CS394S Homework 1

Juexiao Zhou 175615

Date: Sep 17, 2021

Q1.

To prove $\sqrt{\mathbb{E}(Z^2)} = \sqrt{2}\lambda$

as we have $Z \sim Lap(\lambda), p(z) = \frac{1}{2\lambda}exp(-\frac{|z|}{\lambda})$

$\implies \mathbb{E}(Z) = 0 \; and \; \mathbb{V}\mathbb{a}\mathbb{r}(Z) = 2\lambda^2$

since $\mathbb{V}\mathbb{a}\mathbb{r}(Z) = \mathbb{E}(Z^2) - (\mathbb{E}(Z))^2$

$\implies \mathbb{E}(Z^2) = \mathbb{V}\mathbb{a}\mathbb{r}(Z) + (\mathbb{E}(Z))^2 = 2\lambda^2$

$\implies \sqrt{\mathbb{E}(Z^2)} = \sqrt{2}\lambda$

We can also prove this directly,

$$\mathbb{E}(Z^2) = \frac{1}{2\lambda}\int_{-\infty}^{\infty} z^2 exp(-\frac{|z|}{\lambda})dz$$

$$= \frac{1}{2\lambda}\int_{-\infty}^{0} z^2 exp(\frac{z}{\lambda})dz + \frac{1}{2\lambda}\int_{0}^{\infty} z^2 exp(-\frac{z}{\lambda})dz$$

$$= \frac{1}{\lambda}\int_{0}^{\infty} z^2 exp(-\frac{z}{\lambda})dz$$

$$= \lambda^2\int_{0}^{\infty} v^2 exp(-v)dv \quad (let \; v = \frac{z}{\lambda})$$

$$= 2\lambda^2$$

$\implies \sqrt{\mathbb{E}(Z^2)} = \sqrt{2}\lambda$

To prove for every $t > 0 : \mathbb{P}(z > \lambda t) \leq exp(-t)$

We simply integrate z on the PDF of the Laplace distribution

$$\implies \mathbb{P}(z > \lambda t) = \int_{\lambda t}^{\infty} \frac{1}{2\lambda}exp(-\frac{|z|}{\lambda})dz$$

$$= \int_{\lambda t}^{\infty} \frac{1}{2\lambda}exp(-\frac{z}{\lambda})dz$$

$$= \int_{t}^{\infty} \frac{1}{2}exp(-z)dz$$

$$= \frac{1}{2}exp(-t) \leq exp(-t)$$

Q2.

(a)

$\Delta^f = \max_{D,D'} ||f(D) - f(D')||_1 = \max_{D,D'} ||\frac{1}{n}x_j - \frac{1}{n}x'_j||_1 = \frac{1}{n}\max_{D,D'} ||x_j - x'_j||_1 \leq \frac{2}{n}$

where x_j and x'_j are the one differs in D and D'

(b)

let x_j and x'_j to be the one that differs in D and D'

$\Delta^f = \max_{D,D'} ||f(D) - f(D')||_1 = \max_{D,D'} ||\sum_{i=1}^n x_i x_i^T - \sum_{i=1}^n x'_i x_i'^T||_1 = \max_{D,D'} ||x_j x_j^T - x'_j x_j'^T||_1$

considering 2 different vectors x and x', we can write

$x = [w_1, w_2, w_3, \ldots, w_d]$

$x' = [w'_1, w'_2, w'_3, \ldots, w'_d]$

and we define the difference $\Delta_i = w'_i - w_i$, which means $\Delta_1 = w'_1 - w_1$ and so on

as we have $||x||_1 \leq 1,\ ||x'||_1 \leq 1$

so we have $\sum_{i=1}^d \Delta_i \leq 2$

for each corresponding position in the matrix xx^T and $x'x'^T$, let's say at row=1 and column=2, we have x_1x_2 and $x'_1x'_2$

the difference is $x'_1x'_2 - x_1x_2 = (x_1 + \Delta_1)(x_2 + \Delta_2) - x_1x_2 = x_1\Delta_2 + x_2\Delta_1 + \Delta_1\Delta_2$

then for each i and j, we have the corresponding difference $x'_ix'_j - x_ix_j = x_i\Delta_j + x_j\Delta_i + \Delta_i\Delta_j$

