Practical AI and MLOps : Assignment 1

Import libraries and fetch data

```
## DO NOT EDIT
import numpy as np
import pickle

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1-4LzDDmpH-v9m
!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1-AtCVwrcpgFCv

X1_train, X1_test, Y1_train, Y1_test, X2_train, X2_test, Y2_train, Y2_test = [[] for _ in

with open("/content/linear_regression.pkl", 'rb') as f:
    X1_train, X1_test, Y1_train, Y1_test = pickle.load(f)

with open("/content/logistic_regression.pkl", 'rb') as f:
    X2_train, X2_test, Y2_train, Y2_test = pickle.load(f)
```

```
--2023-09-03 13:22:27--  https://docs.google.com/uc?export=download&id=1-4LzDDmpH-v9
Resolving docs.google.com (docs.google.com)... 142.251.2.138, 142.251.2.100, 142.251
Connecting to docs.google.com (docs.google.com)|142.251.2.138|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-10-0s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7
Warning: wildcards not supported in HTTP.
--2023-09-03 13:22:27--  https://doc-10-0s-docs.googleusercontent.com/docs/securesc/l
Resolving doc-10-0s-docs.googleusercontent.com (doc-10-0s-docs.googleusercontent.com
Connecting to doc-10-0s-docs.googleusercontent.com (doc-10-0s-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: 1847 (1.8K) [application/octet-stream]
Saving to: ‘linear_regression.pkl’

linear_regression.p 100%[===================>]   1.80K  --.-KB/s    in 0s

2023-09-03 13:22:27 (113 MB/s) - ‘linear_regression.pkl’ saved [1847/1847]

--2023-09-03 13:22:27--  https://docs.google.com/uc?export=download&id=1-AtCVwrcpgFC
Resolving docs.google.com (docs.google.com)... 142.251.2.138, 142.251.2.100, 142.251
Connecting to docs.google.com (docs.google.com)|142.251.2.138|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0o-0s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7
Warning: wildcards not supported in HTTP.
--2023-09-03 13:22:28--  https://doc-0o-0s-docs.googleusercontent.com/docs/securesc/l
Resolving doc-0o-0s-docs.googleusercontent.com (doc-0o-0s-docs.googleusercontent.com
Connecting to doc-0o-0s-docs.googleusercontent.com (doc-0o-0s-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: 17088 (17K) [application/octet-stream]
```

✓ 0s completed at 6:55 PM

```
logistic_regression 100%[===================>]  16.69K  --.-KB/s    in 0s

2023-09-03 13:22:28 (113 MB/s) - ‘logistic_regression.pkl’ saved [17088/17088]
```

```
X1_train = np.resize(X1_train, (len(X1_train), 1))
X1_test =  np.resize(X1_test,  (len(X1_test), 1))
Y1_train = np.resize(Y1_train, (len(Y1_train), 1))
Y1_test =  np.resize(Y1_test,  (len(Y1_test), 1))
print(X1_train.shape)
print(X1_test.shape)
print(Y1_train.shape)
print(Y1_test.shape)
```

```
(70, 1)
(30, 1)
(70, 1)
(30, 1)
```

```
Y2_train = np.resize(Y2_train, (len(Y2_train), 1))
Y2_test =  np.resize(Y2_test,  (len(Y2_test), 1))
print(X2_train.shape)
print(X2_test.shape)
print(Y2_train.shape)
print(Y2_test.shape)
```

```
(70, 20)
(30, 20)
(70, 1)
(30, 1)
```

```
print((X1_train**-1).shape)
print((X1_train).shape)
print(X1_test[0:5])
print(Y1_train.shape)
print(Y1_test[0:5])
```

```
(70, 1)
(70, 1)
[[ 6.26850875]
 [-5.75615029]
 [-5.98513959]
 [-9.76278328]
 [-6.04948187]]
(70, 1)
[[ 291.87875938]
 [-163.34294373]
 [-184.56229109]
 [-844.96063406]
```

```
      [-190.84148613]]


print(X2_train[0])
print(X2_test[0])
print(Y2_train[0:5])
print(Y2_test[0:5])

     [-0.67857138  2.7670727   2.03131422  0.3322455  -1.2361682  -0.48420497
      -1.6533102   0.17205938 -1.14020401 -1.02420001 -1.41826043  0.26769137
       0.39235085  0.49597238 -2.04503375  0.75743566  2.50947819  0.53326592
       0.92774064 -1.3631717 ]
     [ 1.59980278  0.34209474  1.39349879 -1.44936033 -0.94399061  0.00816769
      -0.87028399 -1.49635521  0.66575811  1.06385996 -1.73467213  0.05441906
      -0.05293944  0.82191122  0.04120958 -1.71024922  0.49349984  0.124863
      -0.22401342  0.47682353]
     [[0]
      [1]
      [1]
      [0]
      [0]]
     [[0]
      [0]
      [0]
      [0]
      [1]]
```

