INT 6151: Statistical Machine Learning	**Second Semester 2023-2024**
Homework 1	
Student Name: Le The Nam	*UID: 23025079*

1 Gradient Descent - 10pts

1.1

The first derivative of f(x) is $\partial_x f = 2x - 1$

With $\lambda = 0.05$, apply Gradient Descent:

- Initialization step $k = 0$, we have $x^{(0)} = 1$ and $f(x^{(0)}) = 6$.
- After updating at step $k = 0$, $\partial_x f(x^{(0)}) = 1.0$, then $x^{(1)} = 0.95$ and $f(x^{(1)}) = 5.9525$
- After updating at step $k = 1$, $\partial_x f(x^{(1)}) = 0.9$, then $x^{(2)} = 0.905$ and $f(x^{(2)}) = 5.914025$
- After updating at step $k = 2$, $\partial_x f(x^{(2)}) = 0.81$, then $x^{(3)} = 0.8645$ and $f(x^{(3)}) = 5.88286025$
- After updating at step $k = 3$, $\partial_x f(x^{(3)}) = 0.729$, then $x^{(4)} = 0.82805$ and $f(x^{(4)}) = 5.8576168025$

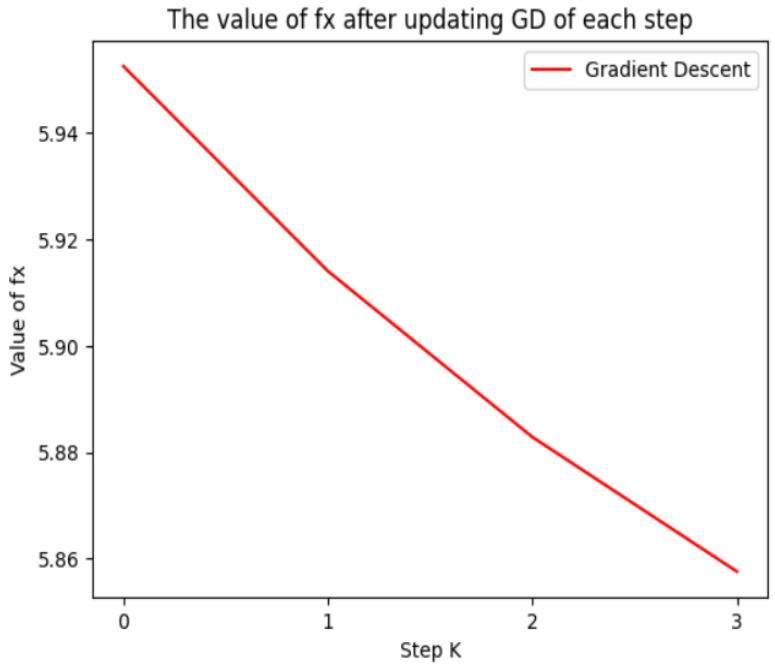

Figure 1: Gradient descent with $\lambda = 0.05$

1.2

The first derivative of f(x):

- $\partial_{x_1} f = 2(x_1 - {x_2}^2 - 3) + 2(x_1 - x_2 - 1)$
- $\partial_{x_2} f = -4x_2(x_1 - {x_2}^2 - 3) - 2(x_1 - x_2 - 1)$

With $\lambda = 0.2$, apply Gradient Descent:

- Initialization step $k = 0$, we have $x^{(0)} = [-0.5, -0.5]$ and $f(x^{(0)}) = 15.0625$.
- After updating at step $k = 0$, $\partial_x f(x^{(0)}) = [-9.5, -5.5]$, then $x^{(1)} = [1.4, 0.6]$ and $f(x^{(1)}) = 3.8816$
- After updating at step $k = 1$, $\partial_x f(x^{(1)}) = [-4.32, 5.104]$, then $x^{(2)} = [2.264, -0.4208]$ and $f(x^{(2)}) = 3.6722526859165687$
- After updating at step $k = 2$, $\partial_x f(x^{(2)}) = [1.54345472, -4.906483867648]$, then $x^{(3)} = [1.955309056, 0.5604967735296]$ and $f(x^{(3)}) = 2.002343476280831$