Then for all positions of the matrix xx^T and $x'x'^T$

the sum of all difference is $\sum_{j=1}^d \sum_{i=1}^d (x_i\Delta_j + x_j\Delta_i + \Delta_i\Delta_j) = \sum_{i=1}^d (x_i \sum_{j=1}^d \Delta_j + \Delta_i \sum_{j=1}^d x_j + \Delta_i \sum_{j=1}^d \Delta_j)$

since we have $\sum_{i=1}^d \Delta_i \leq 2$ and $||x||_1 \leq 1,\ ||x'||_1 \leq 1$

so the sum of all difference between the two matrix xx^T and $x'x'^T$

$$\sum_{j=1}^d \sum_{i=1}^d (x_i\Delta_j + x_j\Delta_i + \Delta_i\Delta_j) = \sum_{i=1}^d (x_i \sum_{j=1}^d \Delta_j + \Delta_i \sum_{j=1}^d x_j + \Delta_i \sum_{j=1}^d \Delta_j)$$
$$\leq \sum_{i=1}^d (2x_i + \Delta_i + 2\Delta_i) \leq 2 + 2 + 2 * 2 = 8$$

Then we have $\Delta^f = 8$

(c)

We first order all elements from 0 to 1, then

$D = [0,\ 0,\ \ldots,\ x_j,\ \ldots,\ 1,\ 1,\ 1]$

$D' = [0,\ 0,\ \ldots,\ x'_j,\ \ldots,\ 1,\ 1,\ 1]$

Where x_j and x'_j are the one that differs in D and D', and we let x_j and x'_j be the median.

There are only two cases

1. $x_j = 0$ and $x'_j = 1$
2. $x_j = 1$ and $x'_j = 0$

In both cases,

$\Delta^f = \max_{D,D'} ||f(D) - f(D')||_1 = \max_{D,D'} ||x_j - x'_j||_1 = 1$

(d)

From the setting, we know the difference between D and D' is the change of one pair of vertices (u, v)

Obviously, the number of total edges in the graph will affect the change of number of connected components if one edge is changed.

So the answer is ∞

Q3.

To prove this, we need to show that any vector $\tilde{s} \in \{0,1\}^d$ that disagrees with s on more than $\frac{r\alpha^2 n^2}{log(k/n)}$ entries, where r is a positive coefficient, cannot satisfy $|F_i\tilde{s} - a_i| \leq \alpha n,\ \forall i\ in[k]$

and thus can not be the output of th reconstruction attack.

So we fix the true secret vector $s \in \{0,1\}^n$ And let

$\mathbb{B} = \{\tilde{s} : \tilde{s}\ and\ s\ disagree\ on\ at\ least\ \frac{r\alpha^2 n^2}{log(k/n)}\ entries\}$

Our goal is to show the reconstruction attack will not output any vector in $\mathbb{B}$

We define that i eliminates $\tilde{s}$ if $|F_i(s - \tilde{s})| \geq 4\alpha n$

As $|F_i\tilde{s} - a_i| \geq |F_i(s - \tilde{s})| - |F_i s - a_i| \geq 3\alpha n$

So, if $\tilde{s}$ is eliminated by i, then $\tilde{s}$ can not be in $\mathbb{B}$

Now, our goal is to show that every vector in $\mathbb{B}$ is eliminated by some i, that is

$\forall \tilde{s}\ in \mathbb{B}, \exists i\ in[k], |F_i(s - \tilde{s})| \geq 4\alpha n$

We fix some $\tilde{s} \in \mathbb{B}$ and show that it is eliminated with a very high probability/

Suppose $\tilde{s} \in \{0,1\}^n$ differs from s on at least $m = \frac{r\alpha^2 n^2}{log(k/n)}$ coordinates, we will show that

$\exists i\ in[k], |F_i(s - \tilde{s})| \geq 4\alpha n$ has very high probability.