Problem 1: Linear Regression (2 Marks)

Train a linear regression model for the above set of data. Use MSE(Mean Squared Error) as the loss function.

Print out the train accuracy and test accuracy.

Data: X1_train, X1_test, Y1_train, Y1_test

```
### Write your code here
## Solve the question using normal equation
# Normal equation is Y1_train = regression coefficients or theta * X1_train
# theta = Y1_train * inv(X1_train)
theta = np.dot(Y1_train.T,(X1_train)**-1)
print("theta:", theta)

## Define the hypothesis function for linear regression
#Answer: Hypothesis y_hat =  beta_1*X1_train
#Cost_function = (np.sum(Y1_train - (bias + weight_1*X1_train) ))/len(Y1_train)
#print(Y1_train[0:5])

## Use linear regression to train the model
```

```
Regression_coefficient = (np.dot(X1_train.T,X1_train)**-1)*np.dot(X1_train.T,Y1_train)
print("Regression_coefficient: ", Regression_coefficient)
Y1_Lin_NoBias = Regression_coefficient*X1_train
print(Y1_Lin_NoBias.shape)
## Compute the training and testing loss using MSE error
Post_training_error = np.sum((Y1_train - Regression_coefficient*X1_train)**2)/len(Y1_train
#print("Post Training Error: ", Post_training_error)

test_error = np.sum((Y1_test - Regression_coefficient*X1_test)**2)/len(Y1_test)
## Print the training and testing loss
## update the below two variables to print the model's loss
loss_train = Post_training_error
loss_test = test_error

print("Linear Regression")
print("Training Loss: ", loss_train)
print("Testing  Loss: ", loss_test)
```

```
theta: [[2781.59003867]]
Regression_coefficient:  [[66.1311054]]
(70, 1)
Linear Regression
Training Loss:  27415.39655697824
Testing  Loss:  44432.264978342
```

```
ones_column = np.ones((X1_train.shape[0], 1))
X1_train_mod = np.hstack((ones_column, X1_train))

ones_column = np.ones((X1_test.shape[0], 1))
X1_test_mod = np.hstack((ones_column, X1_test))

Y1_train_mod = np.resize(Y1_train, (len(Y1_train), 1))
Y1_test_mod =  np.resize(Y1_test,  (len(Y1_test), 1))


#This code snippet is to implement simple linear regression by implementing bias along wit
#X1_train and X1_test has been modified to add 2nd column as ones, because x value is 1 fo

Regression_coefficients = np.dot(np.linalg.pinv(np.dot(X1_train_mod.T, X1_train_mod)), np.
print("Regression_coefficient: ", Regression_coefficients)
Y1_Lin_Bias = np.dot(X1_train_mod, Regression_coefficients)
print(Y1_Lin_Bias.shape)

## Compute the training and testing loss using MSE error
Post_training_error = np.sum((Y1_train_mod - np.dot(X1_train_mod, Regression_coefficients)
print("Post Training Error: ", Post_training_error)
test_error = np.sum((Y1_test_mod - np.dot(X1_test_mod, Regression_coefficients))**2)/len(Y

## Print the training and testing loss
## update the below two variables to print the model's loss
```

```
loss_train = Post_training_error
loss_test = test_error
print("Linear Regression")
print("Training Loss: ", loss_train)
print("Testing  Loss: ", loss_test)
```

```
Regression_coefficient:  [[47.21637965]
 [65.15344921]]
(70, 1)
Post Training Error:  25222.28423671403
Linear Regression
Training Loss:  25222.28423671403
Testing  Loss:  41716.624929654376
```

Problem 2: Polynomial Regression (2 Marks)

Solve the above problem using atleast 3 different hypothesis functions with polynomial degree 2, 3 and 4.