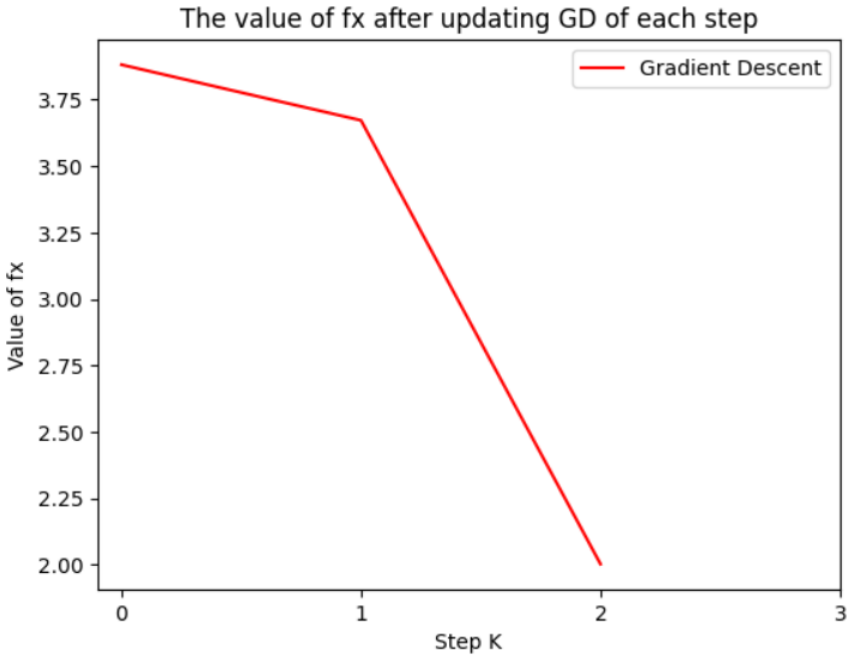

Figure 2: Gradient descent with $\lambda = 0.2$

To determine whether the function has the local minimum or not. We need to solve the following system of equations:

$$\frac{\partial f}{\partial x_1} = 0 \tag{1}$$

$$\frac{\partial f}{\partial x_2} = 0 \tag{2}$$

The solution is $x_s = (\frac{19}{8}, \frac{1}{2})$

Then, we need to calculate the Hessian matrix of $f(x)$, with

$$\begin{aligned} A &= \partial_{x1x1} f(x_s) \\ B &= \partial_{x1x2} f(x_s) \\ C &= \partial_{x2x2} f(x_s) \end{aligned} \tag{3}$$

Because $AC - B^2 = 44 > 0$ and $A > 0$, f(x) has one local minimum at $x_s = (\frac{19}{8}, \frac{1}{2})$

