COMP 430/530: Data Privacy and Security - Fall 2021
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 on computer using LaTeX. Parts 2 and 3 contain mostly implementation questions. You can use the Python skeleton files we provide as your starting point, but you are not forced to do so, as long as you adhere to the same function names and parameters we are asking for. For questions that require plots or discussion, you should submit your answers in a separate report (one pdf file).

Part 1: Privacy Proofs [30 pts]

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:

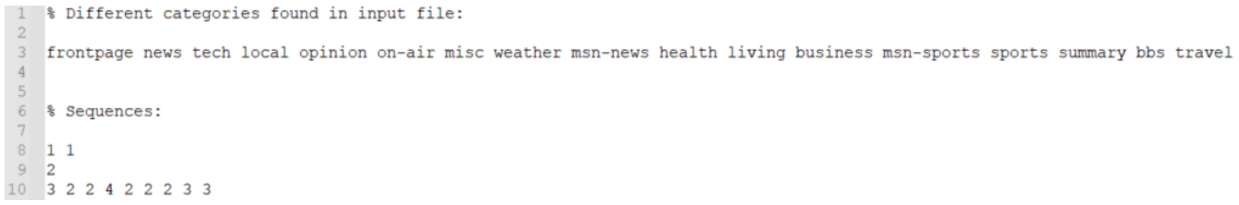
$$\Pr[\Psi(v) = y] = \frac{e^{\frac{-\alpha \cdot d(v,y)}{2}}}{\sum\limits_{z \in \mathcal{U}} e^{\frac{-\alpha \cdot d(v,z)}{2}}}$$

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 [40 pts (a=5, b=5, c+d+e+f+g=20, h+i=10 pts)]

Consider the MSNBC dataset provided to you. This dataset contains page visits of users who visited msnbc.com on September 28, 1999. Each row in the dataset corresponds to the page views of one user. Page views are recorded at the granularity of page category rather than URL. For example, if you find sequence "1 2 1 3" in one of the rows, that means a user first viewed a page under the *frontpage* category, then *news* category, then *frontpage* category, then *tech* category.

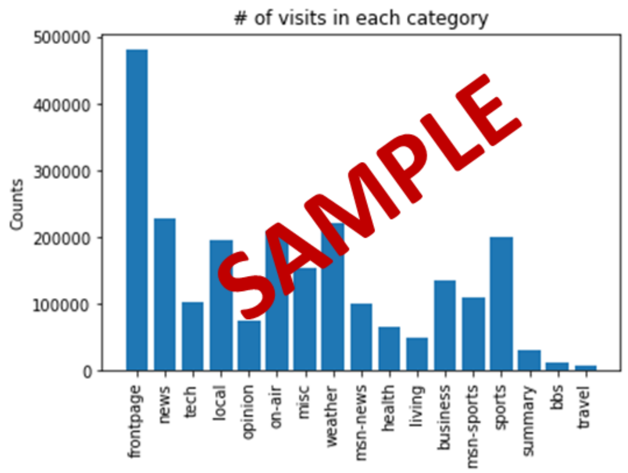


```
1  % Different categories found in input file:
2
3  frontpage news tech local opinion on-air misc weather msn-news health living business msn-sports sports summary bbs travel
4
5
6  % Sequences:
7
8  1 1
9  2
10 3 2 2 4 2 2 2 3 3
```

The above screenshot is from the MSNBC dataset. You can see that there are a total of 17 categories (listed on line 3). The categories are ordered, i.e., *frontpage* is 1, *news* is 2, *tech* is 3, ..., *travel* is 17. Users' page view sequences start on line 8, and from there onwards, there is one sequence per line.

Throughout this part, assume that neighboring datasets are obtained by addition or removal of one user's sequence (one row), i.e., $D' = D \cup s$ or $D' = D \setminus s$ where s is one sequence.

Task: We would like to construct a histogram to analyze which page category was visited most. The x axis of the histogram should contain page categories (ordered 1-17). The y axis should contain the total number of page visits for that category.



(a) Implement function *get_histogram(dataset)* to build this histogram non-privately. The function should save the visual histogram to a file named **np-histogram.png** and return an ordered list containing the counts of each page category [123, 383, 541, ..., 915] for page categories *frontpage, news, tech, ..., travel* respectively. Provide a copy of the histogram in your report.

(b) *add_laplace_noise(real_answer, sensitivity, epsilon)* is a generic function (independent of the histogram, but applicable to it) which adds Laplace noise to a query's real answer according to its sensitivity and epsilon. *real_answer* is a list of floats or integers. Implement this function. Its return value should be *noisy_answer*, which is a list of same size as *real_answer*, but all of its elements have been perturbed.

Now, you would like to construct the same histogram differentially privately. There is initially no upper bound on how long each user's sequence can be, but you know that this causes high sensitivity for the histogram, which is a problem. You further observe that in reality, most users' sequences are quite short. You therefore decide to use the "sensitivity bounding" trick by truncating users' sequences.

(c) Implement function *truncate(dataset, n)* where *dataset* is list of lists representation of MSNBC, and n is the max length of each truncated sequence. For each sequence in the dataset, the function keeps the first n items and removes the rest. Function returns truncated dataset.

