lab11_student_version

December 1, 2021

1 Lab 11: Differential Privacy

Welcome to the 11th (and final) DS102 lab!!

The goal of this lab is to gain a better understanding of differential privacy. We will observe what happens after the Laplace mechanism is applied to an estimator, which was discussed in last Tuesday's lecture. This demonstration is related to an experiment done by Duchi et al. 2017.

The code you need to write is commented out with a message "TODO: fill …". There is additional documentation for each part as you go along.

1.1 Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the labs, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** in the cell below.

Collaborators:

1.2 Gradescope Submission

To submit this assignment, rerun the notebook from scratch (by selecting Kernel > Restart & Run all), and then print as a pdf (File > download as > pdf) and submit it to Gradescope.

This assignment should be completed and submitted before Wednesday, December 1st, 2021 at 11:59 PM. PT

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
%matplotlib inline
```

2 National Estimates of Drug-Related Emergency Department Visits (NEDREDV)

In this lab, we will analyze drug use data from the National Estimates of Drug-Related Emergency Department Visits (NEDREDV).

The NEDREDV dataset tracks the number of hospital emergency department visits related to drug usage in a given year. We look at data from 2004 and consider the number of hospital admissions for several common drugs: *Alcohol, Cocaine, Heroin, Marijuana, Stimulants, Amphetamines, Methamphetamine, MDMA, LSD, PCP, Antidepressants, Antipsychotics, Miscellaneous hallucinogens, Inhalants, lithium, Opiates, Opiates unspecified, Narcotic analgesics, Buprenorphine, Codeine, Fentanyl, Hydrocodone, Methadone, Morphine, Oxycodone, Ibuprofen, and Muscle relaxants.*

The NEDREDV dataset that we import includes the $d = 27$ drugs, and the observed probability that a person admitted to the hospital for drug abuse in 2004 used drug j. **Note that a person admitted to the hospital could have used multiple drugs simultaneously, so the probabilities do not sum to 1**.

[2]:
```
nedredv_df = pd.read_csv('nedredv_probs.csv')
```

[3]:
```
nedredv_df
```

[3]:
```
                     Substance  Probability
0                      Alcohol     0.530527
1                      Cocaine     0.325744
2                       Heroin     0.109529
3                    Marijuana     0.218641
4                   Stimulants     0.101317
5                 Amphetamines     0.027766
6              Methamphetamine     0.077830
7                         MDMA     0.007762
8                          LSD     0.001503
9                          PCP     0.027706
10             Antidepressants     0.094340
11              Antipsychotics     0.052264
12  Miscellaneous_hallucinogens     0.001879
13                   Inhalants     0.009668
14                     lithium     0.007860
15                     Opiates     0.174578
16         Opiates_unspecified     0.032600
17         Narcotic_analgesics     0.147420
18               Buprenorphine     0.001014
19                     Codeine     0.009444
20                    Fentanyl     0.007097
21                 Hydrocodone     0.053049
22                   Methadone     0.029202
23                    Morphine     0.011054
24                   Oxycodone     0.051556
25                   Ibuprofen     0.027810
26            Muscle_relaxants     0.032265
```

2.1 Simulating a privacy-sensitive dataset

The NEDREDV dataset itself does not contain the actual drug usage of each individual person admitted to the hospital: it only contains rates of the total number of people admitted to the hospital for using each drug. For the purposes of this lab, we will instead *simulate* a privacy-sensitive dataset that contains the drug usage of the individuals admitted to the hospital. We will generate a dataset $X = \{X_1, ..., X_N\}$ where each $X_i \in \{0, 1\}^d$ represents an individual admitted to the hospital, and $X_{i,j}$ is 1 if the individual abuses drug j and 0 otherwise. Since drug use is a sensitive topic, it would certainly be a privacy problem if such a dataset X containing the drug usage of individuals admitted to the hospital were made public (and it would likely violate HIPAA).

To generate this privacy-sensitive dataset, let p_j be the observed probability that a person admitted to the hospital used drug j according to the NEDREDV data from 2004. We draw

$$X_{i,j} \sim Bernoulli(p_j)$$

independently for all $i = 1, ..., N$ and for all $j = 1, ..., d$. This results in a set of hypothetical individuals $X_1, ..., X_N$ where the marginal counts $\frac{1}{N}\sum_{i=1}^{N} X_{i,j}$ yield approximately the correct drug use frequencies to match the NEDREDV data.

[4]:
```python
def simulate_private_data(nedredv_df, N=30000, random_seed = None):
    """Simulates the privacy-sensitive dataset with individual drug usage.

