Privacy in Statistics and Machine Learning — Spring 2021
Homework 1: Due Friday, February 12, 2021

Adam Smith and Jonathan Ullman

Collaboration and Honesty Policy Reminder: Collaboration in the form of discussion is allowed. However, all forms of cheating (copying parts of a classmate's assignment, plagiarism from papers or old posted solutions) are NOT allowed. A rough rule of thumb: you should be able to walk away from a discussion of a homework problem with no notes at all and write your solution on your own. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is forbidden.

- *You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem.*
- You must identify your collaborators. If you did not work with anyone, you should write "Collaborators: none."
- Asking and answering questions in every forum the class provides (on Piazza, in class, and in office hours) is encouraged!
- Even though look up answers is forbidden, using the web to find alternative explanations of concepts you need for the homework *is* allowed, and encouraged. For example, you can look up background on probability and linear algebra, documentation for particular programming languages, etc.

Problems to be handed in

1. (Randomized response—Exercise 3 from Lecture 1's in-class exercises.) Consider the second randomized response mechanism described in Lecture 1: On input a function $\varphi : \mathcal{X} \to \{0,1\}$ and a dataset $\mathbf{x} = (x_1, ..., x_n)$ where each x_i is in domain $\mathcal{X}$, compute

$$Y_i = \begin{cases} \varphi(x_i) & \text{w.p. } \frac{e^\varepsilon}{e^\varepsilon+1}, \\ 1-\varphi(x_i) & \text{w.p. } \frac{1}{e^\varepsilon+1}. \end{cases}$$

   and return $(Y_1, ..., Y_n)$.

   Give a procedure that, given the outputs $Y_1, ..., Y_n$ from randomized response on input $x_1, ..., x_n$, returns an estimate A such that

$$\sqrt{\mathbb{E}\left(\left(A - \sum_{i=1}^{n} \varphi(x_i)\right)^2\right)} = \frac{e^{\varepsilon/2}}{e^\varepsilon - 1}\sqrt{n}.$$

   The randomness here is over the choices of the randomized response mechanism. [*Hint:* How would you choose $a, b \in \mathbb{R}$ so that $\mathbb{E}(aY_i - b) = x_i$?]

2. (Global Sensitivities.) For all of the following cases, assume we have a datset $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and a function $f : \mathcal{X}^n \to \mathbb{R}^d$ (for some d that varies between the different parts of the question). For each of the following functions f and data domains $\mathcal{X}$, (i) give as tight a bound as you can on the

global sensitivity of the function f. If the sensitivity is not bounded, answer ∞. (ii) Give as tight a bound as you can on the diameter of the range of f: specifically, what is the maximum difference (in ℓ_1 norm) between two possible outputs of f?

(a) The high-dimensional mean $f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$ when $\mathcal{X} = \{v \in \mathbb{R}^d : \|v\|_1 \leq 1\}$.

(b) The unnormalized covariance matrix when $\mathcal{X} = \{v \in \mathbb{R}^d : \|v\|_1 \leq 1\}$. Here $f(x) = C$ is a $d \times d$ symmetric matrix with entries $C_{j,k} = \sum_{i=1}^{n} x_i(j)x_i(k)$, and $x_i(j)$ refers to the j-th entry of vector x_i. In other words, if we write the input as a $n \times d$ matrix X whose rows are the x_i, then $f(\mathbf{x}) = X^T X$. To measure sensitivity, we think of $f(\mathbf{x})$ as a single vector of length d^2.

(c) The median $f(\mathbf{x}) = \text{median}(x_1, ..., x_n)$, when $\mathcal{X} = [0, 1]$.

(d) Consider an arbitrary domain $\mathcal{X}$, and k different predicates $f_1, .., f_k$ (so each f_j maps $\mathcal{X}$ to $\{0, 1\}$). Let $f(x) = \max_{j=1,...,k} \frac{1}{n}\sum_{i=1}^{n} f_j(x_i)$.

(e) The US government uses a specific algorithm H to decide, based on state populations, how many of the 435 seats in the House of Representatives will go to each state.[1] This "apportionment" is recalculated after each decennial Census. For simplicity, think of the Census data set as just recording each person's state of residence, so $\mathcal{X} = \{1, ..., 50\}$.

Let $f(\mathbf{x})$ be the function that reports the number of people whose residence would need to change in order to change the apportionment $H(\mathbf{x})$. In other words, $f(\mathbf{x})$ is the Hamming distance to the nearest data set $\mathbf{y}$ with $H(\mathbf{y}) \neq H(\mathbf{x})$.

(f) Suppose we a fixed set of vertices V (independent of the data set). Our data set is a list of edges: each x_i is a pair of vertices (u, v) (so that $\mathcal{X} = V \times V$). Let $G_{\mathbf{x}}$ be the resulting graph, and let $f(\mathbf{x})$ denote the number of connected components in $G_{\mathbf{x}}$.

