Machine Learning Homework 1

Question 1)

a)

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

Which provides the probability mass function of Z

b)

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

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

Also integrate over Z on to get

|  |  |  |  |
| --- | --- | --- | --- |
| Y | -1 | 0 | 1 |
|  | 0.45 | 0.25 | 0.3 |

Lastly, integrate over Y on to get

|  |  |  |
| --- | --- | --- |
| Z | 1 | 2 |
|  | 0.3 | 0.7 |

For independence we need to hold.

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  *,* thus, we can conclude these normal variables are not independent

Question 2)

a)

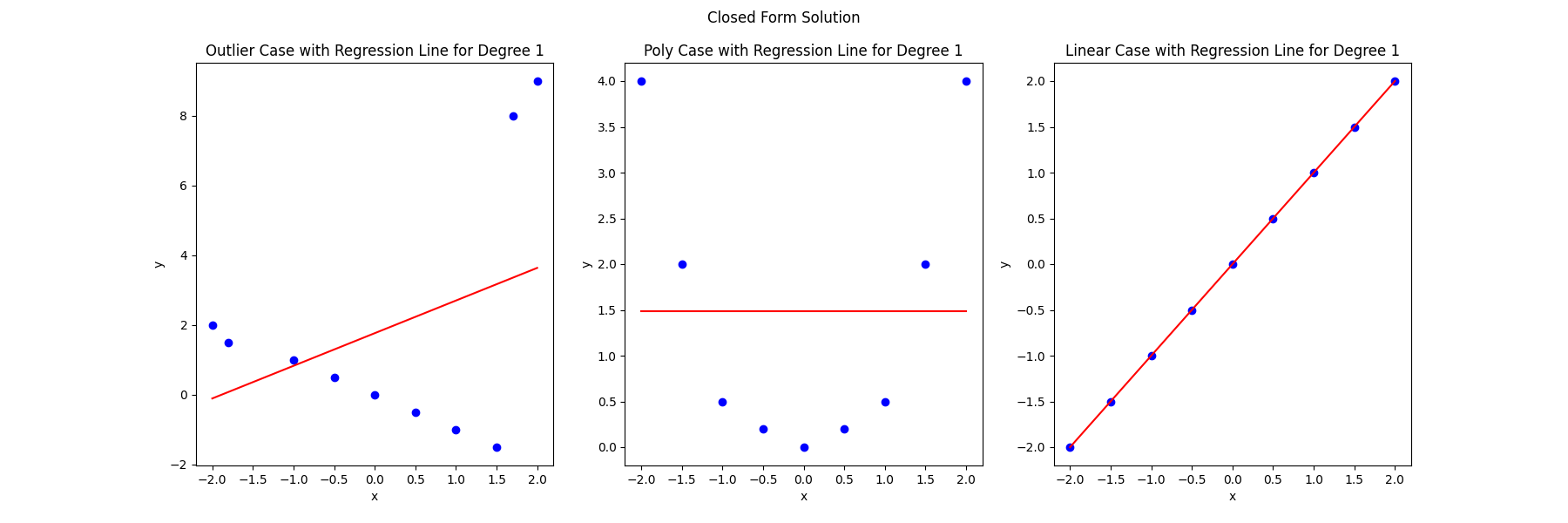


Figure : Closed form solution for linear regression

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)