    Inputs:
        nedredv_df : dataframe containing the drug and the probability that an␣
↪admittee used the drug.
        N : number of individuals to generate.
        random_seed : int, random seed for experimental reproducibility

    Returns:
        X_df : dataframe containing N rows where each row corresponds to an␣
↪admitted individual,
            and a 1 in a column corresponding to a given drug means that the␣
↪individual used that drug.
    """
    X = {}
    for index, row in nedredv_df.iterrows():
        drug_name = row['Substance']
        observed_probability = row['Probability']
        X_row = bernoulli.rvs(observed_probability, size=N, random_state = np.
↪random.RandomState(seed=random_seed))
        X[drug_name] = X_row
        X_df = pd.DataFrame(X)
    return X_df
```

[5]:
```python
X_df = simulate_private_data(nedredv_df, random_seed = 22)
X_df.head()
```

[5]:
```
   Alcohol  Cocaine  Heroin  Marijuana  Stimulants  Amphetamines  \
0        1        0       0          0           0             0
1        1        0       0          0           0             0
2        1        0       0          0           0             0
3        0        1       0          1           0             0
4        1        0       0          0           0             0

   Methamphetamine  MDMA  LSD  PCP  …  Narcotic_analgesics  Buprenorphine  \
0                0     0    0    0  …                    0              0
1                0     0    0    0  …                    0              0
2                0     0    0    0  …                    0              0
3                0     0    0    0  …                    1              0
4                0     0    0    0  …                    0              0

   Codeine  Fentanyl  Hydrocodone  Methadone  Morphine  Oxycodone  Ibuprofen  \
0        0         0            0          0         0          0          0
1        0         0            0          0         0          0          0
2        0         0            0          0         0          0          0
3        0         0            0          0         0          0          0
4        0         0            0          0         0          0          0

   Muscle_relaxants
0                 0
1                 0
2                 0
3                 0
4                 0

[5 rows x 27 columns]
```

Notice that each row of the dataframe X_df corresponds to an individual, X_i, and a 1 in a column corresponding to a given drug means that the individual used that drug. An individual admitted to the hospital could have used multiple drugs simultaneously.

3 Goal: Mean estimation with differential privacy

From now on, we will treat the dataset that we just generated as ground truth. We will seek to analyze statistics about this data without hurting the privacy of the individuals in the dataset that we just generated.

Specifically, the statistic we want to estimate is the mean of the population: $\theta = E[X]$. But, the catch is that we want to estimate this mean in a **differentially private** way.

3.1 The true mean

Based on the way the data was generated, we know that the true mean of the distribution that the samples were drawn from is the original probabilities from the NEDREDV dataset. Our goal will be to estimate this true mean from the dataset that we generated in a differentially private way.

```
[6]: TRUE_MEAN = nedredv_df['Probability'].to_numpy()
     print(nedredv_df)
```