(d) Implement function *get_dp_histogram* that takes as input the original MSNBC dataset, truncation parameter n, privacy parameter ε. The function should truncate the dataset with parameter n, build and return differentially private histogram as a list. When implementing this function, you can make use of some of the previous functions.

(e) 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:

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

Implement function *calculate_average_error(actual_hist, noisy_hist)* that measures error according to the above equation.

(f) You design the following experiment to measure the impact of the truncation parameter n on Err. Fix $\varepsilon = 0.01$. For n values between 1 and 101 in increments of 5, build 0.01-differentially private histograms for each n and measure their Err. Since each histogram is randomized, you should repeat the experiment with each n 30 times and average the Errs. Implement this experiment in the *n_experiment* function. In your report, provide a graph of your results: Err vs n. Discuss which n value is best, why this might be the case, and what trade-offs exist in choosing n (i.e., why can large n be good? why can it be bad?).

(g) You design the following experiment to measure the impact of ε on Err. Fix $n = 50$. For ε values: $\{0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 1.0\}$, build ε-DP histograms and measure their Err. Again, you should repeat each experiment 30 times and average their results for statistical significance. Implement this experiment in the *epsilon_experiment* function. In your report, provide a graph of your results: Err vs ε. Briefly discuss which ε value is best and why.

For the remainder of Part 2: Implement function *extract* that extracts the first 1000 sequences from MSNBC dataset and truncate them to $n = 1$. You will use this smaller version of the dataset in the rest of this part.

We would like to answer the question: "Which category (1-17) received the highest number of page visits?" using the Exponential mechanism.

(h) Implement function *most_visited_exponential(smaller_dataset, epsilon)* to achieve this. This function should implement the Exponential mechanism. It should return 1 for frontpage, 2 for news, 3 for tech, etc.

(i) You design the following experiment to measure the impact of ε on the accuracy of the mechanism you implemented in part h. For ε values in $\{0.001, 0.005, 0.01, 0.03, 0.05, 0.1\}$, run *most_visited_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.

Part 3: LDP Implementation [30 pts (a+b+c=10, d+e+f+g=10, h=10 pts)]

In December 2017, Microsoft shared a blog post (and published a research paper) saying that they will start using Local Differential Privacy (LDP) to collect telemetry data from users' devices, e.g., how much time each user spends using some software[1]. In this part, you will simulate a toy example of such data collection using synthetic data and the LDP protocols we learned in class.

Consider that we are collecting users' *daily screen time* for a particular day. Before collection, screen times are discretized by rounding to the nearest full hour. Thus, each user's true data is an integer between $[0, 24]$, both inclusive. The file *daily_time.txt* is given to you, where each line corresponds to one user's daily screen time.

Say that each user's daily screen time is locally stored on their device, and the server (data collector) wants to learn a histogram of users' daily screen times using LDP.

(a) First, 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*, *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 hour $h \in$ [0,24], how many users' daily screen time equals h?

(c) Implement *grr_experiment(dataset, epsilon)*. In this function, simulate data collection with GRR for the given ε value. Measure the error of the estimated histogram using the error metric Err from Part 2. The return value should be Err (a single float).

(d) Now, consider that we are using Simple RAPPOR instead of GRR. However, there exists an additional step for encoding in RAPPOR. Implement *encode_rappor(val)* function to achieve this. Given a single value, it should return encoded bitvector of a single user as a list: [0, 1, 0, 0, ..., 0].

(e) Implement *perturb_rappor(encoded_val, epsilon)* for user-side perturbation. Given an encoded bit vector, it should return the perturbed bit vector of a single user as a list: [1, 1, ..., 0].

(f) Implement *estimate_rappor(perturbed_values, epsilon)*. *estimate_rappor* takes as input all the users' perturbed bit vectors as a list [bit vector 1, bitvector 2, ..., bitvector n], and outputs the estimated histogram: For each hour $h \in$ [0,24], how many users' daily screen time equals h?

(g) Implement *rappor_experiment(dataset, epsilon)*. In this function, simulate data collection with RAPPOR for a single ε value. Measure the error of the estimated histogram using the error metric Err from Part 2. The return value should be Err.

(h) Design and conduct the following experiment: Simulate data collection with GRR and Simple RAPPOR for different ε values: ε = 0.1, 0.5, 1.0, 2.0, 4.0, 6.0. Provide a graph or table of your results in your report. Briefly discuss the results: For these two protocols, how are their errors impacted by ε? Which protocol is better: GRR or Simple RAPPOR?

[1] `https://www.microsoft.com/en-us/research/blog/collecting-telemetry-data-privately/`

Submission

When you are finished, submit your assignment via Blackboard as follows.

- 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.

Some 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.
- This homework is an individual assignment. All work needs to be your own. Submissions will be checked for plagiarism.
- Your report should be readable on any computer. We cannot grade files that are only readable on a Mac.
- Submit only through Blackboard. Do not e-mail your assignment to the instructor or the TAs. Make sure to submit ahead of time to avoid Blackboard-related problems or delays.
- Do not submit data files such as *msnbc.dat* or *daily_time.txt*.
- 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!