Now we use the Lemma 1,

$\mathbb{P}(|u \cdot t| \geq \frac{\sqrt{mlogw}}{10}) \geq \frac{1}{w}$

then we can modify it into

$\mathbb{P}(|u \cdot t| \leq \frac{\sqrt{mlogw}}{10}) \leq 1 - \frac{1}{w}$

to show $\exists i\ in[k], |F_i(s - \tilde{s})| \geq 4\alpha n$ has very high probability we can show $\forall i \in [k], |F_i \cdot (s - \tilde{s})| \leq 4\alpha n$ has very small probability

with $m = \frac{r\alpha^2 n^2}{log(k/n)}$, let $4\alpha n = \frac{\sqrt{mlogw}}{10}$

we have $w = exp(\frac{1600log(k/n)}{r})$

We can show $\mathbb{P}(|F_i \cdot (s - \tilde{s})| \leq 4\alpha n) \leq 1 - exp(-\frac{1600log(k/n)}{r})$

$\Longrightarrow \mathbb{P}(\forall i \in [k], |F_i \cdot (s - \tilde{s})| \leq 4\alpha n)) \leq (1 - exp(-\frac{1600log(k/n)}{r}))^k$

Thus

$\mathbb{P}(\exists \tilde{s} \in \mathbb{B}, \forall i \in [k], |F_i \cdot (s - \tilde{s})| \leq 4\alpha n)) \leq \sum_{\tilde{s} \in \mathbb{B}} \mathbb{P}(\forall i \in [k], |F_i \cdot (s - \tilde{s})| \leq 4\alpha n)) \leq 2^n \cdot (1 - exp(-\frac{1600log(k/n)}{r}))^k$

$\Longrightarrow \mathbb{P}(\exists \tilde{s} \in \mathbb{B}, \forall i \in [k], |F_i \cdot (s - \tilde{s})| \leq 4\alpha n) \leq 2^n(-\frac{k}{n} + 1 - exp(-\frac{1600}{r}))^k$

with $n^2 \ll k \ll 2^n$

as we can see, the probability above is very very small.

Prove finished.

as we know when we can ask $k = O(n)$ queries with error at most αn, then with extremely high probability, the reconstruction error is at most $O(\alpha^2 n^2)$

when $k \approx n^2$, and all queries have error at most αn, then with extremely high probability, the reconstruction error is at most $O(\alpha^2 n^2 / log(n))$

when $k \approx 2^{\sqrt{n}}$, and all queries have error at most αn, then with extremely high probability, the reconstruction error is at most $O(\alpha^2 n^2 / log(2^{\sqrt{n}}/n))$

when $k \approx 2^n$, and all queries have error at most αn, from the course we know the reconstruction error is at most $O(\alpha n)$

We observe that when the k increases, the reconstruction error decreases.

Q4.

Data Generation

For the generation of initial database, we used the function provided by numpy.random, we will randomly sample n points from values {0, 1} with equal probability (0.5 for 0 and 0.5 for 1).

```
# libraries

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Generate Dataset

def generate_database(n):
    return np.random.choice([0,1], replace = True, size=n, p=[0.5,0.5])
```

for the database, we also generate all neighbor databases with only one point different

```
# get all parallel databases
```

```
def generate_parallel_databases(database):
    output_all = []
    change = {1:0,0:1}
    for i in range(len(database)):
        tmp = database.copy()
        tmp[i] = change[tmp[i]]
        output_all.append(tmp)
    return output_all

def generate_database_and_parallel_databases(n):
    database =  generate_database(n)
    parallel_databases = generate_parallel_databases(database)
    return database, parallel_databases
```

Global sensitivity

given the function $f(D) = \frac{1}{n}\sum_{i=1}^{n} x_i$, we define it in python

```
# f(D)=mean(D)

def f(D):
    return np.mean(D)
```

Then we can calculate the global sensivity by iterating all neighbour databases.

```
# calculate global sensitivity

def GS(database, parallel_databases, func):
    _result = func(database)
    _max_diff = 0
    for pd in parallel_databases:
        __result = np.abs(func(pd) - _result)
        if __result > _max_diff:
            _max_diff = __result
    return _max_diff
```

Randomized response

Before calculating the mean, for each point of in the database, we flip a coin twice.

