Privacy in Machine Learning and Statistical Inference October 15, 2018

Homework 1: Attacks

Instructor: Adam Smith

1. **The Hadamard Attack and Random Queries**

   In this problem, you will implement the Hadamard attack from class (in a language of your choice), as well as a version in which the attacker sees the results of a similar number of random queries. Your goal is to verify the result from class, as well as to understand the differences between the Hadamard and random case.

   We'll consider the specific mechanism that adds Gaussian noise. Assume n is a power of 2. We'll consider two experiments:

   (a) The Hadamard attack (with parameters n and σ):
   - $\mathbf{x}$ is selected uniformly at random in $\{0,1\}^n$.
   - The mechanism releases $\mathbf{a} = \frac{1}{n}H\mathbf{x} + \mathbf{Y}$, where the entries of $\mathbf{Y}$ are i.i.d. from $N(0, \sigma^2)$, and H is Hadamard matrix with entries in $\{-1, +1\}$.
   - The attacker computes $\mathbf{z} = H\mathbf{a}$, and estimates $\hat{\mathbf{x}} = round(z)$.

   (b) The random query attack (with parameters n, m, and σ):
   - $\mathbf{x}$ is selected uniformly at random in $\{0,1\}^n$.
   - Let B be an $m \times n$ matrix with entries selected i.i.d. and unfiromly in $\{0,1\}$
   - The mechanism releases $\mathbf{a} = \frac{1}{n}B\mathbf{x} + \mathbf{Y}$, where the entries of $\mathbf{Y}$ are i.i.d. from $N(0, \sigma^2)$.
   - The attacker computes $\mathbf{z} = \arg\min_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{a} - \frac{1}{n}B\mathbf{y}\|_2$, and estimates $\hat{\mathbf{x}} = round(z)$.

   For this problem, you should implement the two attacks and run them for various values of the parameters n, σ and m. For each parameter setting and attack, run 20 experiments with fresh randomness and compute the mean and average deviation of $\frac{Ham(\hat{x},x)}{n}$ (the fraction of incorrectly predicted bits in x). Try the attacks with $n \in \{128, 512, 2048, 8192\}$, with σ ranging in powers of 2 from $1/2$ down to $1/\sqrt{32n}$ and $m \in \{1.1n, 4n, 16n\}$. (You do not necessarily need to try all possible combinations. For example, if an attack works perfectly with $\sigma = 1/16$, then it will work with smaller values of σ. Also, depending on your machine, the random queries attack with larger values of n, m may take too long to run.)

   To implement the `argmin` step for random queries, you should use a library that solves least-squares problems, for example, `numpy.linalg.lstsq` or `scipy.optimize.least_squares` in Python. Feel free to ask for advice on Piazza—in most programming languages, you have to be a bit careful for operations on big matrices and vectors to run quickly.

   You should submit a single PDF file containing

   - Plots of your results for each of the two settings.
   - Discussion of your results (about 1/2 page), answering the following questions:
     - How does the performance of the Hadamard attack compare to the theory we developed in class? (You may take $\alpha = \sigma$ for the discussion.)
     - How do the two attacks compare?
     - How does m influence the effectiveness of the random queries attack?
   - Documented code
   - (Optional) A link to a public repository with your code.

2. **A running counter:** 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 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

$$a(i) = \sum_{j=1}^{n} x(j) + Z_i \, .$$

   (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 $\mathbf{a}$? 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 $\mathbf{a}$ (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 $\mathbf{a}$ and the vector of guesses $\mathbf{w}$.

   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 $n = 100, 500, 1000$ and 5000. (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 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: what did you learn? (Maximum 1/2 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 $1/2$ of the bits of $\mathbf{x}$ in case 1, and significantly more than $2/3$ of the bits of $\mathbf{x}$ in case 2. There is no specific running time requirement (but you have to be able to run experiments with $n = 5,000$, so exponential time algorithms won't be feasible). You can use any of the techniques from class, or ones you invent. See how well you can do!