```
                     Substance  Probability
0                      Alcohol     0.530527
1                      Cocaine     0.325744
2                       Heroin     0.109529
3                    Marijuana     0.218641
4                   Stimulants     0.101317
5                 Amphetamines     0.027766
6              Methamphetamine     0.077830
7                         MDMA     0.007762
8                          LSD     0.001503
9                          PCP     0.027706
10             Antidepressants     0.094340
11              Antipsychotics     0.052264
12  Miscellaneous_hallucinogens     0.001879
13                   Inhalants     0.009668
14                     lithium     0.007860
15                     Opiates     0.174578
16         Opiates_unspecified     0.032600
17         Narcotic_analgesics     0.147420
18               Buprenorphine     0.001014
19                     Codeine     0.009444
20                    Fentanyl     0.007097
21                  Hydrocodone     0.053049
22                   Methadone     0.029202
23                    Morphine     0.011054
24                   Oxycodone     0.051556
25                   Ibuprofen     0.027810
26            Muscle_relaxants     0.032265
```

3.2 Differential privacy

The idea behind differential privacy is that any given individual should be willing to participate in the statistical analysis because their participation in the study does not change the outcome of the study by very much; their personal information cannot be recovered from the results of either removing or adding them to the study. This applies to the individuals in the dataset we generated: removing any individual from the data should not change our mean estimate that much. Otherwise, it may be possible to recover the drug usage of an individual based on a mean estimate that includes the individual and another mean estimate that doesn't include the individual.

For two datasets X and X' which differ in only one entry (e.g., differing in one individual), an **ϵ-differentially private algorithm $\mathcal{A}$** satisfies:

$$\mathbb{P}\{\mathcal{A}(X) = a\} \leq e^{\epsilon}\mathbb{P}\{\mathcal{A}(X') = a\},$$

for all possible output values a of the algorithm $\mathcal{A}$. In words, the probability of seeing any given output of a differentially private algorithm doesn't change much by replacing any one entry in the dataset.

We will explore three algorithms for estimating the mean in this lab:

1. **Algorithm 1: Non-private:** we will simply take the sample mean of the given data X,

$$\hat{\theta} = \mathcal{A}(X) = f(X) = \frac{1}{N}\sum_{i=1}^{N} X_i.$$

   This is not differentially private: we can recover the drug usage of an individual if we estimate the mean before and after removing the individual.

2. **Algorithm 2: Laplace mechanism:** To introduce differential privacy, we can apply the Laplace mechanism. Given the non-private estimator $f(X)$, we can add noise ξ_ϵ:

$$\hat{\theta} = \mathcal{A}(X) = f(X) + \xi_\epsilon = \left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) + \xi_\epsilon.$$

   We will go over this algorithm in more detail later in the lab.

3. **Algorithm 3: Locally differentially private Laplace mechanism:** another way to introduce differential privacy is to make the data locally differentially private. In the above Algorithm 2, we added a single noise parameter ξ_ϵ to the non-private estimate $f(X)$. Rather than adding noise to the aggregated $f(X)$, we could also add noise to each sensitive bit individually, X_i.

$$\hat{\theta} = \mathcal{A}(X) = f(X + \xi_\epsilon) = \frac{1}{N}\sum_{i=1}^{N}(X_i + \xi_\epsilon^i).$$

   We will also go over this algorithm in more detail later in the lab.

Both Algorithm 2 and Algorithm 3 result in estimators $\hat{\theta}$ that are ϵ-differentially private. The difference between Algorithm 2 and Algorithm 3 is that Algorithm 3 introduces more noise overall by introducing noise ξ_ϵ^i into each row. However, the local approach of Algorithm 3 ensures privacy even if we don't trust the person or program calculating $f(X)$ in Algorithm 2.

4 Question 1. Algorithm 1: Non-private

We will now implement the three algorithms, and compare how well they accomplish the task of mean estimation. We'll start with Algorithm 1.

For Algorithm 1, the obvious algorithm for mean estimation is to simply take the mean of the samples, X_i:

$$\hat{\theta} = \mathcal{A}(X) = f(X) = \frac{1}{N}\sum_{i=1}^{N} X_i$$

However, this is clearly not differentially private: we can recover the drug usage of an individual if we estimate the mean before and after removing the individual.

4.1 1.a. Implement Algorithm 1

First, we need to implement the calculation of $\hat{\theta}$ using Algorithm 1.

[7]:
```
# TODO: estimate the mean of the data using the non-private Algorithm 1.
def alg_1_estimate(input_X_df):
    """Estimates the mean of the data using the non-private Algorithm 1.

    Inputs:
        input_X_df : dataframe where each row corresponds to an individual,
            and a 1 in a column corresponding to a given drug means that
            the individual used that drug.

    Outputs:
        mean_estimate : d-dimensional numpy array containing mean of all of the␣
↪rows in X_df.