Print out the train accuracy and test accuracy. Write your inference on the results.

Data: X1_train, X1_test, Y1_train, Y1_test

```
ones_column = np.ones((X1_train.shape[0], 1))
X1_train_poly2 = np.hstack((ones_column, X1_train, X1_train**2))
print(X1_train_poly2[0:5])

ones_column = np.ones((X1_test.shape[0], 1))
X1_test_poly2 = np.hstack((ones_column, X1_test, X1_test**2))
print(X1_test_poly2[0:3])

Y1_train_poly2 = np.resize(Y1_train, (len(Y1_train), 1))
Y1_test_poly2 =  np.resize(Y1_test,  (len(Y1_test), 1))
print(X1_train_poly2.shape)
print(X1_test_poly2.shape)
print(Y1_train_poly2.shape)
print(Y1_test_poly2.shape)
```

```
[[ 1.         -8.85376401 78.38913723]
 [ 1.         -2.5895495   6.70576661]
 [ 1.          8.08186325 65.31651353]
 [ 1.         -4.97550386 24.75563861]
 [ 1.          5.49334111 30.17679656]]
[[ 1.          6.26850875 39.29420194]
 [ 1.         -5.75615029 33.13326616]
 [ 1.         -5.98513959 35.82189591]]
(70, 3)
(30, 3)
(70, 1)
```

```
(30, 1)
```

Polynomial Degree 2

```
### Write your code here
## Define the hypothesis function
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2]]
#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2]
#Hypothesis = [Beta] * X1_train


## Use polynomial regression to train the model
#Using Normal equation, Regression_cofficients = inv(X1_train.T*X1_train)*X1_train.T*Y1_tr
Regression_coefficients_poly2 = np.dot(np.linalg.pinv(np.dot(X1_train_poly2.T, X1_train_po
print("Regression_coefficient: ", Regression_coefficients_poly2)
Y1_Poly2 = np.dot(X1_train_poly2, Regression_coefficients_poly2)
print(Y1_Poly2.shape)

## Compute the training and testing loss
Training_error = np.sum((Y1_train_poly2 - np.dot(X1_train_poly2, Regression_coefficients_p
test_error = np.sum((Y1_test_poly2 - np.dot(X1_test_poly2, Regression_coefficients_poly2))


## Print the training and testing loss
## update the below two variables to print the model's loss
loss_train = Training_error
loss_test = test_error

print("Polynomial Degree 2")
print("Training loss: ", loss_train)
print("Testing  loss: ", loss_test)
```

```
Regression_coefficient:  [[-5.81412195]
 [64.01720536]
 [ 1.42086134]]
(70, 1)
Polynomial Degree 2
Training loss:  23308.50305148525
Testing  loss:  36380.348114078115
```

```
ones_column = np.ones((X1_train.shape[0], 1))
X1_train_poly3 = np.hstack((ones_column, X1_train, X1_train**2, X1_train**3))
print(X1_train_poly3[0:5])

ones_column = np.ones((X1_test.shape[0], 1))
X1_test_poly3 = np.hstack((ones_column, X1_test, X1_test**2, X1_test**3))
print(X1_test_poly3[0:3])
```

```
Y1_train_poly3 = np.resize(Y1_train, (len(Y1_train), 1))
Y1_test_poly3 =  np.resize(Y1_test,  (len(Y1_test), 1))
print(X1_train_poly3.shape)
print(X1_test_poly3.shape)
print(Y1_train_poly3.shape)
print(Y1_test_poly3.shape)
```

```
[[   1.           -8.85376401   78.38913723 -694.03892235]
 [   1.           -2.5895495     6.70576661  -17.36491455]
 [   1.            8.08186325   65.31651353  527.87913012]
 [   1.           -4.97550386   24.75563861 -123.17177536]
 [   1.            5.49334111   30.17679656  165.77143712]]
[[   1.            6.26850875   39.29420194  246.31604869]
 [   1.           -5.75615029   33.13326616 -190.72005959]
 [   1.           -5.98513959   35.82189591 -214.39904741]]
(70, 4)
(30, 4)
(70, 1)
(30, 1)
```