3. **(Chernoff Bounds)** For this problem, you will have to read some additional material on "Chernoff bounds". The term refers to a family of boudns in probability that give tight control over how the "tails" (the values that occur with small probability) of a sum of independent random variables behave.

   (a) *Read* the following notes:
   `http://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf`
   If the notes are hard to understand, you may want to read more about Chernoff bounds, also called Hoeffding bounds or Bernstein inequalities, on the web (e.g., youtube "Chernoff bounds").

   (b) (**Sampling**) You are given a list of n real numbers $a_1, \ldots, a_n$ in the interval $[-1, 1]$ with the average $\bar{a}$. Let $s_1, \ldots, s_k$ be random samples taken from the list uniformly with replacement (so that each s_1,..., s_k are independent), and $\hat{a}$ be the average of the samples. Let $error = |\hat{a} - \bar{a}|$.

      i. Based on the Chernoff bounds in the notes above, what should k be to guarantee that $error \leq \alpha$ with probability at least $1 - \delta$?

      ii. **(A small synopsis)** Let's generalize the result from part i a bit. Suppose we have a population of n people we wish to study (for example, Penn State undergraduates). There is a collection of d yes/no questions we wish to ask each of them (for example, "do you spend more time studying or exercising?", "do you arrive late to class more than twice a week?", "have you been the victim of a physical assault on campus", etc).
      We want to know the *proportion* of individuals in the population who answer "yes" and "no" to each of the questions. Unfortunately, we don't have time to ask all of the people all of the questions, so we decide to take a random sample of k people and ask them all of the questions. We want to know how big k has to be for us to get a good answer to all of our questions (with high probability).
      We can model the questions as functions $f_1, ..., f_d$ taking values in $\{0, 1\}$.
      You may assume that we chose the k people *uniformly with replacement* from the population (that is, it is ok if we ask some people the questions twice). This allows you to assume that the k samples are truly independent.
      - Show that, for any $\alpha > 0$, it suffices to take $k = O(\frac{\log(d)+\log(1/\delta)}{\alpha^2})$ samples in order for the following to hold: with probability at least $1 - \delta$, for all d of the questions, the proportion of "yes" (and "no") answers in the *sample* is the within α of the proportion in the *population*. By "within α, we mean that for every question f, we have $\left|\frac{1}{n}\sum_{i=1}^{n} f(x_i) - \frac{1}{k}\sum_{j=1}^{k} f(s_j)\right| \leq \alpha$, where $x_1, ..., x_n$ is the population, and $s_1, ..., s_k$ are the people in the sample (possibly including repetitions).
        [*Hint:* Use the union bound.]

      iii. Now we want to show that reconstruction attacks are difficult to carry out when queries are answered based on a sample. Suppose there is an input data set $x(1), ..., x(n)$ where each $x(i)$ is a bit, and suppose a mechanism produces a vector of answers $\mathbf{a}$ by first sampling $k = \frac{n}{3}$ positions at random (without replacement, for simplicity) from this data set and uses that subset of bits to answer a sequence of queries. [It doesn't matter here what the queries are or hw the answers are produced. All that matters is they depend only on a small part of the whole data set.]
      Consider the task of an attacker, who, given $\mathbf{a}$, guesses an approximation $\hat{\mathbf{x}}$ to $\mathbf{x}$. Let $E = \frac{Hamming(\hat{\mathbf{x}},\mathbf{x})}{n}$ denote the fraction of correctly recovered bits.
      Use the Chernoff bound to show that, if $\mathbf{x}$ is selected uniformly at random then, with probability $1 - \exp(\Omega(n))$, the error E is at least $\frac{1}{4}$.
      [*Hint:* Since 2/3 of the bits of x are independent of what the attacker sees, you can think of them as being selected *after* the attacker produces $\hat{x}$.]

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.