    """
    mean_estimate = np.array(np.mean(input_X_df)) # mean
    return mean_estimate
```

[8]:
```
# Validation tests (do not modify)
test_input = simulate_private_data(nedredv_df, N=100, random_seed = 0)
test_output = alg_1_estimate(test_input)
true_output= np.array([0.54, 0.26, 0.1 , 0.17, 0.09, 0.04, 0.09, 0.  , 0.  , 0.
↪04, 0.09,
                       0.05, 0.  , 0.  , 0.  , 0.14, 0.04, 0.11, 0.  , 0.  , 0.␣
↪ , 0.05,
                       0.04, 0.  , 0.05, 0.04, 0.04])
assert np.max(abs(test_output - true_output)) < 0.01
print("Test passed!")
```

```
Test passed!
```

[9]:
```
# Print the estimates according to Algoritm 1
nedredv_df['Algo_1_estimates'] = alg_1_estimate(X_df)
nedredv_df
```

[9]:
```
                 Substance  Probability  Algo_1_estimates
0                  Alcohol     0.530527          0.527567
1                  Cocaine     0.325744          0.328867
2                   Heroin     0.109529          0.109433
3                Marijuana     0.218641          0.220433
4               Stimulants     0.101317          0.100267
5             Amphetamines     0.027766          0.027333
6          Methamphetamine     0.077830          0.077867
7                     MDMA     0.007762          0.007567
8                      LSD     0.001503          0.001300
```

```
9                           PCP    0.027706          0.027233
10               Antidepressants    0.094340          0.093300
11                Antipsychotics    0.052264          0.052433
12  Miscellaneous_hallucinogens    0.001879          0.001667
13                     Inhalants    0.009668          0.009333
14                       lithium    0.007860          0.007700
15                       Opiates    0.174578          0.175867
16           Opiates_unspecified    0.032600          0.032133
17           Narcotic_analgesics    0.147420          0.148033
18                 Buprenorphine    0.001014          0.000667
19                       Codeine    0.009444          0.009133
20                      Fentanyl    0.007097          0.006933
21                   Hydrocodone    0.053049          0.053233
22                     Methadone    0.029202          0.028600
23                      Morphine    0.011054          0.010500
24                     Oxycodone    0.051556          0.051633
25                     Ibuprofen    0.027810          0.027367
26              Muscle_relaxants    0.032265          0.031933
```

4.2 1.b. Compute the max error of the mean estimate

To judge how good our mean estimate was, we will use the max error, or the infinity-norm:

$$||\hat{\theta} - \theta||_\infty = \max_i |\hat{\theta}_i - \theta_i|$$

This just finds the max difference between any two coordinates of the true mean θ and the estimated mean $\hat{\theta}$.

Now, we will implement the max error function, and calculate the max error of Algorithm 1.

[10]:
```python
# TODO: compute the max error between the estimated mean and the true mean.
def max_error(estimated_mean, true_mean):
    """Computes the maximum error between the estimated mean and the true mean.

    Inputs:
        estimated_mean: numpy array of length d containing the estimated mean.
        true_mean: numpy array of length d containing the true mean.

    Outpus:
        max_error: the/'' max error between the estimated_mean and true_mean.
            This should be the max of the absolute value of all of the␣
↪coordinates
            of estimated_mean - true_mean.
    """
    max_err = np.array(max(estimated_mean - true_mean)) # TODO: fill in
    return max_err
```

```
[11]: # Validation tests: Do not modify
test_input=[np.array([0.34, 0.66, 0.41]),np.array([0.36, 0.58, 0.45])]
assert np.abs(max_error(*test_input)-0.08)<0.0001
print("Test passed!")
```

```
Test passed!
```

```
[12]: print("Max error of Algorithm 1: {:.6f}".format(max_error(alg_1_estimate(X_df),␣
 ↪TRUE_MEAN)))
```

