Problem Set 5 CS265, Autumn 2022

Due: Friday 11/4 at 11:59 pm on Gradescope

Group members: Rylan Schaeffer, Connor Settle

Please follow the homework policies on the course website.

---

1. **(14 pt.)** **[Another way to sketch sparse vectors.]** Suppose that A is an list of length n, containing elements from a large universe $\mathcal{U}$. Our goal is to estimate the frequencies of each element in $\mathcal{U}$: that is, for $x \in \mathcal{U}$, how often does x appear in A?

   The catch is that A is too big to look at all at once. Instead, we see the elements of A one at a time: $A[0], A[1], A[2], \ldots$. Unfortunately, $\mathcal{U}$ is also really big, so we can't just keep a count of how often we see each element.

   In this problem, we'll see a construction of a randomized data structure that will keep a "sketch" of the list A, use small space, and will be able to efficiently answer queries of the form "approximately how often did x occur in A"?

   Specifically, our goal is the following: we would like a (small-space) data structure, which supports operations `update`(x) and `count`(x). The `update` function inserts an item $x \in \mathcal{U}$ into the data structure. The `count` function should have the following guarantee, for some $\delta, \epsilon > 0$. After calling `update` n times, `count`(x) should satisfy

$$C_x \leq \texttt{count}(x) \leq C_x + \epsilon n \tag{1}$$

   with probability at least $1 - \delta$, where C_x is the true count of x in A.

   (a) **(3 pt.)** Your friend suggests the following strategy (this will not be our final strategy). We start with an array R of length b initialized to 0, and a random hash function $h : \mathcal{U} \to \{0, 1, \ldots, b-1\}$. You can assume that h is drawn from some universal hash family, i.e $P(h(x) = h(y)) = 1/b$ for any $x \neq y$. Then the operations are:

   - `update`(x): Increment $R[h(x)]$ by 1.
   - `count`(x): return $R[h(x)]$.

   For every entry $A[i]$ in the list it encounters, the scheme calls `update`$(A[i])$.

   After sequentially processing all n items in the list, what is the expected value of `count`(x)?

   (b) **(2 pt.)** Show that there is a choice of b that is $O(1/\epsilon)$ so that, for any fixed $x \in \mathcal{U}$, we have

$$\Pr[\texttt{count}(x) < C_x] = 0$$

   and

$$\Pr[\texttt{count}(x) \geq C_x + \epsilon n] \leq \frac{1}{e}.$$

   [**HINT:** *The first of the requirements is true no matter what b is.*]

   (c) **(2 pt.)** Explain how you would use T copies of the construction in part (a) to define a data structure that, for any fixed $x \in \mathcal{U}$, satisfies (1) with high probability. How big do you need to take T so that the (1) is satisfied with probability at least $1-\delta$? How much space does your modified construction use? (It should be sublinear in $|\mathcal{U}|$ and n). Give a complete description and analysis of the data structure, and explain how much space it uses. You may assume that it takes $O(\log|\mathcal{U}|)$ bits to store the hash function h and $O(\log n)$ to store each element in the array R.

(d) Explain how to use your algorithm to solve the following problem:

i. **(4 pt.)** Given a k-sparse vector $a \in \mathbb{Z}_{\geq 0}^N$ ($\mathbb{Z}_{\geq 0}$ is the set of non-negative integers), design a randomized matrix $\Phi \in \mathbb{R}^{m\times N}$ for $m = O(\frac{k\log N}{\epsilon})$ so that the following happens. With probability at least 0.99 over the choice of Φ, you can recover $\tilde{a}$ given Φa, so that simultaneously for all $i \in 1, ..., N$, we have

$$|\tilde{a}[i] - a[i]| \leq \frac{\epsilon\|a\|_1}{2k}.$$

[**HINT:** *Think of the k-sparse vector a as being the histogram of the items in the list A from the previous parts.*]
[**HINT:** *How can you represent a hash function as a matrix multiplication?*]
[**HINT:** *Note that we want a tighter bound, and we want the bound to hold simultaneously for all i. How can we change b and T to achieve this?*]

ii. **(3 pt.)** Now, assuming the above holds for all i, use the k-sparseness of a to construct $\hat{a}$ from $\tilde{a}$ such that

$$\|\hat{a} - a\|_1 \leq \epsilon\|a\|_1.$$

iii. **(0 pt.)** **[This question is zero points, but worth thinking about.]** How does the guarantee in the previous part compare to the RIP matrices (and the compressed sensing guarantee that we can get from them, Theorem 1 in the Lecture 9 lecture notes) that we saw in class? (i.e., is this guarantee weaker? Stronger? Incomparable? The same?)