Polynomial Degree 3

```
### Write your code here
## Define the hypothesis function
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2] + Beta_3
#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2, Beta_3]
#Hypothesis = [Beta] * X1_train


## Use polynomial regression to train the model
#Using Normal equation, Regression_cofficients = inv(X1_train.T*X1_train)*X1_train.T*Y1_tr
Regression_coefficients_poly3 = np.dot(np.linalg.pinv(np.dot(X1_train_poly3.T, X1_train_po
print("Regression_coefficient: ", Regression_coefficients_poly3)

Y1_Poly3 = np.dot(X1_train_poly3, Regression_coefficients_poly3)
print(Y1_Poly3.shape)

## Compute the training and testing loss
Training_error = np.sum((Y1_train_poly3 - np.dot(X1_train_poly3, Regression_coefficients_p
test_error = np.sum((Y1_test_poly3 - np.dot(X1_test_poly3, Regression_coefficients_poly3))


## Print the training and testing loss
## update the below two variables to print the model's loss
loss_train = Training_error
loss_test = test_error

print("Polynomial Degree 3")
print("Training Loss: ", loss_train)
print("Testing Loss: ", loss_test)
```

```
print( Testing  Loss:  , loss_test)
```

```
Regression_coefficient:  [[-4.29736247e-11]
 [ 1.00000000e+00]
 [ 1.00000000e+00]
 [ 1.00000000e+00]]
(70, 1)
Polynomial Degree 3
Training Loss:  7.92854562673667e-22
Testing  Loss:  8.283010751786646e-22
```

Polynomial Degree 4

```
ones_column = np.ones((X1_train.shape[0], 1))
X1_train_poly4 = np.hstack((ones_column, X1_train, X1_train**2, X1_train**3, X1_train**4))
print(X1_train_poly4[0:5])

ones_column = np.ones((X1_test.shape[0], 1))
X1_test_poly4 = np.hstack((ones_column, X1_test, X1_test**2, X1_test**3, X1_test**4))
print(X1_test_poly4[0:3])

Y1_train_poly4 = np.resize(Y1_train, (len(Y1_train), 1))
Y1_test_poly4 =  np.resize(Y1_test,  (len(Y1_test), 1))
print(X1_train_poly4.shape)
print(X1_test_poly4.shape)
print(Y1_train_poly4.shape)
print(Y1_test_poly4.shape)
```

```
[[ 1.00000000e+00 -8.85376401e+00  7.83891372e+01 -6.94038922e+02
   6.14485684e+03]
 [ 1.00000000e+00 -2.58954950e+00  6.70576661e+00 -1.73649146e+01
   4.49673058e+01]
 [ 1.00000000e+00  8.08186325e+00  6.53165135e+01  5.27879130e+02
   4.26624694e+03]
 [ 1.00000000e+00 -4.97550386e+00  2.47556386e+01 -1.23171775e+02
   6.12841643e+02]
 [ 1.00000000e+00  5.49334111e+00  3.01767966e+01  1.65771437e+02
   9.10639051e+02]]
[[ 1.00000000e+00  6.26850875e+00  3.92942019e+01  2.46316049e+02
   1.54403431e+03]
 [ 1.00000000e+00 -5.75615029e+00  3.31332662e+01 -1.90720060e+02
   1.09781333e+03]
 [ 1.00000000e+00 -5.98513959e+00  3.58218959e+01 -2.14399047e+02
   1.28320823e+03]]
(70, 5)
(30, 5)
(70, 1)
(30, 1)
```

```
### Write your code here
## Define the hypothesis function
```

```
## Define the hypothesis function
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2] + Beta_3
#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2, Beta_3]
#Hypothesis = [Beta] * X1_train


## Use polynomial regression to train the model
#Using Normal equation, Regression_cofficients = inv(X1_train.T*X1_train)*X1_train.T*Y1_tr
Regression_coefficients_poly4 = np.dot(np.linalg.pinv(np.dot(X1_train_poly4.T, X1_train_po
print("Regression_coefficient: ", Regression_coefficients_poly4)
Y1_Poly4 = np.dot(X1_train_poly4, Regression_coefficients_poly4)
print(Y1_Poly4.shape)

## Compute the training and testing loss
Training_error = np.sum((Y1_train_poly4 - np.dot(X1_train_poly4, Regression_coefficients_p
test_error = np.sum((Y1_test_poly4 - np.dot(X1_test_poly4, Regression_coefficients_poly4))


## Print the training and testing loss
## update the below two variables to print the model's loss
loss_train = Training_error
loss_test = test_error

print("Polynomial Degree 4")
print("Training Loss: ", loss_train)
print("Testing  Loss: ", loss_test)
```

```
Regression_coefficient:  [[1.04554861e-10]
 [1.00000000e+00]
 [1.00000000e+00]
 [1.00000000e+00]
 [4.63908321e-14]]
(70, 1)
Polynomial Degree 4
Training Loss:  2.786446139990528e-21
Testing  Loss:  2.7485570890912254e-21
```

```
# Calculate the coefficients (Theta) using the normal equation
X_transpose_X = np.dot(X1_train_poly4.T, X1_train_poly4)
X_transpose_y = np.dot(X1_train_poly4.T, Y1_train_poly4)
coefficients = np.linalg.solve(X_transpose_X, X_transpose_y)

# The coefficients represent the coefficients of the polynomial terms in increasing order
print("Coefficients for Polynomial Regression:")
for i, coeff in enumerate(coefficients):
    print(f"Theta_{i}:", coeff)

## Compute the training and testing loss
Training_error = np.sum((Y1_train_poly4 - np.dot(X1_train_poly4, Regression_coefficients_p
test_error = np.sum((Y1_test_poly4 - np.dot(X1_test_poly4, Regression_coefficients_poly4))
```

```
## Print the training and testing loss
## update the below two variables to print the model's loss
loss_train = Training_error
loss_test = test_error

print("Polynomial Degree 4")
print("Training Loss: ", loss_train)
print("Testing  Loss: ", loss_test)
```

```
Coefficients for Polynomial Regression:
Theta_0: [-1.14347785e-12]
Theta_1: [1.]
Theta_2: [1.]
Theta_3: [1.]
Theta_4: [-9.58115424e-16]
Polynomial Degree 4
Training Loss:  2.786446139990528e-21
Testing  Loss:  2.7485570890912254e-21
```

Inference

The training error and testing error changes with degree of polynomial. Between Simple linear regression, the hypothesis function without bias and with bias, the latter showed higher accuracy in terms of lower MSE than without bias. This confirms the observation that simple linear regression with bias is always better than the one without bias. Comparing Polynomial regression with degree 2,3 and 4, the training error and test error for degree 3 is the lowest, followed by degree 4 and then degree 2. Training and test error are both ~e-21. So the model isnt overfitting or underfitting.

Problem 3: Overfitting and Underfitting (2 Marks)

Let us say we have a dataset with little noise. Then a model is underfitting when:

a) both the train and test errors are high

b) train error is low but test error is high

c) train error is high but the test error is low

d) both train and test errors are low

If we choose the parameters of a model to get the best overfitting/underfitting tradeoff, we will always get a zero test error.

a) True

b) False

State which of the below statements is false with respect to underfitting vs overfitting.

a) If the training set performance is roughly equal, underfitting is generally better.

b) An underfit model is simpler and can usually be improved by looking at where it fits badly.

c) An overfit model is often easier to change because it is easy to know where to start simplifying.

d) Neither overfitting nor underfitting is desirable.

Write your answers below

For the dataset with little noise, it underfits **a) when both the train and test errors are high**

If we choose the parameters of a model to get the best overfitting/underfitting tradeoff, we will always get a zero test error. **FALSE**

State which of the below statements is false with respect to underfitting vs overfitting:

d) Neither overfitting nor underfitting is desirable.

Problem 4: Regularization (2 Marks)

Solve the problem using regularization (Lasso, Ridge and Elastic net) on the 3 polynomial functions defined in **Problem 2**.

