at random and set $A_{h_i,i} = \pm 1$ (the sign is chosen uniformly at random from $\{-1, 1\}$), and all other entries of the i-th column are set to 0. This A has the advantage that in turnstile streams (where each update vector Δx can contain a non-zero integer only at one location) we can process updates in constant time, independent of ϵ. Show that using this A still satisfies the conditions of equation (1) with 2/3 probability for $m = O(\epsilon^{-2})$.

2 Approximate Frequency Estimation [30 points]

Consider the following algorithm. We prove an error guarantee in the following parts. It might be helpful to look at how the counter associated with an element changes as the element gets added to and removed from the bins.

Algorithm 1

Input: stream $i_1, i_2, \ldots, i_m \in [n]$, number of bins k.
initialize each bin $b \in [k]$ with an element $e_b \leftarrow \emptyset$ (initially null) and a counter $c_b \leftarrow 0$.
for each element i_ℓ in the stream **do**
  if i_ℓ is in a bin b **then**
    increment b's counter $c_b \leftarrow c_b + 1$
  else
    find the bucket b_ℓ with the smallest counter value (breaking ties arbitrarily)
    replace the current element $e_{b_\ell} \leftarrow i_\ell$
    increment its counter $c_{b_\ell} \leftarrow c_{b_\ell} + 1$.
Output: for each $i \in [n]$ output $\hat{f}_i = c_b$ if $e_b = i$ and 0 otherwise.

(a) **[15 points]** Consider an element i with $\hat{f}_i = 0$. Show that the true frequency f_i is such that $0 \leq f_i < \frac{m}{k}$. This would imply $|\hat{f}_i - f_i| < \frac{m}{k}$.

(b) **[15 points]** Consider an element i with $\hat{f}_i > 0$. Show that $|\hat{f}_i - f_i| < \frac{m}{k}$.

3 Reducing Randomness via Nisan's Generator[1] [15 points]

Note: *The description of this problem is long, but what you actually need to do is very little compared to the length of the description. This is mostly designed to supplement the in-class discussion of the use of Nisan's pseudorandom generator in the context of streaming.*

We have seen a collection of algorithms for estimating the ℓ_p norm of the n-dimensional vector x induced by the stream, for $p \in (0, 2]$. The idea was to calculate a "linear sketch" $\Pi x = [Z_1 \ldots Z_k]$, where Π was an $k \times n$ random matrix, with i.i.d. entries r_{ij} selected from a p-stable distribution. After calculating Πx, the algorithm outputs

$$\text{median}[|Z_1|, \cdots, |Z_k|]/C(p)$$

as an estimator of $\|x\|_p$, where $C(p)$ denotes some scaling factor that depends only on p.

For the purpose of this problem, we will focus on the decision version of the algorithm, which checks whether

$$\text{median}[|Z_1|, \cdots, |Z_k|]/C(p) \quad \geq \quad T \tag{2}$$

for some threshold T. We assume that $p = 2$, in which case the entries of Π can be selected from Gaussian distribution $\mathcal{N}(0, 1)$. We also assume that the entries of x always remain integers from $\{-M \ldots M\}$ for some $M = n^{O(1)}$, i.e., they have values polynomial in the dimension n.

There are two issues regarding the space requirement:

[1] Thanks to Piotr Indyk and Jelani Nelson for this question.

- Discretization: given that the algorithm space is measured in bits, we need to make sure that each r_{ij} has bounded precision. Dealing with this issue is straightforward, as we can modify the random variables so that their values fall into an interval $[-c\sqrt{\log n} \ldots c\sqrt{\log n}]$, and are multiples of of $1/n^c$, for some $c = O(1)$. The analysis of the modified algorithm remains essentially unchanged, modulo minor increase in the approximation error and failure probability. In what follows we assume that r_{ij} are already generated in this way, and therefore need only $b = O(\log n)$ bits of representation.
- Pseudo-randomness: even if $r'_{ij}s$ are discrete, we cannot afford to store all of them in memory, as this would require knb bits of storage. Instead, they can be generated "on the fly" using a pseudorandom generator, i.e., there is an efficiently computable mapping $G : \{0,1\}^L \rightarrow \{\{0,1\}^b\}^{nk}$ such that we can set $r_{ij} = G(v)_{ij}$, where v is a "random seed" selected from $\{0,1\}^L$ uniformly at random. Formalizing and optimizing this step is the focus of this problem.

