University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Winter 2020
Course Staff: David Cash, Alex Hoover, Rohan Kumar, Blase Ur, Valerie Zhao

Assignment 7: Differential-Privacy

Due at 11:59pm Monday, March 2

Introduction

This assignment covers a few aspects of *differential privacy* as covered in lecture during Week 7. It is divided into two parts.

In the first part, you will implement and experiment with some of the mechanisms discussed in lecture, and observe their effect on a US Census dataset (which has been augmented with synthetic information like names). For this part you will work with `pandas`, a very popular Python data analysis tool set that many of you will probably use in other contexts. The assignment does not assume prior knowledge of `pandas`.

In the second part, you will implement an attack that broke many implementations of the Laplace mechanism in 2012 (it also breaks your implementation from the first part). The root issue exploited by this attack concerns how libraries generate random floating point numbers. Implementing this attack is more technically involved than the first part of the assignment, and requires a deep-dive into the representation of floating point numbers and the implementation of `numpy.random`.

Rules (unchanged from before, except to remove mention of a server)

Collaboration policy. Please respect the following collaboration policy: You may discuss problems with up to 3 other students in the class, *but you must write up your own responses and your own code. You should never see your collaborators' writing, code, flags, or the output of their code.* At the beginning of your submission write-up, you must indicate the names of your (1, 2, or 3) collaborators, if any. You may switch groups between assignments but not within the same assignment.

Sources. Cite any sources you use. You may Google liberally to learn about how to configure Apache, use JavaScript, use MySQL (or another database of your choosing), write to the database (e.g., using PHP), or navigate the command line. You may also Google liberally to learn the concepts underlying fingerprinting. **You may not, however, use or consult existing libraries for browser fingerprinting. Your solutions, as well as the precise way you turn the concepts into code, should be your own.** You **must**, however, note at the end of your writeup for each task the sources you referenced. Please note any you found particularly helpful since we might include them in future versions of this writeup.

Campuswire. We encourage you to post questions on Campuswire, but do not include any of your code in the Campuswire posts. If you have a question that you believe will reveal secrets you have discovered while working on the assignment, post privately to just the instructors. If you have a question that you believe will be of general interest or clarifies the assignment, please post publicly. If you are uncertain, post privately; we will make public posts that we believe are of general interest.

Grading. Responses will be graded for correctness and clarity.

What and How to Submit

You will submit a set of files:

1. Code files `part1.py` and `part2.py`, with the appropriate functions filled in.
2. Plot files `problem1-3.png`, `problem1-4.png`, and `problem1-5.png`.
3. Your writeup file as a pdf or .txt.

You will upload these files to Canvas. You may zip/tar/gzip/7zip/etc. them or submit them individually.

Part 1: Introduction to Differential Privacy (50 points total)

Problem 1.0: (0 points)

For the rest of this part you will work with Python modules `pandas`, `numpy`, and `matplotlib`. All of these can be installed via your favorite method, like `pip`. You can probably learn the latter two libraries as you go, but it is worth taking a minute to familiarize yourself with a tutorial on `pandas` if this is your first time using it. Reasonable tutorials are here and here (the latter is especially gentle). Try to get some idea about what a dataframe is and the various ways to query one. Also pay attention to merging dataframes.

The problems below will use the `adult` dataset, which is available here. This dataset contains synthetic "personally identifiable information" (PII). That dataset, as well as the problems in Part 1, are courtesy of Joe Near at the University of Vermont.

Problem 1.1: Linking Attack (10 points)

1. In the `part1.py` file, fill in the function `deidentify`. Your function should take as input the dataframe from the `adult` dataset (or one with the same columns) produced by `read_csv` called on the `csv` file. It should return the same dataframe, but with the `'Name'` and `'SSN'` columns deleted. This models a typical faulty data release.