```
Max error of Algorithm 1: 0.003123
```

5 Question 2. Algorithm 2: Laplace mechanism

To introduce differential privacy, we can apply the Laplace mechanism that we will go over in Discussion 13. Given the non-private estimator $f(X)$, we can add noise ξ_ϵ:

$$\hat{\theta} = \mathcal{A}(X) = f(X) + \xi_\epsilon = \left(\frac{1}{N} \sum_{i=1}^{N} X_i \right) + \xi_\epsilon$$

$\xi_\epsilon \in \mathbb{R}^d$ has independent coordinates, each distributed according to the zero-mean Laplace distribution with scale parameter $\frac{\Delta_f}{\epsilon}$, denoted $\text{Lap}(0, \frac{\Delta_f}{\epsilon})$.

Δ_f is the sensitivity of the function f, defined as

$$\Delta_f = \max_{\text{neighboring } X, X'} ||f(X) - f(X')||_1,$$

where $||.||_1$ is the 1-norm. We need to use the 1-norm here because $f(X) \in \mathbb{R}^d$.

Solving for this,

$$\begin{aligned} \Delta_f &= \max_{\text{neighboring } X, X'} ||f(X) - f(X')||_1 \\ &= \max_{\text{neighboring } X, X'} \left|\left| \frac{1}{N} \sum_{i=1}^{N} X_i - \frac{1}{N} \sum_{i=1}^{N} X'_i \right|\right|_1 \\ &= \frac{d}{N} \end{aligned}$$

In this week's discussion, we will show that the above algorithm $\mathcal{A}(X)$ is ϵ-differentially private.

5.1 Implement Algorithm 2

Calculate $\hat{\theta}$ using Algorithm 2 above.

Plugging in the calculation for Δ_f above, we have $\xi_\epsilon \in \mathbb{R}^d$ has independent coordinates, each distributed according to the zero-mean Laplace distribution with scale parameter $\frac{d}{N\epsilon}$, denoted $\text{Lap}(0, \frac{d}{N\epsilon})$.

[13]:
```
# TODO: estimate the mean of the data using Algorithm 2.
def alg_2_estimate(input_X_df, epsilon=0.5, random_seed = None):
    """Estimates the mean of the data using Algorithm 2.

    Inputs:
        input_X_df: dataframe where each row corresponds to an individual,
            and a 1 in a column corresponding to a given drug means that
            the individual used that drug.
        epsilon: differential privacy parameter.
        random_seed: int, random seed for experimental reproducibility

    Outputs:
        mean_estimate: d-dimensional numpy array containing the mean estimate.

    """
    random_state = np.random.RandomState(seed=random_seed)
    d = len(input_X_df.columns)
    N = len(input_X_df)
    laplace_scale = (d/N)/epsilon # TODO

    xi = random_state.laplace(0, laplace_scale, size=d)
    mean_estimate = np.array(np.mean(input_X_df) + xi) # TODO
    return mean_estimate
```

[14]:
```
# Validation tests (do not modify)
test_input = simulate_private_data(nedredv_df, N=1000, random_seed = 1)
test_output = alg_2_estimate(test_input, random_seed = 6)
true_output= np.array([ 0.61118555,  0.30388517,  0.16553917,  0.08185385,  0.
 ↪01907432,
                        0.04238588,  0.07632029, -0.00456863, -0.02156009,  0.
 ↪04617881,
                        0.08586842,  0.08546348,  0.00198393,  0.01626853,  0.
 ↪02354852,
                        0.38347409,  0.08912664,  0.1377036 ,  0.07541032,  0.
 ↪06230894,
                       -0.11471126,  0.08304802,  0.08106895,  0.04457083,  0.
 ↪07825384,
                        0.0356127 , -0.04093591])
assert np.max(abs(test_output - true_output)) < 0.01
print("Test passed!")
```

```
Test passed!
```

[15]:
```
# Print the estimates according to Algoritm 2
alg_2_estimates = alg_2_estimate(X_df, random_seed = 17)
nedredv_df['Algo_2_estimates'] = alg_2_estimates
nedredv_df
```

[15]:
```
                       Substance  Probability  Algo_1_estimates  \
0                        Alcohol     0.530527          0.527567
1                        Cocaine     0.325744          0.328867
2                         Heroin     0.109529          0.109433
3                      Marijuana     0.218641          0.220433
4                     Stimulants     0.101317          0.100267
5                    Amphetamines     0.027766          0.027333
6                 Methamphetamine     0.077830          0.077867
7                           MDMA     0.007762          0.007567
8                            LSD     0.001503          0.001300
9                            PCP     0.027706          0.027233
10                Antidepressants     0.094340          0.093300
11                  Antipsychotics     0.052264          0.052433
12   Miscellaneous_hallucinogens     0.001879          0.001667
13                      Inhalants     0.009668          0.009333
14                        lithium     0.007860          0.007700
15                        Opiates     0.174578          0.175867
16            Opiates_unspecified     0.032600          0.032133
17             Narcotic_analgesics     0.147420          0.148033
18                  Buprenorphine     0.001014          0.000667
19                        Codeine     0.009444          0.009133
20                       Fentanyl     0.007097          0.006933
21                    Hydrocodone     0.053049          0.053233
22                      Methadone     0.029202          0.028600
23                       Morphine     0.011054          0.010500
24                      Oxycodone     0.051556          0.051633
25                      Ibuprofen     0.027810          0.027367
26                Muscle_relaxants     0.032265          0.031933