SOLUTION:

(a) We want to compute the expected count of x. We know:

$$\mathbb{E}[count(x)] = \mathbb{E}[\sum_i \mathbb{I}[h(x) == h(x_i)]] = \sum_i \mathbb{P}[h(x) == h(x_i)]$$

The C_x values of x in A will deterministically contribute C_x to the count. What about the remaining $n - C_x$ elements in A? Each will collide with $h(x)$ with probability $1/b$, leading us to:

$$\mathbb{E}[count(x)] = \sum_{i:x=x_i} 1 + \sum_{i:x_i\neq x} \frac{1}{b} = C_x + \frac{n-C_x}{b}$$

(b) As the hint suggests, the requirement $\mathbb{P}[count(x) < C_x] = 0$ is always true because if there are C_x values of x, all will be mapped to the same bin, and since other elements $y \neq x$ can be mapped to the same bin, $count(x) \geq C_x$.

For the second statement, we will use Markov's inequality, which is applicable due to the previous property:

$$\begin{aligned}
\mathbb{P}[count(x) \geq C_x + \epsilon n] &= \mathbb{P}[count(x) - C_x \geq +\epsilon n] \\
&\leq \frac{\mathbb{E}[count(x) - C_x]}{\epsilon n} \\
&= \frac{\mathbb{E}[count(x)] - C_x}{\epsilon n} \\
&= \frac{C_x + (n - C_x)/b - C_x}{\epsilon n} \\
&= \frac{n - C_x}{b \epsilon n}
\end{aligned}$$

Choose $b = e/\epsilon = O(1/\epsilon)$. Then:

$$\begin{aligned}
\mathbb{P}[count(x) \geq C_x + \epsilon n] &\leq \frac{n - C_x}{b \epsilon n} \\
&= \frac{n - C_x}{en} \\
&= \frac{1}{e} \frac{n - C_x}{n} \\
&\leq \frac{1}{e}
\end{aligned}$$

(c) Suppose we have T copies. We showed in $1b$ that the probability of 1 copy failing is $\leq 1/e$. Thus the probability that all fail is the intersection, given by $(1/e)^T$. We want to choose T such that this probability is δ. Solving for T as a function of δ:

$$(1/e)^T = \delta \Rightarrow T = -\log \delta$$

To calculate the modified construction's space, it'll need T copies of the hash map that each take $O(\log |\mathcal{U}|)$ and T copies of the arrays with b entries, where each entry takes $O(\log n)$ space. This requires a total of $O(T \log |\mathcal{U}| + Tb \log n)$ space.

(d) i. How could we implement a single hash function for a single fixed x? Sample a matrix $H \in \{0,1\}^{b \times N}$, where each column has exactly 1 non-zero entry. Then, to obtain T hash functions, repeat this T times and stack, yielding $\Phi \in \{0,1\}^{Tb \times N}$. But we need to choose T and b carefully to ensure we avoid failure for all $x \in \mathcal{U}$ with sufficiently high probability. To accomplish this, let's return to 1(b) and 1(c) to identify how to choose new b and T.
Define $\epsilon' = \epsilon/2k$. From 1(b), we know that the desired property

$$|\tilde{a}_i - a_i| \leq \epsilon' n = \frac{\epsilon n}{2k} = \frac{\epsilon ||a||_1}{2k}$$

will hold with probability $1/b\epsilon' = 2k/b\epsilon$. So choose $b = 2ke/\epsilon = O(1/\epsilon)$, yielding the same probability as before: $1/e$. Using the desired error probability and b, we now need to choose T. In part 1c, we considered a single fixed $x \in \mathcal{U}$; now, we need to consider any x_i. To do this, we'll use a union bound:

$$\mathbb{P}[\cup_i T \text{ copies fail for } x_i] \leq \sum_i \mathbb{P}[T\text{copies fail for } x_i] = N(1/e)^T$$

Using the desired $\delta = 0.01$, we can now solve for T:

$$\delta = N(\frac{1}{e})^T \Rightarrow \log \frac{\delta}{N} = T \log e^{-1} = -T \Rightarrow T = \log N - \log \delta = O(\log N)$$

Thus, $Tb = (\log N) * \frac{2ke}{\epsilon} = O(\frac{k \log N}{\epsilon})$.
We now have T and b. By sampling Φ as specified at the beginning, with $m = Tb = O(\frac{k \log N}{\epsilon})$, we ensure a low probability of error.

ii. Our approach will be to construct $\hat{a}$ by taking the k-th biggest entries of $\tilde{a}$ and setting the rest to 0. From the definition of the L1 norm and our chosen construction:

$$||\hat{a} - a||_1 = \sum_{i=1}^{N} |\hat{a}_i - a_i| \leq 2k \frac{\epsilon ||a||_1}{2k} = \epsilon ||a||_1$$

where the middle step that replaces the sum with $2k$ relies on the fact that there can be at most $2k$ elements in the sum whose values could be non-zero since both $\tilde{a}$ and a are k-sparse, meaning even if the two have "disjoint" non-zero values, the remaining $N - 2k$ values must all be 0.

