2024 ATML assignment 1

Catalog

(a)

Answer:

The function $f(x) = \langle x, q\rangle = \sum_{i=1}^{n} x_i q_i$ maps the dataset x to a real number.

The sensitivity Δf is defined as the maximum change in the function's output between any two neighboring datasets x and x' (which differ in exactly one coordinate):

$$\Delta f = \max_{x,x'} |f(x) - f(x')|$$

Suppose x and x' differ at index i, the difference in the function's output is:

$$|f(x) - f(x')| = |(x_i - x_i{}')q_i|$$

Since $x_i,\ x_i{}',\ q_i \in \{-1, 1\}$, thus, the sensitivity is:

$$\Delta f = \max_{x,x'} |f(x) - f(x')| = 2$$

The Laplace mechanism adds noise drawn from the Laplace distribution $z \sim Lap(\lambda)$ to achieve differential privacy. The scale parameter λ is related to the desired privacy level ε and the sensitivity Δf by:

$$\lambda = \frac{\Delta f}{\varepsilon}$$

Rewriting this equation to solve for ε:

$$\varepsilon = \frac{\Delta f}{\lambda}$$

Using the computed sensitivity $\Delta f = 2$:

$$\varepsilon = \frac{2}{\lambda}$$

Each single query is ε-differentially private with $\varepsilon = 2/\lambda$ because the Laplace noise added accounts for the sensitivity of the function, ensuring that the probability distributions of the outputs under neighboring datasets are within a multiplicative factor e^{ε} of each other.

Because the function's sensitivity is 2, adding $Lap(\lambda)$ noise with $\lambda = 2/\varepsilon$ makes each query ε-differentially private with $\varepsilon = 2/\lambda$.

(b)

Answer:

The screenshot from the output of my code:

0.86	test01Byqvl920	200

linear program statement:

Set variables up $x_i \in [-1,1]$ for $i = 1,2,\ldots,n$

Minimize the total absolute error between the observed noisy query results y and the estimated query results Q_x: Minimize $\sum_{k=1}^{t} s_k$.

Constraints: For each query $k = 1,2,\ldots,t$: $(Q_x)_k - y_k \leq s_k$; $-(Q_x)_k - y_k \leq s_k$; $s_k \geq 0$ Variable bounds: $-1 \leq x_i \leq 1$ for all i

Explanation of the Code (Please refer to Appendix: Linear Program Code for the code comments.)

Generating Queries: generate num_queries random queries Q where each entry is either -1 or $+1$.

Collecting Responses: send these queries to the remote database to get the noisy responses y.

Setting Up the Linear Program:

Variables: We have n variables for x and t slack variables s.

Objective Function: minimize the sum of the slack variables s_k, which represents the total absolute error.

Constraints: For each query, set up two inequalities to handle the absolute value in the ℓ_1 norm. ensure $x_i \in [-1,1]$ and $s_k \geq 0$.

Solving the Linear Program:

As suggested in the code file use scipy.optimize.linprog with the 'highs' method for efficiency.

Reconstructing x: extract the solution for xxx and threshold it to obtain values in $\{-1,+1\}$.

Submitting the Reconstruction: submit the reconstructed xxx to the server to get the fraction of correct entries.

(c)

Answer:

Yes, the mechanism $\mathcal{M}$ satisfies $(ln9)$-differential privacy. Here's why:

Randomized Response for Bits

The randomized response mechanism for a single bit works as follows:

Given a bit $b \in \{0,1\}$:

With probability p, output the true bit b.

With probability $q = 1 - p$, output the flipped bit $1 - b$.

Differential Privacy of Randomized Response

For a single bit, the randomized response mechanism satisfies ε-differential privacy, where:

$$\varepsilon = \ln(\frac{p}{q})$$

This is because the maximum privacy loss when flipping a bit is:

$$\max_{b,b',o} \ln\left(\frac{Pr[\mathcal{M}(b) \ = \ o]}{Pr[\mathcal{M}(b') \ = \ o]}\right) = \ln(\frac{p}{q})$$

Applying Randomized Response to Two Bits Independently

Since $\mathcal{M}$ applies randomized response independently to each bit, the overall privacy guarantee combines the privacy loss from both bits.

Per Bit Privacy Loss: $\varepsilon' = \ln(\frac{p}{q})$

Total Privacy Loss: $\varepsilon = 2\varepsilon'$.

We want to find p such that the total privacy loss $\varepsilon = ln9$. Therefore:

$$\varepsilon = 2\varepsilon' = ln9 \implies \varepsilon' = ln3$$

Then, solve for p:

$$\varepsilon' = \ln(\frac{p}{q}) \implies \ln\left(\frac{p}{1-p}\right) = ln3$$

Simplify: $\frac{p}{1-p} = 3 \implies p = 3(1-p) \implies p = \frac{3}{4}$

Thus: $p = \frac{3}{4}$, $q = 1—p = \frac{1}{4}$

Encoding and Mechanism:

Each character $x \in \chi = \{A, C, G, T\}$ is encoded as a 2-bit string: $\{A, C, G, T\} \equiv \{00,01,10,11\}$

The mechanism $\mathcal{M}$ applies the randomized response independently to each bit of x.