    Algo_2_estimates
0           0.526615
1           0.328980
2           0.107706
3           0.216840
4           0.101803
5           0.028008
6           0.078446
7           0.007862
8          -0.003289
9           0.026631
10          0.097296
11          0.048618
12          0.004011
13          0.011862
14          0.003598
15          0.176521
16          0.032330
```

```
17          0.148424
18          0.000606
19          0.008109
20          0.006000
21          0.053470
22          0.028180
23          0.012050
24          0.051313
25          0.025125
26          0.029777
```

[16]:
```python
# Print the max error of Algorithm 2
print("Max error of Algorithm 2: {:.6f}".format(max_error(alg_2_estimates,␣
 ↪TRUE_MEAN)))
```

```
Max error of Algorithm 2: 0.003236
```

6 Question 3. Algorithm 3: Locally differentially private Laplace mechanism

Finally, another way to introduce differential privacy is to make the data locally differentially private. In the above Algorithm 2, we added a single noise parameter ξ_ϵ to the non-private estimate $f(X)$. Rather than adding noise to the aggregated $f(X)$, we could also add noise to each sensitive bit individually, X_i.

$$\hat{\theta} = \mathcal{A}(X) = f(X + \xi_\epsilon) = \frac{1}{N}\sum_{i=1}^{N}(X_i + \xi_\epsilon^i)$$

Here, $\xi_\epsilon \in \mathbb{R}^{N\times d}$ is making each row in the data differentially private before it even reaches the function $f(X)$. For each individual row X_i, $\xi_\epsilon^i \in \mathbb{R}^d$ has independent coordinates, each distributed according to the zero-mean Laplace distribution with parameter $\frac{\Delta_{X_i}}{\epsilon}$, denoted Lap$(0, \frac{\Delta_{X_i}}{\epsilon})$.

Δ_{X_i} is the sensitivity of changing a single row X_i:

$$\Delta_{X_i} = \max_{X_i \neq X_i'} ||X_i - X_i'||_1 = d$$

It can be similarly shown that adding noise ξ_ϵ^i to each row X_i makes each row itself differentially private, and by the Composition Guarantees for Differential Privacy, this locally differentially private algorithm $\mathcal{A}(X)$ is also ϵ-differentially private.

The difference between Algorithm 2 and Algorithm 3 is that Algorithm 3 introduces more noise overall by introducing noise ξ_ϵ^i into each row. However, the local approach of Algorithm 3 ensures privacy even if we don't trust the person or program calculating $f(X)$ in Algorithm 2.

6.1 3.a. Implement Algorithm 3

Calculate $\hat{\theta}$ using Algorithm 3 above.

Plugging in the calculation for Δ_{X_i} above, we have $\xi_\epsilon^i \in \mathbb{R}^d$ has independent coordinates, each distributed according to the zero-mean Laplace distribution with parameter $\frac{d}{\epsilon}$, denoted $\mathrm{Lap}(0, \frac{d}{\epsilon})$.

[17]:
```
# TODO: estimate the mean of the data using Algorithm 3.
def alg_3_estimate(input_X_df, epsilon=0.5, random_seed = None):
    """Estimates the mean of the data using Algorithm 3.