2. **(6 pt.)[Aquarium apportionment.]**

Suppose that there is a population of n fish, belonging to one of two types: (**A**lgae eaters and **B**arbs). There are also a bunch of fish tanks that these fish could live in. These fish are somewhat picky: each one has a list of at least $1 + \log_2(n)$ fish tanks that it would be happy in (e.g., the algae eater Algernon likes the tank with the mini pirate ship, or the tank with the colorful rocks, but he is not into the tank with the fake ferns). The fishes' lists may be different, and may overlap.

Unfortunately, these two types of fish don't get along well: when an algae eater and a barb are put in the same tank as each other, they will become agitated and fight and stress each other (and you) out.

Use the probabilistic method to show that it is possible to put all of the fish into fish tanks in such a way that each fish is happy with their tank, and none of them fight with each other. That is, define a probability distribution for putting fish into tanks so that, with positive probability, everyone is happy and there is no fighting.

[**HINT:** *If you put each fish in a random tank that they like, the resulting scheme does* ***not*** *seem to work...you'll have to come up with a more interesting random distribution.*]

SOLUTION: Our algorithm will be to sample a random fish type (A or B) for each tank uniformly at random i.e. each with probability 0.5. Then, iterating over each tanks, we take all fish who (a) are happy with this tank and who (b) match the tank's type, and put them in this tank.

Note that, by construction, each tank will have fish of all one type i.e. there's no way to assign a fish of type A and a fish of type B to the same tank. Therefore, we only need to worry about whether we are unable to place a fish in some tank it is happy with.

Define A_i as the event that the $i \in [n]$ fish is unable to be placed in a satisfactory tank. Recall that each fish likes $1 + \log_2(n)$ tanks. Since each tank's type is sampled uniformly at random, the probability that the i-th fish's type disagrees with all of the i-th fish's preferences is:

$$\mathbb{P}[A_i] = \left(\frac{1}{2n}\right)^{1+\log_2 n} = \frac{1}{2n}$$

Consequently, the probability that the ith fish can find some tank matching its type is:

$$\mathbb{P}[\overline{A_i}] = 1 - \frac{1}{2n}$$

Now, each fish's preferences might be dependent - we don't know! But per the union bound, the probability that at least 1 fish is unsatisfied is:

$$\mathbb{P}[\cup_i A_i] \leq \sum_i \mathbb{P}[A_i] = n\frac{1}{2n} = \frac{1}{2}$$

So the probability that no fish is unsatisified i.e. all fish are satisified is:

$$\begin{aligned}\mathbb{P}[\text{all fish satisfied}] &= 1 - \mathbb{P}[\text{at least 1 fish unsatisified}] \\ &= 1 - \mathbb{P}[\cup_i A_i] \\ &\geq 1/2\end{aligned}$$

Thus, the probability that all fish are satisfied under a random assignment scheme is at least 1/2. So we know there exists a way to assign fish into tanks such that everyone is happy and there is no fighting.

3. **(0 pt.)[Who needs geometry?]**

This question is worth zero points, but it is an entertainingly obnoxious use of the probabilistic method that you will probably find satisfying.

Suppose that your friend has – perhaps adversarially – drawn 10 points on the surface of a flat table, which you can think of as an infinite plane. Your friend gives you 10 quarters, which you can think of as discs, all of the same size.

Your challenge is to place the quarters on the table such that no quarters overlap, *and* every one of your friend's chosen points is covered. Show that this is *always* possible, regardless of the locations of the points.

One hint is below. Some more hints appear on the **next** page, so that you can avoid spoilers if you wish.

[**HINT:** *We'll save you some time – trying to think about / classify the possible arrangements of points is not helpful, at least for the purpose of this problem in the context of this course on randomized algorithms. The solution we have in mind behaves the same way regardless of the locations of the points.*]

[**HINT:** *Imagine an arbitrarily large planar sheet of quarters, pressed together as tightly as possible. Such a sheet covers about 90.69% of the plane. See `https://en.wikipedia.org/wiki/Circle_packing` for details.*]

[**HINT:** *The fact that there are 10 points is important, although the significance may not be clear until you start down the right solution path...*]

SOLUTION: