Machine Learning Homework 1

Question 1)

a)

Given the table, we are asked to evaluate $P(Z|\ X\ =\ 0, Y\ =\ 0)$. If we look at the conditions where X and Y are 0, we see that Z can take 1 or 2 values with

$$P(Z=1|\ X\ =\ 0, Y\ =\ 0) = \frac{0.06}{0.06+0.09} = 0.4$$

$$P(Z=2|\ X\ =\ 0, Y\ =\ 0) = \frac{0.06}{0.06+0.09} = 0.6$$

Which provides the probability mass function of Z

b)

To check whether Y and Z are independent, we firstly integrate over X to get $P(Y, Z)$

Z\Y	-1	0	1
1	0.12	0.08	0.1
2	0.33	0.17	0.2

Also integrate over Z on $P(Y, Z)$ to get $P(Y)$

Y	-1	0	1
	0.45	0.25	0.3

Lastly, integrate over Y on $P(Y, Z)$ to get $P(Z)$

Z	1	2
	0.3	0.7

For independence we need $P(Y, Z) = P(Y)P(Z)$ to hold.

$P(Y)P(Z)$ can be calculated as product of individual elements which is

Z\Y	-1	0	1
1	0.135	0.075	0.09
2	0.315	0.175	0.21

Which is not the same as $P(Y, Z)$, thus, we can conclude these normal variables are not independent

Question 2)

a)

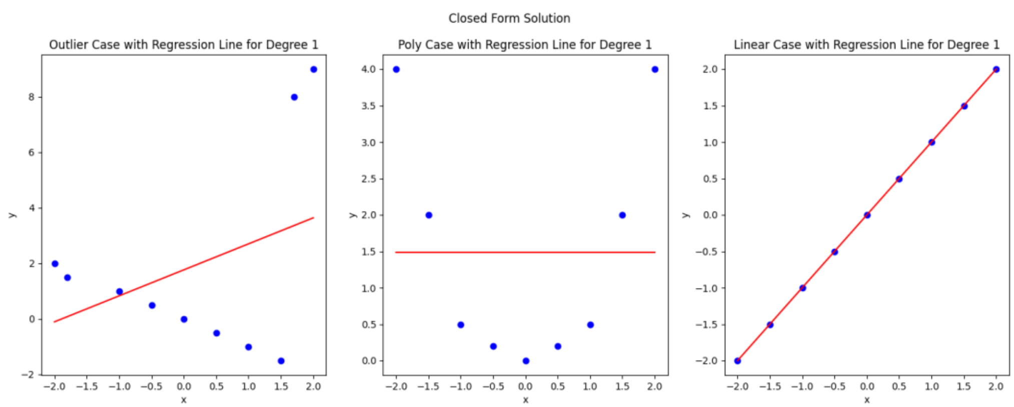

Figure 1: Closed form solution for linear regression

The expected absolute error for the Outlier data using closed_form and degree 1 is 2.9062

The expected absolute error for the Poly data using closed_form and degree 1 is 1.3432

The expected absolute error for the Linear data using closed_form and degree 1 is 1.4815

b)

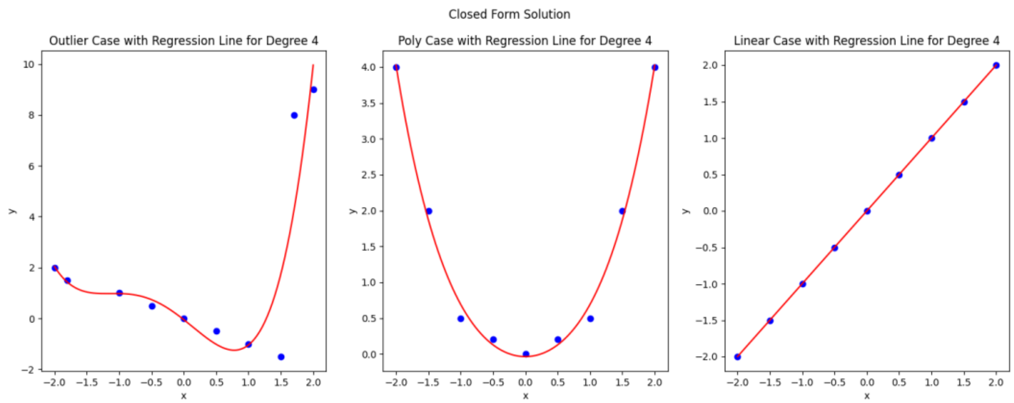

Figure 2: Closed form solution for fourth order polynomial

The expected absolute error for the Outlier data using closed_form and degree 4 is 3.3262

The expected absolute error for the Poly data using closed_form and degree 4 is 1.6297

The expected absolute error for the Linear data using closed_form and degree 4 is 1.4815

c)

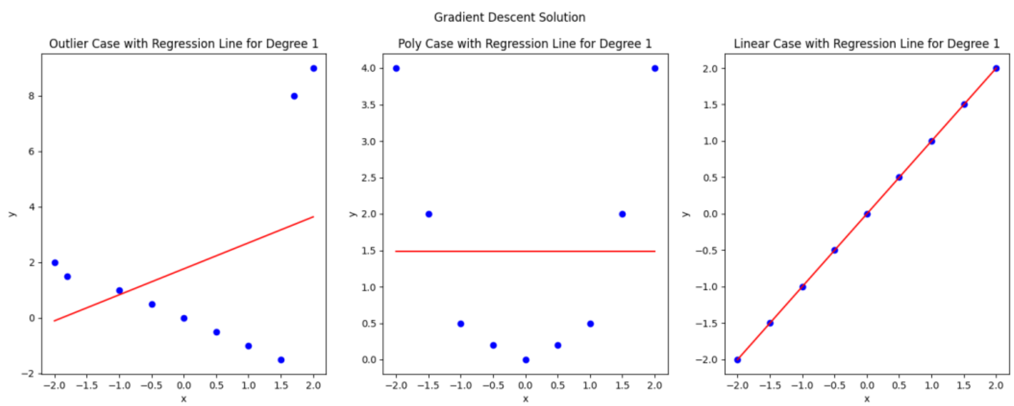

Figure 3: Linear regression with gradient descent

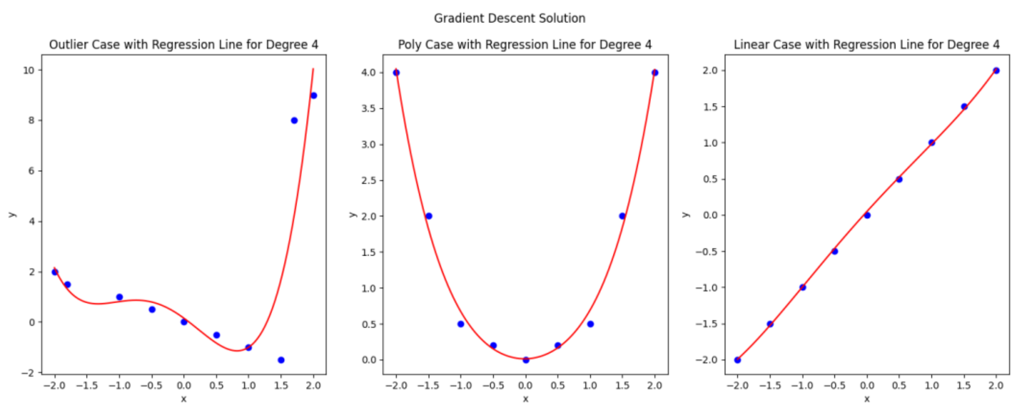

Figure 4: Fourth order regression with gradient descent

The expected absolute error for the Outlier data using gradient_descent is 0.8991

The expected absolute error for the Poly data using gradient_descent is 0.0964

The expected absolute error for the Linear data using gradient_descent is 0.0511

Question 3)

$$H = healthy, S = sick, N = negative, P = positive$$

$$P(S) = 0.02, P(H) = 0.98$$

$$P(P|S) = 0.9$$

$$P(N|H) = 0.9, P(P|H) = 0.1$$

To find probability of being sick given the test is positive we expand using bayes formula

$$P(S|P) = \frac{P(P|S)\ x\ P(S)}{P(P)} = \frac{P(P|S)\ x\ P(S)}{P(P|S)\ x\ P(S) + P(P|H)\ x\ P(H)} = \frac{0.9\ x\ 0.02}{0.9\ x\ 0.02 + 0.1\ x\ 0.98} = 0.1552$$

Therefore, the test is approximately 155 times accurate out of 1000 times. I would not trust the test out of a single positive and try to redo it until we approach an acceptable probability.

Question 4)

Given the probabilities and utility matrix, expected utility for each class c_{pred} can be calculated with

$$E[U(c_{pred}|x)] = \sum_{c_{true}=1}^{\#classes=3} p(c_{true}|x)\ x\ U(c_{true}, c_{pred})$$

$$E[U(1|x)] = p(c=1|x)\ x\ U(1,1) + p(c=2|x)\ x\ U(2,1) + p(c=3|x)\ x\ U(3,1)$$
$$= 0.7\ x\ 5 + 0.2\ x\ 0 + 0.1\ x(-3) = 3.2$$

$$E[U(2|x)] = p(c=2|x)\ x\ U(1,2) + p(c=2|x)\ x\ U(2,2) + p(c=3|x)\ x\ U(3,2)$$
$$= 0.7\ x\ 3 + 0.2\ x\ 4 + 0.1\ x\ 0 = 2.9$$

$$E[U(3|x)] = p(c=1|x)\ x\ U(1,3) + p(c=2|x)\ x\ U(2,3) + p(c=3|x)\ x\ U(3,3)$$
$$= 0.7\ x\ 1 + 0.2\ x\ (-2) + 0.1\ x\ 10 = 1.3$$

Based on this, selecting $c_{pred} = 1$ makes the most sense.

Question 5)

a)

For a random variable x, which has Laplace distribution, $p(x) = \frac{1}{2}\exp{(-|x|)}$

Laplace distribution can be seen in Figure 5.

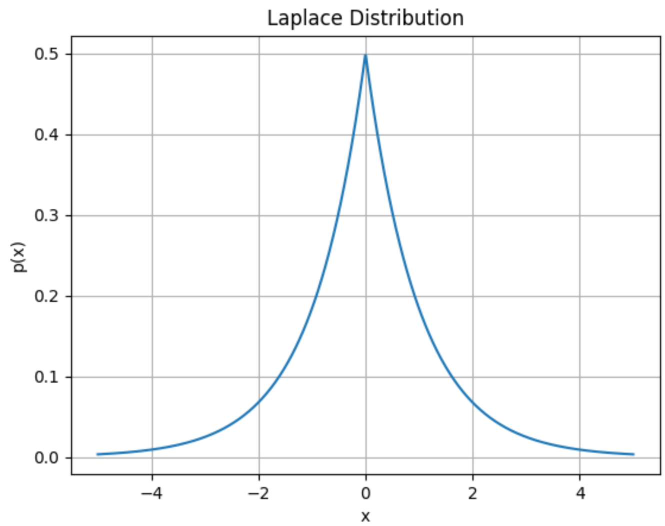

Figure 5: Laplace distribution

The probability that $x > 2$ can be calculated with integrating $p(x)$ from 2 to ∞, that is since x > 0,

$$p(x) = \frac{1}{2}\exp{(-x)}$$

$$\int_{2}^{\infty} p(x)dx = 0 + \frac{1}{2}\exp(-2) \cong 0.6767$$

b)

For a random variable x, which has binomial distribution, $p(x) = \binom{n}{x} p^x (1-p)^{n-x}$

$$E[x] = \sum_{x=0}^{n} x \binom{n}{x} p^x (1-p)^{n-x}$$

$$= \sum_{x=1}^{n} x \binom{n}{x} p^x (1-p)^{n-x}$$

$$= \sum_{x=1}^{n} n \binom{n-1}{x-1} p^x (1-p)^{n-x}$$

$$= np \sum_{x=1}^{n} \binom{n-1}{x-1} p^{x-1} (1-p)^{(n-1)-(x-1)}$$

$$= np \sum_{j=0}^{m} \binom{m}{j} p^{j-1} (1-p)^{m-j}$$

$$= np$$

$$E[x^2] = \sum_{x=0}^{n} x^2 \binom{n}{x} p^x (1-p)^{n-x}$$

$$= \sum_{x=0}^{n} nx \binom{n-1}{x-1} p^x (1-p)^{n-x}$$

$$= np \sum_{x=1}^{n} x \binom{n-1}{x-1} p^{x-1} (1-p)^{(n-1)-(x-1)}$$

$$= np \sum_{j=0}^{n} (j+1) \binom{m}{j} \; p^j (1-p)^{m-j}$$

$$= np \left(\sum_{j=0}^{n} j \binom{m}{j} \; p^j (1-p)^{m-j} + \sum_{j=0}^{n} \binom{m}{j} \; p^j (1-p)^{m-j} \right)$$

$$= np \left((n-1)p \sum_{j=1}^{n} \binom{m-1}{j-1} \; p^{j-1} (1-p)^{(m-1)-(j-1)} + \sum_{j=0}^{n} \binom{m}{j} \; p^j (1-p)^{m-j} \right)$$

$$= np\big((n-1)p + 1\big)$$

$$= n^2p^2 + np(1-p)$$

İbrahim Halil Bayzan

504221559

Appendix for Q2 and Q5

```
# Question 2
import numpy as np
import matplotlib.pyplot as plt

# This is a helper function to create a design matrix for polynomial regression
def design_matrix(x, degree):
    X = np.ones((len(x), 1))
    for i in range(1, degree + 1):
        X = np.hstack((X, np.power(x, i).reshape(-1, 1)))
    return X

# Closed-form solution to find the coefficients of the polynomial regression
def polynomial_regression_closed_form(x, y, degree):
    X = design_matrix(x, degree)
    coeffs = np.linalg.inv(X.T @ X) @ X.T @ y
    return coeffs

# Gradient Descent Algorithm
def gradient_descent(X, y, learning_rate, iterations, degree):
    m = len(y)
```

```
    theta = np.random.randn(degree + 1, 1)  # random initialization of
parameters

    for iteration in range(iterations):
        gradients = 2/m * X.T @ (X @ theta - y)
        theta = theta - learning_rate * gradients

    return theta

def calculate_expected_absolute_error(X, y, theta, verbose=False,
method='closed_form', data_name='', degree=None):
    y_pred = X @ theta
    error = np.abs(y_pred - y).mean()
    if verbose:
        if degree is not None:
            print(f'The expected absolute error for the {data_name} data using
{method} and degree {degree} is {error:.4f}')
        else:
            print(f'The expected absolute error for the {data_name} data using
{method} is {error:.4f}')
    return error

def plot_data(x, y, coeffs, title, suptitle, degree):
    for i in range(len(x)):
        plt.subplot(1, len(x), i + 1)
        plt.scatter(x[i], y[i], color='blue')
        x_values = np.linspace(-2, 2, 1000)
        y_values = np.polyval(coeffs[i][::-1], x_values)
        plt.plot(x_values, y_values, color='red')
        plt.title(title[i] + ' Case with Regression Line for Degree ' +
str(degree))
        plt.xlabel('x')
        plt.ylabel('y')
    plt.suptitle(suptitle)
    plt.show()

x_example = [None] * 3
y_example = [None] * 3

# Data for the 'outlier' case
x_example[0] = np.array([-2, -1.8, -1, -0.5, 0, 0.5, 1, 1.5, 1.7, 2])
y_example[0] = np.array([2, 1.5, 1, 0.5, 0, -0.5, -1, -1.5, 8, 9])

# Data for the 'Poly' (polynomial) case
x_example[1] = np.array([-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2])
```

```
y_example[1] = np.array([4, 2, 0.5, 0.2, 0, 0.2, 0.5, 2, 4])

# Data for the 'Linear' case
x_example[2] = np.array([-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2])
y_example[2] = np.array([-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2])

# Map of data names to their respective indices
data_names = {0: 'Outlier', 1: 'Poly', 2: 'Linear'}

degree = 4

coeffs_closed_form = [None] * 3
for i in range(3):
    coeffs_closed_form[i] = polynomial_regression_closed_form(x_example[i],
y_example[i], degree)

# Now set up the gradient descent parameters
learning_rate = 0.01
iterations = 1000

# Prepare the design matrix for gradient descent
X_design = [None] * 3
theta_gradient_descent = [None] * 3
for i in range(3):
    X_design[i] = design_matrix(x_example[i], degree)
    theta_gradient_descent[i] = gradient_descent(X_design[i],
y_example[i].reshape(-1, 1), learning_rate, iterations, degree)

# Calculate expected absolute error for both models
error_closed_form = [None] * 3
error_gradient_descent = [None] * 3

# print('Expected Absolute Errors:' + ' for degree ' + str(degree))
for i in range(3):
    error_closed_form[i] = calculate_expected_absolute_error(X_design[i],
y_example[i].reshape(-1, 1), coeffs_closed_form[i], True, 'closed_form',
data_names[i], degree)
    error_gradient_descent[i] = calculate_expected_absolute_error(X_design[i],
y_example[i].reshape(-1, 1), theta_gradient_descent[i], True,
'gradient_descent', data_names[i])

# Plot for the 'Outlier' case
plt.figure(figsize=(18, 6))
plot_data(x_example, y_example, coeffs_closed_form, data_names, 'Closed Form
Solution', degree)
```

```python
# Plot the gradient descent solution along with the data points in a 3 x 1
subplot
plt.figure(figsize=(18, 6))
plot_data(x_example, y_example, theta_gradient_descent, data_names, 'Gradient
Descent Solution', degree)
```

```python
# Question 5

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def laplace_distribution(x):
    return 0.5 * np.exp(-np.abs(x))

x_values = np.linspace(-5, 5, 1000)
y_values = laplace_distribution(x_values)
plt.plot(x_values, y_values)
plt.xlabel('x')
plt.ylabel('p(x)')
plt.title('Laplace Distribution')
plt.grid(True)
plt.show()

prob_x_gt_2 = quad(laplace_distribution, 2, np.inf)
print('The probability that x > 2 is', prob_x_gt_2[0])
```