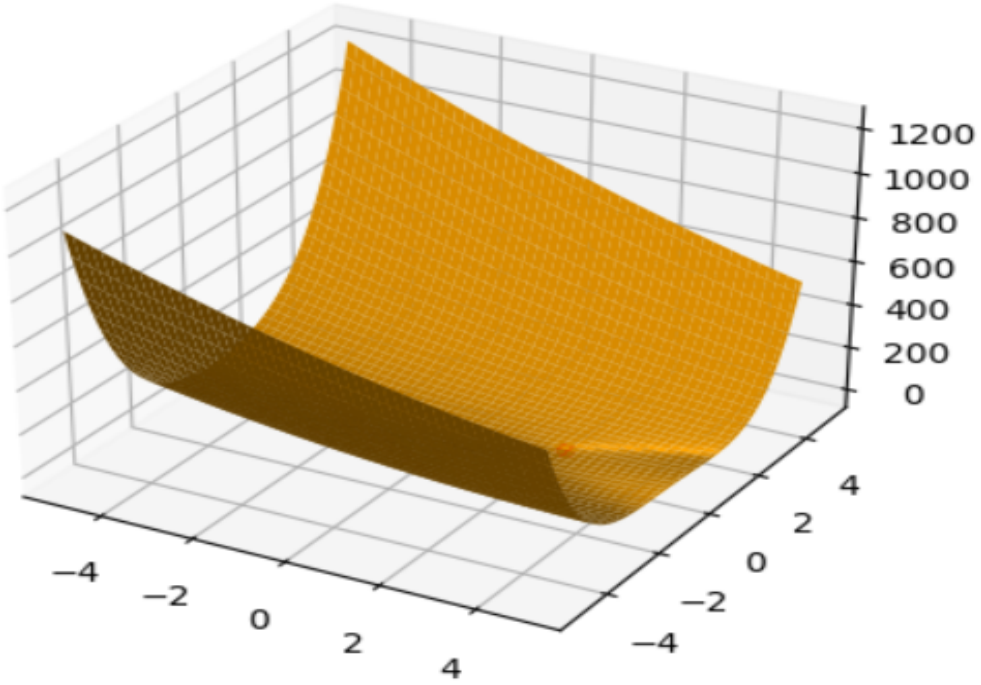

Figure 3: The plot of the function $f(x)$

1.3

Apply the gradient descent with $\lambda = 0.01$ and $\lambda = 0.05$ respectively, and draw the contour plot of $f(x)$

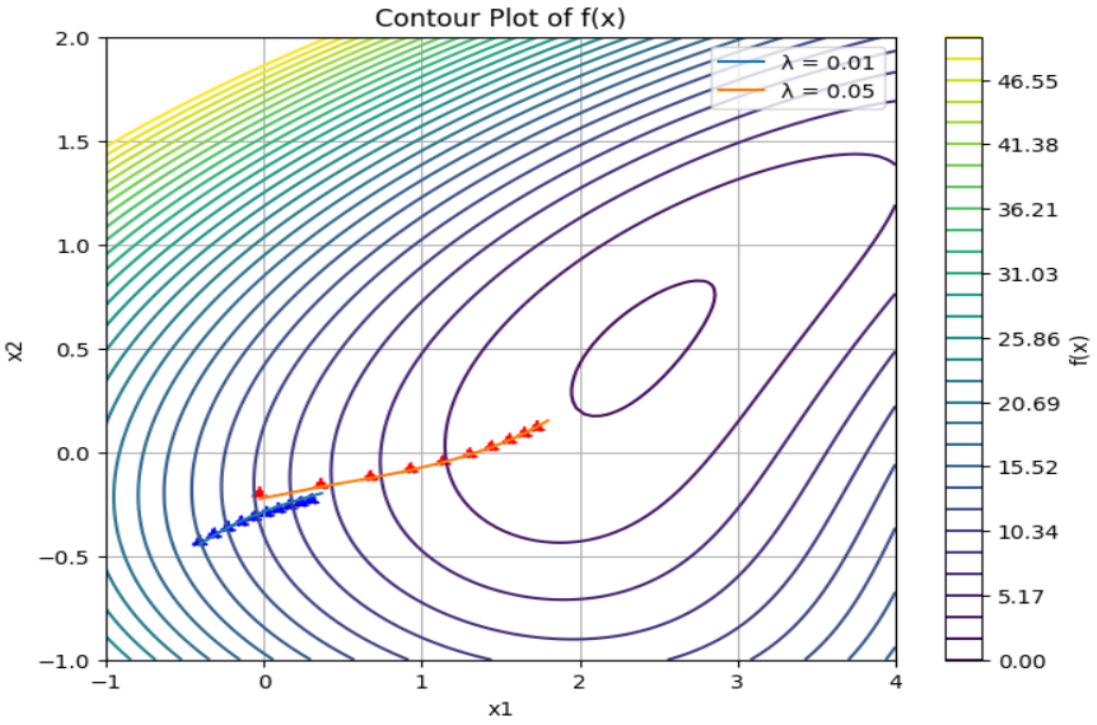

Figure 4: Gradient descent with $\lambda = 0.01$ and $\lambda = 0.05$

After 10 steps, the values of the function $f(x)$ show that using $\lambda = 0.05$ helps the value of the function converge faster through each step than using $\lambda = 0.01$. Therefore, when running gradient descent for $f(x)$, the $\lambda = 0.05$ yields better optimization results