    Inputs:
        input_X_df: dataframe where each row corresponds to an individual,
            and a 1 in a column corresponding to a given drug means that
            the individual used that drug.
        epsilon: differential privacy parameter.
        random_seed: int, random seed for experimental reproducibility

    Outputs:
        mean_estimate: d-dimensional numpy array containing the mean estimate. ␣
 ↪
    """
    random_state = np.random.RandomState(random_seed)
    d = len(input_X_df.columns)
    N = len(input_X_df)
    laplace_scale = d/epsilon # TODO

    # Adds the xi_i noise to each row X_i.
    X = input_X_df.to_numpy(dtype=float)
    for i in range(len(X)):
        xi_i = random_state.laplace(0, laplace_scale, size=d)
        X[i] = X[i] + xi_i # TODO

    mean_estimate = np.mean(X, axis=0)
    return mean_estimate
```

[18]:
```
# Validation tests (do not modify)
test_input = simulate_private_data(nedredv_df, N=10000, random_seed = 8)
test_output = alg_3_estimate(test_input, random_seed = 2)
true_output= np.array([-0.40967514,  0.07351822, -0.4317238 ,  1.00062335, -0.
 ↪5200155 ,
                       -0.89679615, -0.65737125, -0.29722826,  0.84077436,  0.
 ↪40867117,
                       -1.00796038, -0.43696672,  0.01821487, -1.80613563, -0.
 ↪90836173,
                       -0.17835337,  0.72904829, -0.72107227,  0.15482221, -0.
 ↪55084658,
                        -0.31396992, -0.3488769 ,  0.77788347,  0.45583122,  0.
↪05131975,
                         0.77705583,  0.56286043])
assert np.max(abs(test_output - true_output)) < 0.01
print("Test passed!")
```

```
Test passed!
```

[19]:
```
# Print the estimates according to Algoritm 3
alg_3_estimates = alg_3_estimate(X_df, random_seed = 11)
nedredv_df['Algo_3_estimates'] = alg_3_estimates
nedredv_df
```

[19]:
```
                   Substance  Probability  Algo_1_estimates  \
0                    Alcohol     0.530527          0.527567
1                    Cocaine     0.325744          0.328867
2                     Heroin     0.109529          0.109433
3                  Marijuana     0.218641          0.220433
4                 Stimulants     0.101317          0.100267
5               Amphetamines     0.027766          0.027333
6            Methamphetamine     0.077830          0.077867
7                       MDMA     0.007762          0.007567
8                        LSD     0.001503          0.001300
9                        PCP     0.027706          0.027233
10            Antidepressants     0.094340          0.093300
11             Antipsychotics     0.052264          0.052433
12  Miscellaneous_hallucinogens     0.001879          0.001667
13                  Inhalants     0.009668          0.009333
14                    lithium     0.007860          0.007700
15                    Opiates     0.174578          0.175867
16        Opiates_unspecified     0.032600          0.032133
17        Narcotic_analgesics     0.147420          0.148033
18              Buprenorphine     0.001014          0.000667
19                    Codeine     0.009444          0.009133
20                   Fentanyl     0.007097          0.006933
21                Hydrocodone     0.053049          0.053233
22                  Methadone     0.029202          0.028600
23                   Morphine     0.011054          0.010500
24                  Oxycodone     0.051556          0.051633
25                  Ibuprofen     0.027810          0.027367
26           Muscle_relaxants     0.032265          0.031933

    Algo_2_estimates  Algo_3_estimates
0           0.526615          0.563574
1           0.328980          0.388357
2           0.107706         -0.866334
3           0.216840          0.131663
```

```
4             0.101803         0.198052
5             0.028008        -0.269192
6             0.078446        -0.282818
7             0.007862        -0.046022
8            -0.003289        -0.279306
9             0.026631         0.558815
10            0.097296         0.620563
11            0.048618         0.420631
12            0.004011        -0.265626
13            0.011862        -0.322983
14            0.003598        -0.312335
15            0.176521         0.035884
16            0.032330        -0.349142
17            0.148424        -0.666163
18            0.000606        -0.285246
19            0.008109        -0.665541
20            0.006000         0.871207
21            0.053470        -0.362075
22            0.028180        -0.315096
23            0.012050         0.144929
24            0.051313         0.587558
25            0.025125        -0.653888
26            0.029777        -0.129760
```

[20]:
```
# Print the max error of Algorithm 3
print("Max error of Algorithm 3: {:.6f}".format(max_error(alg_3_estimates,␣
 ↪TRUE_MEAN)))
```