Print out the train and test accuracy. Write your inference on the results

Data: X1_train, X1_test, Y1_train, Y1_test

Polynomial Degree 2

[ ] ↳ *6 cells hidden*

Polynomial Degree 3

[ ] ↳ *6 cells hidden*

Polynomial Degree 4

Lasso Regularization

```
### Write your code here
## Define a new hypothesis function by adding a Lasso regularizer
## to the 2nd order hypothesis function that you defined earlier

#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2]
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2]]
#Hypothesis = [Beta] * X1_train

## Use polynomial regression to train the model

# Define (hyper)parameters for Gradient descent algorithm
alpha = 0.00000001  # Learning rate
lambda_ = 1  # Regularization parameter (lambda is L2 penalty)
num_iterations = 50000 #Nr of iteration for GD to run
m, n = X1_train_poly4.shape # no_of_training_examples, no_of_features

#print(X1_test_poly3.T.shape)
# Initialize coefficients
Beta = np.zeros(n)  #  3 coefficients [Beta_0, Beta_1, Beta_2]
Beta = np.resize(Beta, (len(Beta), 1))
#print(Beta.shape)

# Gradient Descent
for _ in range(num_iterations):
# Calculate predictions
    predictions = np.dot(X1_train_poly4, Beta)

# Compute errors
    errors = predictions - Y1_train_poly4
    #print(errors.shape)
# Compute gradient with regularization for Lasso or L1 regularization
    L1_penalty = (lambda_ / m) * np.sign(Beta)  # Derivative of |theta|
#    print(L1_penalty.shape)
    gradient = (1/m) * np.dot(X1_train_poly4.T, errors) + L1_penalty
#    print(gradient.shape)
# Update coefficients meaning bias and weights
    Beta -= alpha*gradient
# Print coefficients
print("Lasso Coefficients:", Beta)

## Compute the training and testing accuracy
Training_error = np.sum((Y1_train_poly4 - np.dot(X1_train_poly4, Beta))**2)/len(Y1_train_p
Test_error = np.sum((Y1_test_poly4 - np.dot(X1_test_poly4, Beta))**2)/len(Y1_test_poly4)

## Print the training and testing accuracy
## update the below two variables to print the model's accuracy
accuracy_train = Training_error
accuracy_test = Test_error
```

```
print("Polynomial Degree 4 : Lasso Regularization")
print("Training Accuracy: ", accuracy_train)
print("Testing  Accuracy: ", accuracy_test)
```

```
Lasso Coefficients: [[0.00346592]
 [0.01753696]
 [0.0774439 ]
 [1.01313201]
 [0.01172984]]
Polynomial Degree 4 : Lasso Regularization
Training Accuracy:  144.73021732768845
Testing  Accuracy:  222.93692085429433
```

Ridge Regularization

```
### Write your code here
## Define a new hypothesis function by adding a Lasso regularizer
## to the 2nd order hypothesis function that you defined earlier

#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2]
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2] + Beta_3
#Hypothesis = [Beta] * X1_train_poly4
#X1_test_poly4 = np.hstack((ones_column, X1_test, X1_test**2, X1_test**3, X1_test**4))

## Use polynomial regression to train the model

# Define (hyper)parameters for Gradient descent algorithm
alpha = 0.0000001  # Learning rate
lambda_ = 10 # Regularization parameter (lambda is L2 penalty)
num_iterations = 50000 #Nr of iteration for GD to run
m, n = X1_train_poly4.shape # no_of_training_examples, no_of_features

#print(X1_test_poly2.T.shape)
# Initialize coefficients
Beta = np.zeros(n)  #  3 coefficients [Beta_0, Beta_1, Beta_2]
Beta = np.resize(Beta, (len(Beta), 1))
#print(Beta.shape)

# Gradient Descent
for _ in range(num_iterations):
# Calculate predictions
    predictions = np.dot(X1_train_poly4, Beta)


# Compute errors
    errors = predictions - Y1_train_poly4
    #print(errors.shape)
# Compute gradient with regularization (except for theta_0)
```

```
    gradient = (1/m) * np.dot(X1_train_poly4.T, errors) + (lambda_/m) * Beta
#   print(gradient.shape)
    gradient[0] = (1/m) *np.sum(errors)  # Exclude regularization for theta_0
    #print(gradient)
# Update coefficients meaning bias and weights
    Beta -= alpha*gradient
# Print coefficients
print("Ridge Coefficients:", Beta)

## Compute the training and testing accuracy
Training_error = np.sum((Y1_train_poly4 - np.dot(X1_train_poly4, Beta))**2)/len(Y1_train_p
Test_error = np.sum((Y1_test_poly4 - np.dot(X1_test_poly4, Beta))**2)/len(Y1_test_poly4)

## Print the training and testing accuracy
## update the below two variables to print the model's accuracy
accuracy_train = Training_error
accuracy_test = Test_error

print("Polynomial Degree 4 : Ridge Regularization")
print("Training Accuracy: ", accuracy_train)
print("Testing  Accuracy: ", accuracy_test)
```

```
Ridge Coefficients: [[0.02489389]
 [0.04675362]
 [0.55622444]
 [1.01290007]
 [0.00562428]]
Polynomial Degree 4 : Ridge Regularization
Training Accuracy:  37.84812673901232
Testing  Accuracy:  56.78742947883662
```