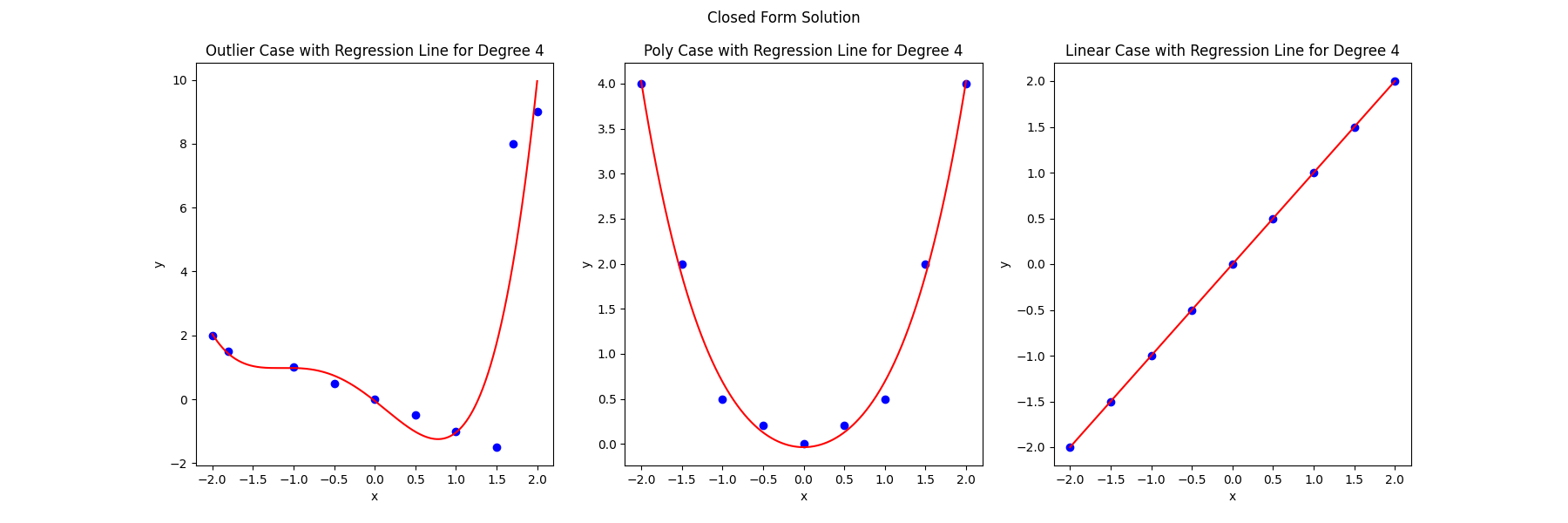


Figure : Closed form solution for fourth order polynomial

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)