We will use the pseudo-random generator for bounded space due to Nisan [1]. Consider a class of (S, b)-*automata* Q, that have 2^S states and read sequences of symbols from $\{0,1\}^b$, i.e., operate over an alphabet of size 2^b. Such automata are defined by:

- A transition function $Q(s, a)$, which describe the state the automaton moves to from state s after reading a,
- An initial state $start$, and
- A set of accepting states Acc.

Such automata can model any deterministic computation device that processes a sequence u of symbols from $\{0,1\}^b$ in space S. We use $Q(u)$ to denote the state reached by the automaton after reading u, starting from $start$.

Nisan's generator G has the following wonderful properties. Suppose that the automaton is applied to sequences of length R. Then:

- The seed length L of G is equal to $O(S \log R)$, assuming $b = O(S)$.
- It ϵ-*fools* any (S, b)-automaton Q , i.e.,

$$|\Pr_{u \in (\{0,1\}^b)^R}[Q(u) \in Acc] - \Pr_{v \in \{0,1\}^L}[Q(G(v)) \in Acc]| \leq \epsilon$$

  for $\epsilon = 2^{-\Omega(S)}$.

Note that in the above definition, the input to Q consists of (pseudo)-random bits, which are "tested" by Q. Nisan's generator is designed to ϵ-fool all such tests, despite generating randomness from a relatively small truly random seed.

To use Nisan's generator in our streaming algorithm, we need to model the algorithm as a finite automaton reading the random entries of the matrix Π and producing some decision in the end. Then we use the properties of the generator to argue that replacing truly random Π by a pseudorandomly generated version does not (significantly) alter the behavior of the algorithm.

The computation specified in Equation (2) can be performed by an automaton Q that reads the entries $r_{1,1}, \ldots, r_{1,n}, r_{2,1}, \ldots, r_{2,n}, \ldots r_{k,n}$ of Π (i.e., in the row-wise order), computes the vector $[Z_1, \cdots, Z_k]$, evaluates the median and accepts if the result is at least T. (Note that Q is *parameterized* by the vector x, i.e., x is *not* an input to Q!).

Finally, we are ready to state the problems:

(a) **[5 points]** Observe that Q can be implemented so that $S = O(k \log n)$. Calculate the length of the seed L required to $2^{-\Omega(S)}$-fool such automata Q.

(b) **[10 points]** Show a better implementation of Q that requires only $S = O(\log k + \log n)$. Calculate the length of the seed L required to $2^{-\Omega(S)}$-fool such automata Q.

4 Sparse Recovery using Count-Min Sketches [40 points]

In the sparse recovery problem, we want to find a k-sparse vector $\hat{x}$ (i.e., having at most k nonzero entries) that minimizes the error $\|x - \hat{x}\|_q$ given the linear sketch Ax. In this problem, we focus on minimizing the ℓ_1 error $\|x - \hat{x}\|_1$. We assume A is chosen at random from some distribution specified below (inspired by the Count-Min sketch) and show that some recovery algorithms work with high probability. Recall $\|x\|_q = (\sum_i |x_i|^q)^{1/q}$ for a vector x.

We consider matrix A generated in the following way. Let w be a parameter (specified later) and H be the set of all hash functions $h : \{1, \ldots, n\} \to \{1, \ldots, w\}$. For each hash function $h \in H$, let $A(h)$ denote the $w \times n$ matrix with 0/1 entries where $(A(h))_{ji}$ is equal to 1 if $j = h(i)$ and 0 if otherwise. For d hash functions $h_1, \ldots, h_d$ chosen independently and uniformly at random from H, we define A to be a vertical concatenation of $A(h_1), \ldots, A(h_d)$. The number of rows in A is equal to $m = wd$ and the number of columns is equal to n. In this problem, we ignore the issue of representing the hash functions (and hence the matrix A) in small space. This can be fixed in standard ways.

Intuitively speaking, for a fixed value of i and hash function h_l, the coordinate of the sketch $(Ax)_{(l-1)w+h_l(i)}$ is equal to the sum $\sum_{t:h_l(t)=h_l(i)} x_t$ which is the sum of x_i and some contributions from other x_t's. We want to aggregate these coordinates over different h_l to obtain an approximation of x_i.