2 Coding - 10pts

2.1

We have a logistic regression problem, so the concentration bound is:

$$P\left\{|\widehat{err}_D(h) - err_P(h)| \leq \epsilon\right\} \geq 1 - 2e^{-2n\epsilon^2} \tag{4}$$

With $\widehat{err}_D h = 0$, apply (4), we have:

$$\begin{aligned} & P\left\{|-err_P(h)| \leq \epsilon\right\} \geq 1 - 2e^{-2n\epsilon^2} \\ \Leftrightarrow\ & P\left\{|-err_P(h)| \leq \epsilon\right\} \geq 1 - 2e^{-2n\epsilon^2} \\ \Leftrightarrow\ & P\left\{err_P(h) \leq \epsilon\right\} \geq 1 - 2e^{-2n\epsilon^2} \\ \Leftrightarrow\ & P\left\{err_P(h) \leq \epsilon\right\} \geq 1 - 2e^{-2n\epsilon^2} \\ \Leftrightarrow\ & P\left\{err_P(h) \leq 0.1\right\} \geq 1 - 2 * e^{-2*10*0.1^2} = -0.637(AlwaysTrue) \end{aligned} \tag{5}$$

So, it can be said that when $\widehat{err}_D(h) = 0$, $P\left\{err_P(h) > 0.1\right\} = 0$ or it doesn't occur that $err_P(h) > 0.1$

2.2

Training code to achieve $\widehat{err}_D(h) = 0$

```
import numpy as np

def add_noise_data(input_data, input_labels, n_points, mean, scale):
    """
    Create a noise version of the input data

    Params:
        input_data: base input data
        input_labels: base input labels
        n_points: the number of needed points
        mean, scale: the Gaussian data
    """
    raw_X = []
    raw_labels = []

    noise = np.random.normal(loc=mean, scale=scale, size=(n_points, 2))
    for i in range(n_points):
        k = np.random.randint(len(input_data))

        raw_X.append([input_data[k][0] + noise[i][0],
                      input_data[k][1] + noise[i][1]])
```

```
        raw_labels.append(input_labels[k])

    return np.array(raw_X), np.array(raw_labels)

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def bce_loss(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def gradient(X, y_true, y_pred):
    return X.T @ (y_pred - y_true)

def err_metric(y_true, y_pred):
    n = y_true.shape[0]
    y_pred[np.where(y_pred >= 0.5)] = 1
    y_pred[np.where(y_pred < 0.5)] = 0
    return 1/n*sum(abs(y_true - y_pred))

def train_logistic_regression(X, y, learning_rate, epochs):
    # Initialize the weights
    w = np.zeros(X.shape[1])
    b = 0

    training_loss = []
    for epoch in range(epochs):
        y_pred = sigmoid(X @ w + b)
        current_loss = bce_loss(y, y_pred)
        grad_w = gradient(X, y, y_pred)
        grad_b = np.sum(y_pred - y)
        w -= learning_rate * grad_w
        b -= learning_rate * grad_b
        err = err_metric(y, y_pred)
        # print(f"Epoch {epoch + 1}: Loss = {current_loss}, Weights = {w}, Bias = {b}, E
        if err == 0:
            print(f"Epoch {epoch + 1}: Loss = {current_loss}, Weights = {w}, Bias = {b},
            print("Traning done!")
            break
        else:
            if current_loss <= 1e-5:
                print(f"Traning failed with {epochs}!")
        training_loss.append(current_loss)
    if (epoch + 1) == epochs:
        print(f"Traning failed with {epochs}, err = {err}!")
    return w, b, training_loss
```

```
# Training phase
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
# sample dataset: D = 10
SEEDS = [0, 42, 100, 1000, 2024]
for seed in SEEDS:
    print(f"seed = {seed}")
    np.random.seed(seed)
    X_noise, y_noise = add_noise_data(X, y, 10, 0., 0.2)
    # print(X_noise)
    learning_rate = 0.1
    epochs = 10000
    w, b, training_loss = train_logistic_regression(X_noise, y_noise, learning_rate, epo
```