Randomized Response on a Bit:

For each bit $b \in \{0,1\}$:

Output the true bit b with probability $p = {}^{3}/_{4}$.

Output the flipped bit $1-b$ with probability $q = 1-p = {}^{1}/_{4}$.

The privacy loss for a single bit is:

$$\varepsilon' = \ln(\frac{p}{q}) = \ln\left(\frac{3/4}{1/4}\right) = ln3$$

Differential Privacy of the Mechanism:

Since the bits are independent, the total privacy loss is the sum over both bits:

$\varepsilon = \varepsilon' + \varepsilon' = 2 \times ln3 = ln9$

Therefore, $\mathcal{M}$ satisfies $ln9$-differential privacy.

By applying the randomized response with $p = \frac{3}{4}$ independently to each bit, the mechanism $\mathcal{M}$ ensures that for any two possible inputs and any output, the ratio of probabilities is bounded by $e^{ln9} = 9$. Hence, $\mathcal{M}$ satisfies $ln9$-differential privacy.

(d)

Answer:

For each bit of the 2-bit encoding: Instead of using $p = \frac{3}{4}$ and $q = \frac{1}{4}$, increase the probability of outputting the correct bit (while maintaining the privacy loss constraint). Denote the new probability as p' and the probability of flipping the bit as $q' = 1 - p'$.

Adjusting $\boldsymbol{p'}$ and $\boldsymbol{q'}$:

In $\mathcal{M}$, the privacy loss per bit is:

$$ln\left(\frac{p}{q}\right) = ln3$$

To ensure that $\mathcal{M}'$ also satisfies $ln9$-differential privacy, the new mechanism $\mathcal{M}'$ should satisfy: $ln\left(\frac{p'}{q'}\right) \leq ln3$

Solving for p' in terms of $q' = 1 - p'$, get:

$$\frac{p'}{q'} = 3$$
$$\Rightarrow p' = 3q'$$

To increase the probability of outputting the correct bit while maintaining the privacy constraint, let's choose a slightly larger p' that still satisfies $ln3$-differential privacy: $p' = \frac{8}{9}$ and $q' = \frac{1}{9}$.

Constructing the Mechanism $\mathcal{M}'$:

For each bit of the 2-bit encoding:

With probability $p' = \frac{8}{9}$, output the true bit.

With probability $q' = \frac{1}{9}$, output the flipped bit.

Since applying the mechanism to both bits independently, the overall probability of correctly outputting xxx (the true encoded value) is:

$$Pr[\mathcal{M}'(x) = x] = \frac{8}{9} \times \frac{8}{9} = \frac{64}{81}$$

This is higher than the probability for the original mechanism $\mathcal{M}$, which outputs the correct value with probability:

$$Pr[\mathcal{M}(x) = x] = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$$

Differential Privacy Check:

Verify that $\mathcal{M}'$ satisfies $ln9$-differential privacy.

The privacy loss per bit is: $\varepsilon' = \ln(\frac{p'}{q'}) = \ln\left(\frac{8/9}{1/9}\right) = ln8$

For two bits, the total privacy loss is: $\varepsilon = 2ln8 = ln64$

Since $ln64 < ln81$, the mechanism $\mathcal{M}'$ satisfies $ln9$-differential privacy.

Appendix: Linear Program Code

```
from scipy.optimize import linprog
# Set up the linear program
t = num_queries
Q = queries  # Query matrix of shape (t, n)
y = query_results  # Observed noisy responses of shape (t,)

# Variables: x (n variables) and s (t slack variables)
# Total variables: n + t

# Objective function coefficients
c = np.concatenate([np.zeros(n), np.ones(t)])

# Inequality constraints: A_ub * [x; s] <= b_ub
# For each k:
```

```python
# (Qx)_k - y_k <= s_k
# -(Qx)_k + y_k <= s_k
# s_k >= 0

A_ub = np.zeros((2*t, n + t))
b_ub = np.zeros(2*t)

# First t inequalities: (Qx)_k - y_k - s_k <= 0
A_ub[:t, :n] = Q
A_ub[:t, n:] = -np.eye(t)
b_ub[:t] = y

# Next t inequalities: -(Qx)_k + y_k - s_k <= 0
A_ub[t:, :n] = -Q
A_ub[t:, n:] = -np.eye(t)
b_ub[t:] = -y

# Bounds for x_i: -1 <= x_i <= 1
x_bounds = [(-1, 1) for _ in range(n)]
# Bounds for s_k: s_k >= 0
s_bounds = [(0, None) for _ in range(t)]
bounds = x_bounds + s_bounds

# Solve the linear program
res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

# Extract the solution for x
x_hat = res.x[:n]

# Convert x_hat to {-1, +1}
x_hat_sign = np.sign(x_hat)
x_hat_sign[x_hat_sign == 0] = 1  # In case of zero, assign +1

# Submit the reconstructed x_hat_sign
reconstruction_result = query(challenge_id, x_hat_sign.astype(int),
submit=True)

# Print the fraction of correct entries
fraction_correct = (1 + reconstruction_result / n) / 2
print(f"\nReconstruction attack achieves fraction
{fraction_correct:.2%} correct values")
```