Given a vector x, we define $\mathrm{Err}_q^k(x)$ to be the smallest ℓ_q approximation error $\min_{k\text{-sparse } x'} \|x - x'\|_q$ where x' ranges over all k-sparse vectors, i.e., those having at most k nonzero entries. Note for any value of q, $\|x - \hat{x}\|_q$ is minimized when $\hat{x}$ consists of the k largest (in magnitude) coordinates of x; that is, $\hat{x}_i$ equals x_i for these k largest coordinates and 0 for other coordinates. The smallest possible error for the sparse recovery problem is $\mathrm{Err}_q^k(x)$.

In what follows, assume $\epsilon \in (0, 1)$. For Parts (a) and (b), assume $x \geq 0$, that is, all the coordinates are nonnegative. Consider the Count-Min algorithm that computes the approximation x^* where $x_i^* = \min_l (A(h_l)x)_{h_l(i)}$. Let $i_1, i_2, \ldots$ be the ordering of $[n]$ such that $|x_{i_1}| \geq \cdots \geq |x_{i_n}|$. The k largest (in magnitude) coordinates are $x_{i_1}, \ldots, x_{i_k}$.

(a) **[10 points]** For any i, argue that $\Pr\left((A(h_l)x)_{h_l(i)} - x_i \geq \frac{\epsilon}{k}\operatorname{Err}_1^k\right) \leq \Pr\left(h_l(i) \in h_l(\{i_1, \ldots, i_k\} \setminus \{i\})\right) + \Pr\left(\sum_{r>k:h_l(i_r)=h_l(i), i_r \neq i} x_{i_r} \geq \frac{\epsilon}{k}\operatorname{Err}_1^k\right)$. Note $h_l(\{i_1, \ldots, i_k\} \setminus \{i\})\big)$ is the set of hash values for elements in the set $\{i_1, \ldots, i_k\} \setminus \{i\}$. For $w = \frac{4k}{\epsilon}$, show that $\Pr\left((A(h_l)x)_{h_l(i)} - x_i \geq \frac{\epsilon}{k}\operatorname{Err}_1^k\right) \leq \frac{1}{2}$. *Hint: the Markov's inequality might be useful.*

(b) **[5 points]** Given Part a, show that $\Pr[x_i^* - x_i \geq \frac{\epsilon}{k}\operatorname{Err}_1^k] \leq \frac{1}{2^d}$. For an appropriately chosen $d = O(\log n)$, show that $\|x^* - x\|_\infty \leq \frac{\epsilon}{k}\operatorname{Err}_1^k$ with high probability (i.e., with a failure probability of the form $\frac{1}{n^c}$ for some constant c).

For Parts (c) and (d), assume general x (so, a coordinate can be negative). The count-min algorithm and the above analysis would not work because the x_i's can be negative.

(c) **[15 points]** For general x, design and analyze a different approximation scheme with $d = O(\log n)$ that given Ax, returns an approximation x^* such that $\|x^* - x\|_\infty \leq \frac{\epsilon}{k}\operatorname{Err}_1^k$ with high probability. *Hint: You do not want to use* min *to aggregate. You can adapt the same line of reasoning outlined in Parts a and b with some changes. It might be useful to show for an appropriately chosen w, $\Pr\left(|(A(h_l)x)_{h_l(i)} - x_i| \geq \frac{\epsilon}{k}\operatorname{Err}_1^k\right)$ is at most some constant strictly less than $\frac{1}{2}$. The Chernoff bound might be useful.*

Part (c) implies that for any x, given Ax, we can recover x^* such that $\|x^* - x\|_\infty \leq \frac{\epsilon}{k}\operatorname{Err}_1^k$ with high probability. Given this, we can solve the sparse recovery problem.

(d) **[10 points]** Let $\hat{x}$ be consisting of the k largest (in magnitude) coordinates of the recovered x^* (and 0's elsewhere) satisfying $\|x^* - x\|_\infty \leq \frac{\epsilon}{k}\operatorname{Err}_1^k$. Show that $\|x - \hat{x}\|_1 \leq (1 + 3\epsilon)\operatorname{Err}_1^k$. *Hint: It might be helpful to use that $\|\hat{x}_S\|_1 \leq \|\hat{x}_{\hat{S}}\|_1$ where S is the set of the k largest (in magnitude) coordinates of x and $\hat{S}$ is the support of $\hat{x}$.*

References

[1] Noam Nisan. Pseudorandom Generators for Space-Bounded Computation. *Combinatorica*, 12(4):449-461, 1992.