If the first coin flip is heads, answer honestly

If the first coin flip is tails, answer according to the second coin flip (heads for 1, tails for 0)

Then, based on the noised database, we output the value from the function f

```
# randomized response mechanism

def randomized_response(func, database):
    _noised_database = []
    change = {1:0,0:1}
    for d in database:
        _flip1 = np.random.rand()
        _flip2 = np.random.rand()
        if _flip1 >= 0.5:
```

```
            _noised_database.append(d)
        elif _flip1 < 0.5:
            if _flip2 >= 0.5:
                _noised_database.append(1)
            else:
                _noised_database.append(0)
    return func(_noised_database)
```

Laplacian Mechanism

for each point in the database, we add the laplacian noise to those values and output the result of f

```
# laplacian mechanism

def laplacian(func, sensitivity, e, database):
    _result = func(database)
    _noise = np.random.laplace(0, sensitivity/e, 1)
    return _result + _noise[0]
```

Experiments

We tested different size of the database with n from [100,200, 300, 500, 1000,2000, 3000, 5000, 10000]

and different ϵ from [0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1, 2, 3, 5]

for each pair of experiment, we repeat 100 times to calculate the average of final estimation error.

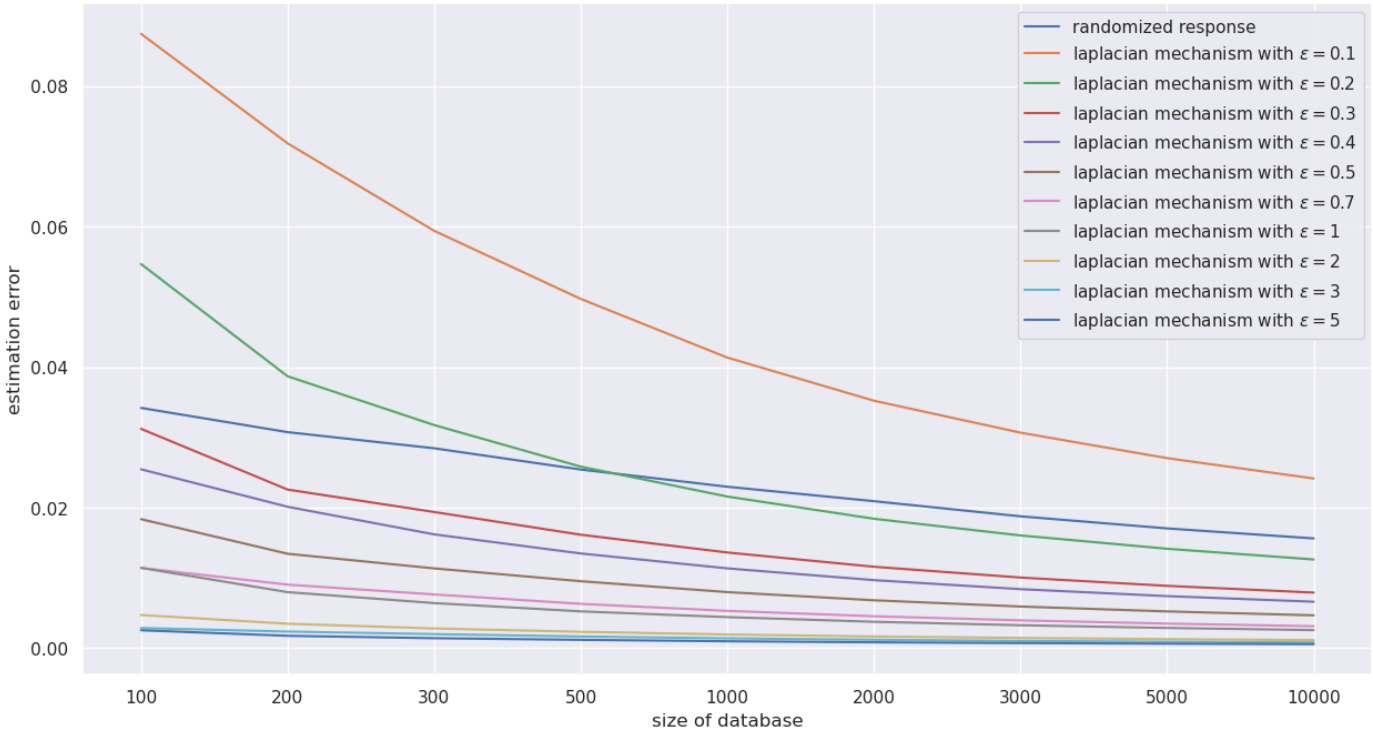
The utility (estimation error) of each experiment is defined as $estimation\ error = \ noised\ output - true\ answer$

