COMP 430/530: Data Privacy and Security - Fall 2024
Homework Assignment # 2

KOÇ UNIVERSITY

Introduction

This assignment consists of 3 parts (100 pts + 5 pts bonus). Part 1 is theoretical; you can write your answer on paper, on a tablet, or using LaTeX. Parts 2 and 3 are mostly implementation. In these parts, use the Python skeleton files we provide as your starting point. Ensure that you adhere to the function names and parameters we ask for. For questions that require plots or discussion, submit your answers in a separate report (pdf file).

Part 1: Privacy Proofs [35 pts (15+20 pts)]

(a) Let algorithm A take as input a dataset D and a numeric query q. The algorithm executes the query q on D and then adds Laplace noise with mean $= 0$ and scale $= \varepsilon$. (Note that the scale is different from the original Laplace mechanism.) Does A satisfy ε-DP? Why or why not? Are there any circumstances in which A satisfies ε-DP?

(b) Consider two values v_i, v_j. There are many ways to measure the distance $d(v_i, v_j)$ between them, e.g., absolute value distance, Euclidean distance, etc. A **metric** is a measure of distance which satisfies the following properties:
(1) Non-negativity: $d(v_i, v_j) \geq 0$, for all v_i, v_j
(2) Identity of indiscernibles: $d(v_i, v_j) = 0$ if and only if $v_i = v_j$
(3) Symmetry: $d(v_i, v_j) = d(v_j, v_i)$ for all v_i, v_j
(4) Triangle inequality: $d(v_i, v_k) \leq d(v_i, v_j) + d(v_j, v_k)$

Now consider that we are in a Local Differential Privacy (LDP) scenario. Each user has a true value v coming from a finite universe $\mathcal{U}$. A distance metric d is known for $\mathcal{U}$. The notion of metric-based LDP (MLDP) is defined as follows.

Definition 1 (α-MLDP) *A randomized algorithm $\mathcal{A}$ satisfies α-MLDP, where $\alpha > 0$, if and only if for any inputs $v_1, v_2 \in \mathcal{U}$:*

$$\forall y \in Range(\mathcal{A}) : \quad \frac{Pr[\mathcal{A}(v_1) = y]}{Pr[\mathcal{A}(v_2) = y]} \leq e^{\alpha \cdot d(v_1, v_2)}$$

where $Range(\mathcal{A})$ denotes the set of all possible outputs of algorithm $\mathcal{A}$.

Notice that MLDP is a modified version of the original LDP definition. In MLDP, indistinguishability of v_1, v_2 is dependent on not only the privacy parameter α but also $d(v_1, v_2)$. Since MLDP is different from LDP, we need to design new algorithms to achieve MLDP.

Let Ψ be a perturbation algorithm $\Psi : \mathcal{U} \to \mathcal{U}$, i.e., it takes as input some value $v \in \mathcal{U}$ and perturbs it to some value $y \in \mathcal{U}$. Given $v \in \mathcal{U}$, the probability that Ψ produces y as its output is:

$$\Pr[\Psi(v) = y] = \frac{e^{\frac{-\alpha \cdot d(v,y)}{2}}}{\sum\limits_{z \in \mathcal{U}} e^{\frac{-\alpha \cdot d(v,z)}{2}}}$$

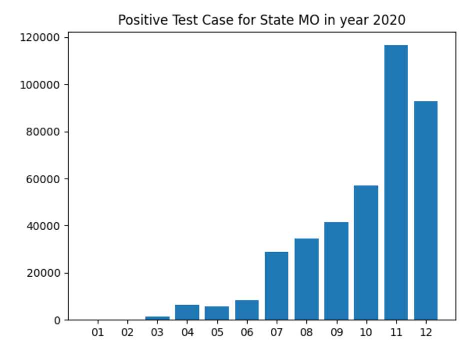
Prove that this Ψ satisfies α-MLDP.

Hints: The proof strategy is similar to one of the proofs we saw in class. In the proof, you may need to use the properties of a metric.

Part 2: DP Implementation [35 pts (Task 1: 20 pts, Task 2: 15 pts)]

Consider the COVID-19 dataset provided to you: *covid19-states-history.csv*. This dataset is originally from the Covid Tracking Project, but we modified it for this homework assignment. The CSV file contains daily data on the COVID-19 pandemic for individual US states. Each row corresponds to a state's reported monthly death count, negative test count, and positive test count.