Experiment with some different seeds, I have the following results

SEED	NumEpochs	ERR
0	4	0
42	199	0
100	46	0
1000	24	0
2024	118	0

Table 1: The results for training $D = 10$, with $\lambda = 0.1$

2.3

When the sample size increases, the time for the logistic regression algorithm to classify may increase, and even it may not be separable when the noise is large.

```
# Training phase
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

for N in [20, 50, 100, 500]:
    SEEDS = [0, 42, 100, 1000, 2024]
    for seed in SEEDS:
        print(f"N = {N}, seed = {seed}")
        np.random.seed(seed)
        X_noise, y_noise = add_noise_data(X, y, N, 0., 0.2)
        # print(X_noise)
        learning_rate = 0.1
        epochs = 10000
        w, b, training_loss = train_logistic_regression(X_noise, y_noise, learning_rate,
    print("--------------")
```

Experiment with some different seeds, I have the following results

NumSamples (D)	SEED	NumEpochs (MAX: 10000)	ERR
20	0	6	0
	42	41	0
	100	49	0
	1000	18	0
	2024	Failed	0.05
50	0	2385	0
	42	3	0
	100	Failed	0.02
	1000	Failed	0.1
	2024	Failed	0.04
100	0	Failed	0.04
	42	Failed	0.01
	100	Failed	0.02
	1000	Failed	0.03
	2024	Failed	0.03
500	0	Failed	0.022
	42	Failed	0.02
	100	Failed	0.016
	1000	Failed	0.024
	2024	Failed	0.026

Table 2: The results for training different sizes of D, with $\lambda = 0.1$

2.4

If the testing dataset is drawn from $\mathcal{N}(0, 0.3)$, while the training dataset is drawn from $\mathcal{N}(0, 0.2)$, the performance of the logistic regression model will be decreased. This action is nearly meaningless in practice.

```
# Training phase
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
# sample dataset: D
SEEDS = [0, 42, 100, 1000, 2024]
for N in [10, 20, 50]:
    for seed in SEEDS:
        print(f"N = {N}, seed = {seed}")
        np.random.seed(seed)
        X_noise, y_noise = add_noise_data(X, y, N, 0., 0.2)
        X_test, y_test = add_noise_data(X, y, N, 0., 0.3)

        # training
        learning_rate = 0.1
        epochs = 10000
        w_train, b_train, training_loss = train_logistic_regression(X_noise, y_noise, le
```

```
        # tesing
        y_pred = sigmoid(X_test @ w_train + b_train)
        print(f"Testing ERR: {err_metric(y_test, y_pred)}")
    print("---------")
```

Experiment with some different seeds and the testing dataset has the same size with D. I have the following results

NumSamples (D)	SEED	NumEpochs (MAX: 10000)	Trainnig ERR	Testing ERR
10	0	4	0	0
	42	199	0	0
	100	46	0	0
	1000	24	0	0
	2024	118	0	0.1
20	0	6	0	0.1
	42	41	0	0.15
	100	49	0	0.15
	1000	18	0	0.05
	2024	Failed	0.05	0.05
50	0	2385	0	0.04
	42	3	0	0.18
	100	Failed	0.02	0.1
	1000	Failed	0.1	0.04
	2024	Failed	0.04	0.04

Table 3: The results for testing dataset drawn from different distribution of D, with $\lambda = 0.1$