COMP 430/530: Data Privacy and Security - Fall 2022
Homework Assignment # 2

Introduction

This assignment consists of 3 parts. Part 1 is theoretical; you can either write your answer on paper and scan it, or you can type your answer using LaTeX. Parts 2 and 3 contain mostly implementation questions. Here, you can use the Python skeleton files we provide as your starting point. Make sure you adhere to the function names and parameters we are asking for. For questions that require plots or discussion, you should submit your answers in a separate report (pdf file).

Part 1: Privacy Proofs [30 pts (9+9+12)]

Decide if the following algorithms satisfy ε-DP or not. If they do, you must give a full proof similar to the examples we showed in the lectures. If they do not, you must give a full counter-example. Correct answers without justification will receive little or no credit.

(a) Let $A_1, A_2, ..., A_n$ be n independent algorithms which satisfy $\varepsilon_1, \varepsilon_2, ..., \varepsilon_n$-DP respectively. Does their sequential composition (i.e., applying one after the other in successive fashion) satisfy $(\sum_{i=1}^{n} \varepsilon_i)$-DP? *Hint: This is a simplified version of sequential composition in which $A_1, A_2, ..., A_n$ are independent. Use the fact that the algorithms are independent.*

(b) Let algorithm A take as input a dataset D and count the number of records in D. If the number of records is greater than e^{ε}, A prints out: "large". Otherwise, A prints out: "small". Does A satisfy ε-DP?

(c) 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 = ε. Does A satisfy ε-DP?

Hint: How does this mechanism differ from the original Laplace mechanism?

Note: This question may be a bit more challenging than the two above. For full credit, your proof or counter-example must be complete; but we will give partial credit to informal or partial explanations as well.

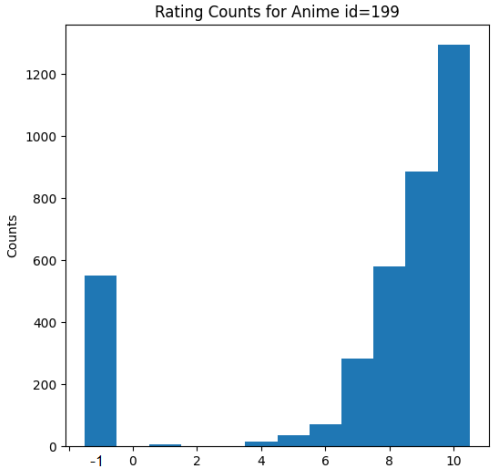
Part 2: DP Implementation [35 pts (a,b,c,d,f each worth 5 pts; e worth 10 pts)]

Consider the Anime Ratings dataset provided to you: *anime-dp.csv*. This dataset is originally from Kaggle, but we modified it for this homework assignment. The dataset contains users' ratings for 12 animes, each column corresponding to one anime. For example, 5114 is the ID of the first anime, 2904 is the ID of the second anime, and so forth. Each row corresponds to a different user's ratings for the animes. Users rate animes between 0 and 10, where 0 means the user strongly disliked the anime and 10 means the user strongly liked the anime. A special -1 rating is given if

the user watched the anime but did submit their rating. An empty field means the user did not watch and therefore did not rate the anime.

Throughout this part, assume that neighboring data sets are obtained by modification of one user's ratings (one row).

Task 1: We would like to construct a histogram to analyze the rating counts of a given anime, such as the anime with id = 199 (Sen to Chihiro no Kamikakushi, Spirited Away). How many users rated it -1? How many rated it 0? How many rated it 1? 2? ... 10? The x axis of the histogram should contain unique rating values in ascending order. The y axis should contain the counts of each rating for anime id = 199.



(a) Implement function *get_histogram(dataset, chosen_anime_id)* to build this histogram non-privately. The function should return a list containing the counts of each rating for the chosen anime id. Notice that by default, chosen_anime_id=199, but your function should work for other anime ids as well. The list should be in ascending order according to the ratings (-1 to 10). Provide a copy of the histogram in your report.

Now, you would like to construct the same histogram differentially privately.

(b) Implement function *get_dp_histogram* that takes as input the original counts (histogram from part a) and the ε privacy parameter. This function should add appropriate amount of Laplace noise to the histogram (you must decide what that appropriate amount should be) and return the resulting histogram. In other words, the return value should be the noisy histogram, which is a list of same length as the original histogram, but all of its elements have been perturbed.

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

$$AvgErr(\hat{H}, H) = \frac{\sum_b |\hat{H}[b] - H[b]|}{\text{number of bins}} \qquad MSE(\hat{H}, H) = \frac{\sum_b (\hat{H}[b] - H[b])^2}{\text{number of bins}}$$

Implement functions *calculate_average_error* and *calculate_mean_squared_error* to compute $AvgErr$ and MSE according to these equations. Both functions should return a float (the error amount).

(d) You design the following experiment to measure the impact of ε on $AvgErr$ and MSE. 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 and mean squared errors. You should repeat each experiment 40 times and average their results for statistical significance. The function should return two error lists: one contains mean squared errors and the other contains average errors. Implement this experiment in the

epsilon_experiment function. In your report, provide a table or graph of your results and briefly discuss the relationships between ε and error.

Task 2: We would like to answer the question: "Which anime received the highest number of 10 ratings?" using the Exponential mechanism.

(e) Implement function *most_10rated_exponential* to achieve this. This function should internally implement the Exponential mechanism so that a randomized result will be returned to achieve ε-DP. The return value should be the id of the anime.

(f) 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 $\{0.001, 0.005, 0.01, 0.03, 0.05, 0.1\}$, run *most_10rated_exponential* 1000 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. In your report, provide a graph or table of your results: 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 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 most 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 most. The 4th line contains the number 5, which means user 4 visited the "opinion" category the most, 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 most visited page category is locally stored on their device, and the server (data collector) wants to learn: For each category $c \in$ [1,17], how many users' category equals c? We will implement this analysis using LDP protocols.

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 users' most visited page category equals c?

(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 Simple 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].

(e) 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].

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

(g) 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)

(h) Now consider that we are using OUE. Implement OUE's encoding step in function: *encode_oue(val)*.

(i) 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].

(j) 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 estimated histogram: For each category $c \in$ [1,17], how many users' most visited page category equals c?

(k) Implement *oue_experiment(dataset, epsilon)*. In this function, simulate 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

(l) Conduct the following experiment: Simulate data collection with GRR, Simple 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 assignment via Blackboard:

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

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.
- Your report should be a pdf file. Do not submit Word files (or others which may only be opened on Windows or Mac).
- Only Blackboard 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!