Task 1: Your task is to construct a histogram to examine the monthly positive case counts of COVID-19 for Texas (identified by the state code TX). This histogram should help us understand how positive cases fluctuate over time. Your histogram should look something like:



(a) Implement the function *get_histogram(dataset, state, year)* to construct a non-private histogram of monthly COVID-19 positive test counts. By default this function is configured for state = TX and year = 2020, but it must be adaptable to accommodate different states and years as inputs. The return value of this function should be a Python list containing 12 elements:

[January_positives, February_positives, ..., December_positives]

Draw a histogram using this list and add it to your report (your histogram should have a similar setup compared to the one given above, but the counts can be different).

(b) Implement the function *get_dp_histogram(dataset, state, year, epsilon, N)* to construct a DP histogram of monthly COVID-19 positive test counts. Among the parameters, *dataset, state, year* have the same meaning as the previous part. *epsilon* is the DP privacy parameter. N denotes the max number of times an individual can test positively for COVID-19 in one month. *get_dp_histogram* should add appropriate Laplace noise to the histogram and return the resulting histogram. Its return value has the same format as *get_histogram*.

Note that, given *epsilon* and N, a critical part of your job is to decide what the appropriate amount of Laplace noise should be. Assume that neighboring datasets are obtained by addition or removal of one individual.

(c) Let H denote the actual histogram and $\hat{H}$ denote the private histogram. The Average Error in $\hat{H}$ can be measured bin-by-bin (bar-by-bar) as follows:

$$AvgErr(\hat{H}, H) = \frac{\sum_b |\hat{H}[b] - H[b]|}{\text{number of bins}}$$

Implement function *calculate_average_error* to compute $AvgErr$ according to this equation. This function should return a float (the error amount).

(d) You design the following experiment to measure the impact of ε on $AvgErr$. For ε values: $\{0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 1.0\}$, build ε-DP histograms and measure their average errors. Implement this experiment in the *epsilon_experiment* function. Within the function, you should measure error with each ε value 10 times and average the results for statistical significance. The function should return one list of errors (one error value for each ε value).

Use $N = 2$ to report the results of this experiment. In your report, provide a table containing your results. Briefly discuss the relationships between ε and error.

(e) You design the following experiment to measure the impact of N on $AvgErr$. For fixed $\varepsilon = 0.5$ and varying $N = \{1, 2, 4, 8\}$, build ε-DP histograms and measure their average errors. Implement this experiment in the *N_experiment* function. Within this function, you should measure error with each N value 10 times and average the results for statistical significance. The function should return one list of errors (one error value for each N value).

For this experiment, use $\varepsilon = 0.5$. In your report, provide a table containing your results. Briefly discuss the relationships between N and error.

Task 2: We would like to answer the question: "Which month has the highest death count for the given state and year?" using the Exponential Mechanism.

Since some states have high death counts, there can be memory overflows when implementing the Exponential Mechanism in Python. To avoid this issue, it is acceptable to assume this task will be executed only with states that have low death counts, e.g., UT, WY, NV.

(f) Implement function *max_deaths_exponential* to achieve the given task. This function should internally implement the Exponential Mechanism so that a randomized result is returned to satisfy ε-DP. The return value should be the month with the most deaths in the given state and year.

If you are wondering what is the right sensitivity value to use here, we urge you to consider how many times a person can die. ⌣̈

(g) You design the following experiment to measure the impact of ε on the accuracy of the mechanism you implemented in the previous part. For ε values in $\{0.0001, 0.001, 0.01, 0.05, 0.1, 1.0\}$, run *max_deaths_exponential* 10000 times with each ε, and measure its accuracy as the percentage of times it returns the correct answer. Implement this experiment in the *exponential_experiment* function. Provide a graph or table of your results in your report: accuracy vs ε. Briefly discuss your observations regarding the relationship between accuracy and ε.

Part 3: LDP Implementation [35 pts (10 pts * 3 protocols, 5 pts for experimental analysis)]

Recall that one use case of Local Differential Privacy (LDP) is to collect users' browsing information. In this part, you will simulate a toy example of such data collection using a real dataset and the LDP protocols we learned in class.

The MSNBC dataset contains page visits of users who visited msnbc.com on September 28, 1999. In *msnbc-short-ldp.txt* file, we provided you a modified version of the MSNBC dataset in which each line corresponds to one user, and the number in that line corresponds to the visited page category for that user. Page categories are defined as follows:

```
% Different categories found in input file:

frontpage news tech local opinion on-air misc weather msn-news health living business msn-sports sports summary bbs travel
```

You can see that there are a total of 17 categories. The categories are ordered, i.e., frontpage is 1, news is 2, tech is 3, ..., travel is 17. Given these categories, here is how you should interpret the contents of *msnbc-short-ldp.txt*: the 1st line in the dataset contains the number 1, which means user 1 visited the "frontpage" category. The 4th line contains the number 5, which means user 4 visited the "opinion" category, etc. Since the categories are numbered between 1 and 17, each line in *msnbc-short-ldp.txt* will contain an integer between 1 and 17, both inclusive.