```
Max error of Algorithm 3: 0.864110
```

6.1.1 3.b. Question: Rank all three algorithms in order of how close the mean estimate was to the true mean. For the algorithm that had the worst estimate, why do you think it had the worst estimate?

Max error of Algorithm 1: 0.003123

Max error of Algorithm 2: 0.003236

Max error of Algorithm 3: 0.864110

In Algorithm 1, we do not add noise. In Algorithm 2, we added a single noise parameter to the non-private estimate ( ). On the other hand, in Algorithm 3, we are adding noise to each sensitive bit individually. Algorithm 3 had the worst estimate because it introduces a unique noise to each row before the sample mean is being applied. This takes it furthest away from the true estimate.

6.1.2 3.c. Question: Both Algorithm 2 and Algorithm 3 are ϵ-differentially private, but have different performances for mean estimation. Can you come up with a hypothetical practical scenario where you might want to use Algorithm 3 instead of Algorithm 2?

The main difference between Algorithm 3 and Algorithm 2 is that Algorithm 3 ensures privacy even if we don't trust the person or program calculating ( ) in Algorithm 2.

A potential hypothetical scenario where I might want to use Algorithm 3 instead of Algorithm 2 is if I'm dealing with sensitive patient data where I do not trust a doctor, nurse or researcher to be confidential with patient data (e.g. if they have not undergone HIPAA compliance training). In such a scenario, the people carrying out calculations on patient data cannot be trusted so we must introduce noise at each row to protect the data.

7 Question 4. Break the privacy of Algorithm 1

We said that Algorithm 1 is not private. In this question you will try to 'break' it in order to obtain private patient information. Denote by S the original dataset, and by S' a 'neighboring' dataset. The dataset S and S' differ only for one patients, whose data has been modified. Here by modifications we mean flipping some of the respective 0/1 from the row corresponding to that patient.

You are given only $N = 30000$, $f(S)$ and $f(S')$ (you don't have access to either S or S: Attempt to obtain private information about the patient whose data has been modified.

`Hint:` You should be able to infer that this patient was admitted to hospital having consumed alcohol but not cocaine and amphetamines. Explain.

[21]:
```
N = 30000

f_S = np.array([0.52756667, 0.32886667, 0.10943333, 0.22043333, 0.10026667,
                0.02733333, 0.07786667, 0.00756667, 0.0013    , 0.02723333,
                0.0933    , 0.05243333, 0.00166667, 0.00933333, 0.0077    ,
                0.17586667, 0.03213333, 0.14803333, 0.00066667, 0.00913333,
                0.00693333, 0.05323333, 0.0286    , 0.0105    , 0.05163333,
                0.02736667, 0.03193333])

f_S_prime = np.array([0.52753333, 0.3289    , 0.10943333, 0.22043333, 0.
 ↪10026667,
                      0.02736667, 0.07786667, 0.00756667, 0.0013    , 0.
 ↪02723333,
                      0.0933    , 0.05243333, 0.00166667, 0.00933333, 0.0077    ␣
 ↪,
                      0.17586667, 0.03213333, 0.14803333, 0.00066667, 0.
 ↪00913333,
                      0.00693333, 0.05323333, 0.0286    , 0.0105    , 0.
 ↪05163333,
                      0.02736667, 0.03193333])
```

[22]:
```
# TODO: Fill in with your answer
(f_S - f_S_prime)*N
```

[22]:
```
array([ 1.0002, -0.9999,  0.    ,  0.    ,  0.    , -1.0002,  0.    ,
        0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,
        0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,
        0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ])
```

The negative signs indicate that they were modified in the same fashion. The positive value indicates that it was modified in another fashion.

The first positive value indicates someone who was not accounted for had indeed consumed alcohol.

[23]:
```
%matplotlib inline
import matplotlib.image as mpimg
img = mpimg.imread('cute_alpaca.jpg')
imgplot = plt.imshow(img)
imgplot.axes.get_xaxis().set_visible(False)
imgplot.axes.get_yaxis().set_visible(False)
print("Yay, you've made it to the end of ALL THE LABS!")
plt.show()
```

```
Yay, you've made it to the end of ALL THE LABS!
```

[ ]:
```

```