2. Now fill in `pii` to also take in a dataframe from `adult`, but have it produce a dataframe with only columns `'Name'`, `'DOB'` `'SSN'` and `'Zip'`. This models the data that an attacker may have obtained.

3. Now fill in the `link_attack` function. It should take as input the deidentified and PII dataframes, and should return a dataframe consisting of all rows from the deidentified dataframe that could be uniquely linked, reattached to their PII. Proceed as in the classic Sweeney attack from lecture. Your function should output a dataframe consisting only of the individuals that were uniquely recoverable. (Hint: Try `pandas merge`.)

What to submit. In addition to filling in the given functions, describe how many individuals could be uniquely identified, and how many could not.

Problem 1.2: k-Anonymity (15 points)

Fill in the function `is_k_anon`. It takes as input a dataframe `df`, a list of column names `cols`, and an integer `k` assumed to be at least 1. It should output True of False depending on if the dataframe is k-anonymous with respect to `cols`.

You should test your implementation on dataframes that are k-anonymous but not $(k+1)$-anonymous. It should also run reasonably fast with the `adult` dataset (which is not 2-anonymous for most choices of columns) but you will not otherwise be graded based on run-time.

What to submit. In addition to filling in the given function, in your write-up, give a big-Oh estimate of the runtime of your algorithm in terms of the number of rows in the dataframe, and explain very briefly how you determined this.

Problem 1.3: Laplace Mechanism (15 points)

1. Fill in the function `num_bachelors` that takes a dataframe as input and returns the number of individuals with a exactly Bachelor's level of education.

2. Fill in the function `laplace_mech` that takes as input a float `query` (representing the output of a query), a float `sensitivity`, and a float `epsilon`. It should return the output of the Laplace mechanism with these values. (Note: Use `numpy.random.laplace`.)

3. Now compute *empirical probability density functions* for the Laplace mechanism with called on `query` 200.0, `sensitivity` 1, and `epsilon` set to 0.5, 1, and 10. More specifically, run each setting 10,000 times, and plot the outputs (binned into groups) by calling `plt.hist(..., bins=50)` three times and then `plt.show()`. You should get a plot with three overlayed histograms.

What to submit. Submit your plot as `problem1-3.png`. In your write-up, answer the following: What did `num_bachelors` return when run on the `adult` dataset? What is the sensitivity of `num_bachelors`? Explain briefly.

Problem 1.4: Comparing Queries (5 points)

Using `laplace_mech` from the previous section, compute a differentially private answer with `query` set to 200.0 and again with `query` set to 201.0, with `sensitivity` set to 1 and `epsilon` set to 0.5. Take 10,000 samples again for each and overlay the *empirical probability density functions* again using `plt.hist(..., bins=50)` and `plt.show()`. Repeat for `epsilon` set to 1 and 10.

What to submit. Submit your plot with `epsilon` set to 1 as `problem1-4.png`. In your write-up explain how much the overlap between the histograms relates to the security/privacy of an individual in a dataset.

Problem 1.5: Plotting the Error (5 points)

Fill in the function `plot_error` which produces the following plot: Using `laplace_mech` from the previous section, compute 10,000 differentially private answers with `query` set to the result output for `num_bachelors` with `sensitivity` set correctly. For each answer, compute absolute error (which is always positive) between the differentially private answer and the true answer. Put the errors in bins of size 20.

Repeat this with `epsilon` set to 0.5, 1, and 10. Overlay the 3 histograms (all with bin-size 20) from the three runs in on plot as before.

What to submit. Submit the plot you obtained from your different runs as `problem1-5.png`.

In your write-up, answer the following: What do you notice about the error as `epsilon` increases? How does this compare to the security/privacy of an individual in the dataset as `epsilon` increases? How would you go about choosing a value for `epsilon` when trying to protect individuals as well as obtain accurate results?