Consider a scenario in which each user's visited page category is locally stored on their device, and a data collector wants to learn: For each category $c \in [1,17]$, how many users' category equals c? We will implement this analysis using 3 LDP protocols: GRR, RAPPOR, OUE.

Protocol 1: Generalized Randomized Response (GRR)

(a) Consider that we are using the GRR protocol. Implement *perturb_grr(val, epsilon)* for user-side perturbation. Given a single user's true value *val* (integer), *perturb_grr* returns the output that the user reports to the server.

(b) Implement *estimate_grr(perturbed_values, epsilon)* for server-side estimation. *estimate_grr* takes as input all users' perturbed values in a list and outputs the estimated histogram: For each category $c \in [1,17]$, how many visited page category equals c? *Note: If you end up with negative counts due to LDP noise, that's OK, keep them that way.*

(c) Implement *grr_experiment(dataset, epsilon)*. In this function, simulate data collection for the whole user population, with GRR as the protocol and ε as the privacy parameter. Measure the error of the estimated histogram using the *AvgErr* metric from Part 2. The return value should be *AvgErr* (a single float).

Protocol 2: RAPPOR

(d) Now consider that we are using RAPPOR instead of GRR. There exists an encoding step in RAPPOR before perturbation, which should be implemented in *encode_rappor(val)* function. Given a single user's true value *val* (integer), it should return the encoded bitvector as a list, e.g.: [0, 1, 0, 0, ..., 0]. Then implement *perturb_rappor(encoded_val, epsilon)* for user-side perturbation. Given an encoded bitvector, it should return the perturbed bitvector as a list, e.g.: [1, 1, ..., 0].

(e) Implement *estimate_rappor(perturbed_values, epsilon)*. *estimate_rappor* takes as input all users' perturbed bitvectors as a list [bitvector 1, bitvector 2, ..., bitvector n], and outputs the estimation result: For each category $c \in [1,17]$, how many users' visited page category equals c?

(f) Implement *rappor_experiment(dataset, epsilon)*. In this function, simulate data collection for the whole user population, with RAPPOR as the protocol and ε as the privacy parameter. Measure the error of the estimated histogram using the *AvgErr* metric from Part 2. The return value should be *AvgErr* (a single float).

Protocol 3: Optimized Unary Encoding (OUE)

(g) Now consider that we are using OUE. Implement OUE's encoding step in function: *encode_oue(val)*. Then implement *perturb_oue(encoded_val, epsilon)* for user-side perturbation. Given an encoded bitvector, it should return the perturbed bitvector as a list: [1, 1, ..., 0].

(h) Implement *estimate_oue(perturbed_values, epsilon)*. *estimate_oue* takes as input all the users' perturbed bitvectors as a list [bitvector 1, bitvector 2, ..., bitvector n], and outputs the estimation result: For each category $c \in [1,17]$, how many users' visited page category equals c?

(i) Implement *oue_experiment(dataset, epsilon)*. This function simulates data collection for the whole user population, with OUE as the protocol and ε as the privacy parameter. Measure the error of the estimated histogram using the *AvgErr* metric from Part 2. The return value should be $AvgErr$ (a single float).

Experimental Analysis

(j) Conduct the following experiment: Simulate data collection with GRR, RAPPOR, and OUE for different ε values: $\varepsilon = 0.1$, 0.5, 1.0, 2.0, 4.0, 6.0. Provide a table containing your results in your report. Briefly discuss the results: How are the protocols' errors impacted by ε? Is there a protocol that is always better?

Submission

When you are finished, submit your homework via LearnHub:

- Move all of your relevant files (including Python files, pdf report, etc.) into a folder named **your_KU_ID**.
- Compress this folder into a single zip file. Do not use compression methods other than zip.
- Upload your zip file to LearnHub.

Notes and reminders:

- After submitting, download your submission and double-check that: (i) your files are not corrupted, (ii) your submission contains all the files you intended to submit. If we cannot run your code due to missing files or functions, we cannot give you credit!
- You must upload your code in py files, we do not accept Python notebooks (ipynb extension).
- This homework is an individual assignment. All work needs to be your own. Submissions will be checked for plagiarism (including comparing to previous years' assignments).
- Your report should be a pdf file. Do not submit Word files or other file formats which may only be opened on Windows or Mac.
- Only LearnHub submissions are allowed. Do not e-mail your assignment to the instructor or TAs.
- Do not change the names or parameters of the functions we will grade.
- If your code does not run (e.g., syntax errors) or takes so long that grading it becomes impossible, you may receive 0 for the corresponding part.

Good Luck!