(g) Let $\mathcal{X} = V \times V$ as above, and let $f(\mathbf{x})$ be the length of the shortest path between two specific vertices (let's say vertices number 1 and 2, assuming they are numbered 1 through $|V|$). (If no path exists, the length of the shortest path is ∞).

(h) Let $\mathcal{X} = V \times V$ as above, and let $f(\mathbf{x})$ be the number of edge-disjoint paths from vertex 1 to vertex 2.

3. (In-class Exercise 3 from Lecture 3: More accurate reconstruction with more random queries.) In this question we'll explore how to interpolate between the two reconstruction theorems we've seen. Specifically, we will prove a version of Theorem 2.5 that gives a more accurate reconstruction when we have $k \gg n$ queries. Suppose we have the following version of Claim 2.6 from the lecture notes:

Claim 0.1. *Let $t \in \{-1, 0, +1\}^n$ be a vector with at least m non-zero entries and let $u \in \{0, 1\}^n$ be a uniformly random vector. Then for every parameter $2 \leq w \ll 2^m$*

$$\mathbb{P}\left(|u \cdot t| \geq \frac{\sqrt{m \log w}}{10}\right) \geq \frac{1}{w} \tag{1}$$

Using this claim, prove the following theorem

Theorem 0.2. *If we ask $n^2 \ll k \ll 2^n$ queries, and all queries have error at most αn, then with extremely high probability, the reconstruction error is at most $O(\frac{\alpha^2 n^2}{\log(k/n)})$.*

[1] The exact algorithm doesn't matter here, but it's called the "Huntingdon-Hill method".

How does this theorem compare to the reconstruction attacks we've seen for $k \approx n^2$? What about $k \approx 2^{\sqrt{n}}$? What about $k \approx 2^n$?

4. A social media company wants to provide information to its advertisers on how often people click on their ads. You work for an advertiser. Each hour, the company gives you the demographic profile of one user who saw your ad. This demographic profile is rich enough for you to identify the user in your own database. They also give you a running count of the number of users who have actually clicked on your ad. They add a little bit of noise "to protect users' privacy", but you suspect that you can derive a good guess as to who, exactly clicked on your ads.

   We'll model this as follows: there is a sequence $\mathbf{x}$ of secret bits, $x_1, x_2, x_3, ..., x_n \in \{0, 1\}$. The bit x_i indicates that the ith user clicked on the ad.

   The mechanism for releasing the running counter works as follows: At each time step $i = 1, 2, ..., n$, select a uniformly random bit Z_i (independently of previous ones) and output

$$a_i = \sum_{j=1}^{i} x_j + Z_i \,.$$

   (In other words, at each step, we flip a coin to decide whether we should add one fake person to the counter or not).

   How well can one recover the vector $\mathbf{x}$ from the sequence of outputs $\vec{a}$? The answer depends on what we know about $\mathbf{x}$ ahead of time. We will consider two situations:

   (a) The bits of $\mathbf{x}$ are uniformly random and independent. The only input to the attacker is $\vec{a}$ (the vector of noisy counters).

   (b) The bits of $\mathbf{x}$ are uniformly random and independent, but the attacker has some extra information. For each i, the attacker has a guess w_i which is equal to x_i with probability 2/3, independently for each i. The attacker's inputs consist of $\vec{a}$ and the vector of guesses $\vec{w}$.

   For each of these situations, your job is to come up with as good an algorithm as you can to guess the entries of $\mathbf{x}$. You should evaluate your algorithm by coding it up (in a language of your choice) and plotting the fraction of bits of $\mathbf{x}$ recovered by your algorithm for $n = 100, 500, 1000, 5000$, and, if you can, $50,000$. (You should run the algorithm on fresh random inputs at least 20 times for each values of n, and report the mean and standard deviation for each one.) You'll need to code up the release mechanism, too, in order to run the experiments.

   You may use standard linear algebra libraries (such as those provided by `numpy` in Python). You may use standard solvers for linear systems or linear programs.

   You should submit a single PDF file containing

   - A written description of your algorithms (whatever mix of English and pseudocode you like is fine—it should be clear, readable, and self-contained). (Maximum 1 page each)
   - Plots of your results for each of the two settings
   - Discussion of your results (Maximum 1/3 page)
   - Documented code
   - (Optional) A link to a public repository with your code, ideally in a notebook format.

To receive full credit, your algorithms should recover significantly more than $3/4$ of the bits of $\mathbf{x}$ in case 1, and significantly more than $4/5$ of the bits of $\mathbf{x}$ in case 2. There is no specific running time requirement, but you should ideally be able to run experiments with $n = 50,000$, so exponential time algorithms won't be feasible, and even quadratic-time algorithms will be painfully slow. You can use any of the techniques from class, or ones you invent. See how well you can do!