If the estimatio error is large, then the utility is small. if the estimation error is small, then the utility is high.

The complete code:

```
ns = [100,200, 300, 500, 1000,2000, 3000, 5000, 10000]
es = [0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1, 2, 3, 5]
repeats = 100

true_errors = []
randomized_errors = []
tmp_randomized_errors = []
for i in range(len(es)):
    exec('laplacian_errors_e{}=[]'.format(str(i)))
    exec('tmp_laplacian_errors_e{}=[]'.format(str(i)))

for n in ns:

    print(n)

    for repeat in range(repeats):

        database, parallel_databases = generate_database_and_parallel_databases(n)
        #sensitivity = GS(database, parallel_databases, func=f)
        sensitivity = 1/n
```

```
        # true result
        true_result = f(database)

        # randomized result
        _result = f_randomized_response(f, database)
        tmp_randomized_errors.append(np.abs(_result-true_result))

        # laplacian result
        for i in range(len(es)):
            _result = f_laplacian(f, sensitivity, es[i], database)
            #laplacian_errors.append(np.abs(_result-true_result))
            exec('tmp_laplacian_errors_e{}.append(np.abs(_result-true_result))'.format(str(i)))

    for i in range(len(es)):
        exec('laplacian_errors_e{}.append(np.mean(tmp_laplacian_errors_e{}))'.format(str(i),str(i)))
    randomized_errors.append(np.mean(tmp_randomized_errors))

sns.set()
plt.figure(figsize=[15,8],dpi=100)
sns.lineplot(range(len(randomized_errors)),randomized_errors, label='randomized response')
sns.lineplot(range(len(laplacian_errors)),laplacian_errors, label='laplacian mechanism')
for i in range(len(es)):

 sns.lineplot(range(len(eval('laplacian_errors_e{}'.format(str(i))))),eval('laplacian_errors_e{}'.format(str(i
))), label='laplacian mechanism with $\epsilon={}$'.format(es[i]))
plt.xticks(range(len(ns)),ns)
plt.xlabel('size of database')
plt.ylabel('estimation error')
```



We find:

1. When the sample size increase, the estimation error will decrease (utility increases) for both randomized response

and laplacian mechanism. (we can use the law of large number to explain for that)
2. The randomized response gives better utility (lower estimation error) compared to laplacian mechanism, when $\epsilon = 0.1$.
3. When the size of the database < 500, the randomized response has better utility than laplacian mechanism with $\epsilon = 0.2$, while when the size of the database > 500, the randomized response has a worse utility than laplacian mechanism with $\epsilon = 0.2$.
4. When $\epsilon \geq 0.3$, the utility of the laplacian mechanism is always better than the randomized response.
5. When the ϵ is very large, the estimation error with laplacian mechanism coverges to 0.
6. Smaller ϵ -> larger λ in $Lap(\lambda)$ -> larger noise added -> better privacy -> less utility -> higher estimation error.