Part 2: Floating-Point Attack (50 points total)

The included file `mironov12.pdf` describes an attack against implementations of the Laplace mechanism that use tools like `numpy.random.laplace`. This is an example of a nicely written research paper, and it doesn't require much more than our lectures to be fully understood. (You should refer to the paper in the course of working this part, but you can skip the analysis in Section 4.6 and the proof of Theorem 1 in Section 5.)

Here is a brief summary of the attack. It shows that one can often distinguish with certainty if a given float was output by either `numpy.random.laplace(1)` or `1.0+numpy.random.laplace(1)`; That is, just by looking at the bits of the float, you can often tell if the sample was shifted by 1 or not! Section 4.5 of the paper has a visualization of what this looks like for an attacker. The ability to determine this information conclusively, even part of the time, is a bad violation of the requirement for differential privacy (in this case, failing to protect a query with result 0 or 1).

How does this attack work? Start by looking at the implementation of `my_laplace` in `part2.py`. This uses a standard technique also used in the `numpy` version (which is written in C). The sampler first draws a sample from `numpy.random.uniform`, which returns a float between 0 and 1. It then takes the natural log of this number using `numpy.random.log`, and multiplies the result by a random sign. Finally it scales the result.

There are two important features that cause the failure of this sampler:

1. First, the uniform sampler only returns certain floats between 0 and 1, and not every possible representable float in that range. Let us call this set of floats U. (See Table 1 on page 6 of the Mironov paper.)

2. Second, the implementation of the log function also does not output every possible float. Let us denote by L the set all floats that log would ever output. (See the figures on page 6 of the Mironov paper.)

This suggests a general strategy for an attack. Given a scale s and a float y which is either drawn from `my_laplace(s)` or from that distribution plus `1.0`, the goal is to determine if it could have came from `my_laplace(s)`. At a high level, it proceeds:

1. If $y > 0$ set $y \leftarrow -1 * y$.

2. Find *all* floats x such that $s \cdot \texttt{numpy.log}(x) = y$, where s is the scale (note the *equality* of floats here; you almost never want to test this, but here you do).

3. If any $x \in U$, output True (y could be output by the sampler). Else output False.

It is possible that we settle for approximations of these tests for membership in sets. The goal is after all an attack that will distinguish the types of samples with some probability, so it will not work 100% of the time anyway.

What to submit. Fill in the function `is_scaled_laplace_sample` in `part2.py`. You may implement helper functions above in the file. Further hints and a rubric are provided below.

In your write-up, explain in a sentence why the attack in the paper violates differential privacy (you should refer to the definition of differential privacy).

Next, describe the details of your version of the above attack (e.g. how you tested for membership in sets, and other issues you ran into). Run your function on 1000 samples from `my_laplace(1)`, and then on 1000 samples from `1.0+my_laplace(1)`. Report how many times your function returned True for each case.

Correctness. Part of your grade for this part will depend on the effectiveness of your distinguishing function (the rest will come from your write-up). We will run your code on 1000 samples from either distribution (as you are instructed do above). Let x and x_1 be the respective counts of how many times your function outputs True. If $x - x_1 > 450$, your code will receive full credit. If $x - x_1$ is smaller, you will receive partial credit.

Hints. The paper describes how floats are encoded. Python uses 64 bit floats that follow this standard. At a few points in your code it might be helpful to have the bits of a float, either to print out or to manipulate. To help, the file `part2.py` includes some utility functions for converting a float to an ASCII string of 64 ones and zeros, and also to convert such a string back to a float. Feel free to change these if you want. Note that these bits follow the encoding from Section 5.2 (see also Wikipedia for a visualization of this bit encoding).

To find x such that $s \cdot \texttt{numpy.log}(x) = y$, you can try using `numpy.exp`. This isn't exactly an inverse of the log function though, but it will get you close enough to start looking. Don't worry about doing this part with absolute correctness. Something that works based on empirical observations can be good enough.

To test if a value $x \in U$, think about the binary representation of a number fitting the condition given in Table 1. Don't try to test this algebraically.