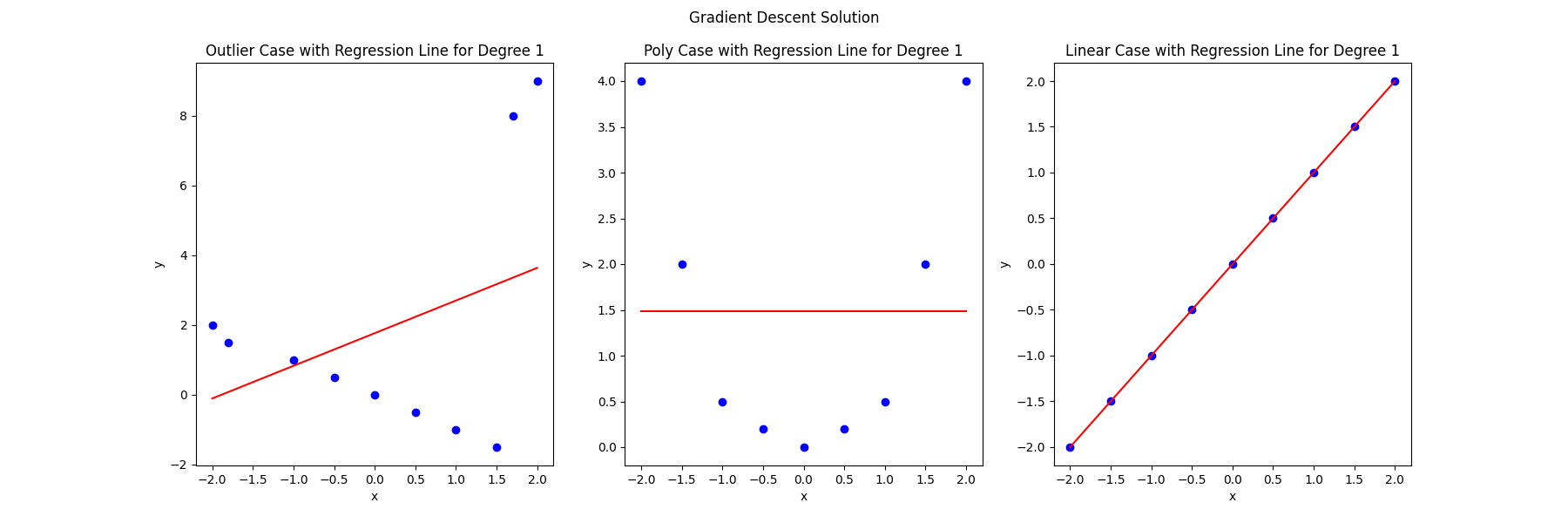


Figure : Linear regression with gradient descent

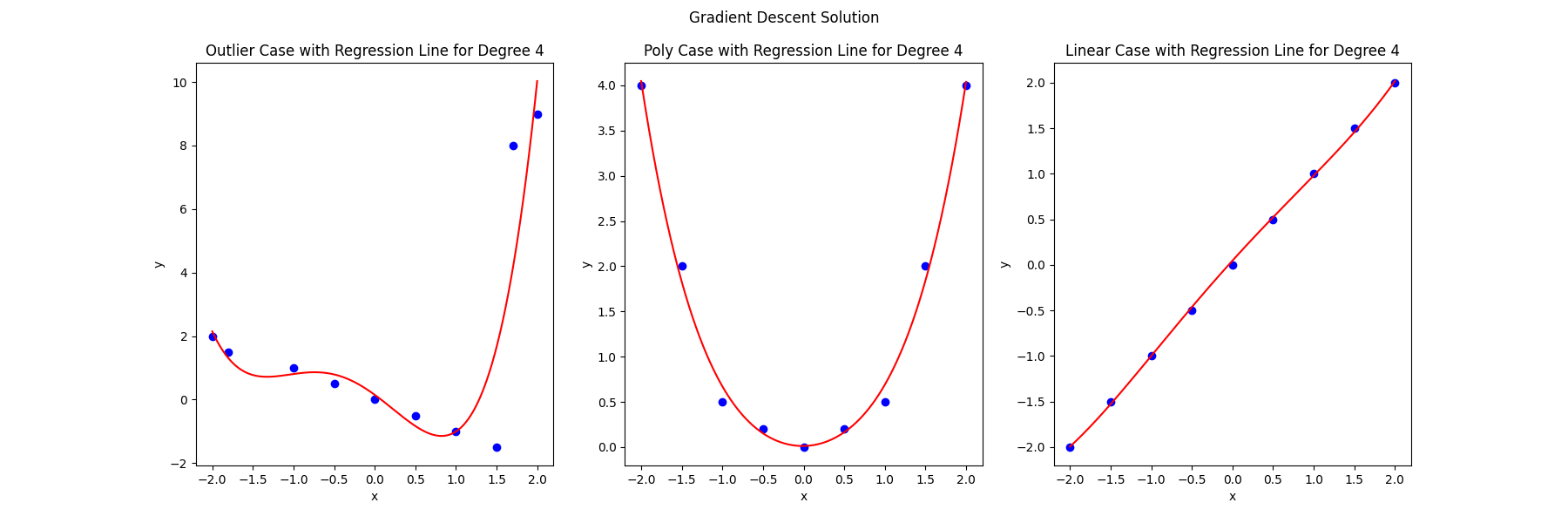


Figure : Fourth order regression with gradient descent

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

It seems like gradient descent

Question 3)

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

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 can be calculated with

Based on this, selecting makes the most sense.

Question 5)

a)

For a random variable x, which has Laplace distribution,

Laplace distribution can be seen in Figure 5.

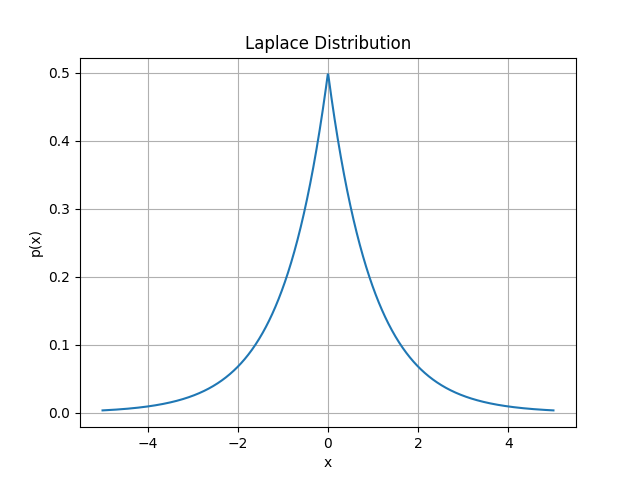


Figure : Laplace distribution

The probability that can be calculated with integrating from 2 to , that is since x > 0,

b)

For a random variable x, which has binomial distribution,

**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)  # 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) |
| # 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]) |