Elastic net Regularization

```
### Write your code here
## Define a new hypothesis function by adding a Lasso regularizer
## to the 2nd order hypothesis function that you defined earlier

#Regression_cofficients = Beta = [Beta_0, Beta_1, Beta_2, Beta_3, Beta_4]
#Hypothesis = [Beta_0 * X1_train[0] + Beta_1 * X1_train[1] + Beta_2 * X1_train[2] + Beta_3
#Hypothesis = [Beta] * X1_train_poly4
#X1_test_poly4 = np.hstack((ones_column, X1_test, X1_test**2, X1_test**3, X1_test**4))
## Use polynomial regression to train the model

# Define (hyper)parameters for Gradient descent algorithm
alpha = 0.000001  # Learning rate
lambda_l1 = 1  # L1 regularization parameter is lasso parameter (for feature selection)
lambda_l2 = 10  # L2 regularization parameter is ridge parameter (for handling correlated
num_iterations = 50000 #Nr of iteration for GD to run
m, n = X1_train_poly4.shape # no_of_training_examples, no_of_features
```

```
#print(X1_test_poly2.T.shape)
# Initialize coefficients
Beta = np.zeros(n)  #  5 coefficients [Beta_0, Beta_1, Beta_2,Beta_3, Beta_4]
Beta = np.resize(Beta, (len(Beta), 1))
print(Beta.shape)

# Gradient Descent
for _ in range(num_iterations):
# Calculate predictions
    predictions = np.dot(X1_train_poly4, Beta)



# Compute errors
    errors = predictions - Y1_train_poly4
    #print(errors.shape)
# Compute gradient with regularization (except for Beta_0) with Elastic Net regularization
    L1_penalty = (lambda_l1 / m) * np.sign(Beta[1:])  # Exclude Bias term from L1 regulari
    L2_penalty = (lambda_l2 / m) * Beta[1:]  # Exclude Bias term from L2 regularization
    gradient = (1/m) * np.dot(X1_train_poly4.T, errors)
    gradient[1:] += L1_penalty + L2_penalty  # Apply both L1 and L2 regularization
# Update coefficients meaning bias and weights
    Beta -= alpha*gradient
# Print coefficients
print("Elastic Net Coefficients:", Beta)

## Compute the training and testing accuracy
Training_error = np.sum((Y1_train_poly4 - np.dot(X1_train_poly4, Beta))**2)/len(Y1_train_p
Test_error = np.sum((Y1_test_poly4 - np.dot(X1_test_poly4, Beta))**2)/len(Y1_test_poly4)

## Print the training and testing accuracy
## update the below two variables to print the model's accuracy
accuracy_train = Training_error
accuracy_test = Test_error

print("Polynomial Degree 4 : Elastic net Regularization")
print("Training Accuracy: ", accuracy_train)
print("Testing  Accuracy: ", accuracy_test)
```

```
(5, 1)
<ipython-input-230-d839b4eec0b9>:39: RuntimeWarning: invalid value encountered in sul
  Beta -= alpha*gradient
Elastic Net Coefficients: [[nan]
 [nan]
 [nan]
 [nan]
 [nan]]
Polynomial Degree 4 : Elastic net Regularization
Training Accuracy:  nan
Testing  Accuracy:  nan
```

Double-click (or enter) to edit

Inference

Write your inference here

...

Problem 5: Logistic Regression (2 Marks)

Train a logistic regression model. Print the F1 score, accuracy and confusion matrix for both training and testing.

Data: X2_train, X2_test, Y2_train, Y2_test

```
print(X2_train.shape)
print(X2_test.shape)
print(Y2_train.shape)
print(Y2_test.shape)
```

```
(70, 20)
(30, 20)
(70, 1)
(30, 1)
```

```
### Write your code here

# Hyperparameters
learning_rate = 0.001
num_iterations = 1000

X2_train_Bias = np.column_stack([np.ones(X2_train.shape[0]), X2_train])
#print("X2_train_Bias: ",X2_train_Bias.shape)
#print(X2_train_Bias[0])

X2_test_Bias = np.column_stack([np.ones(X2_test.shape[0]), X2_test])
#print("X2_test_Bias:",X2_test_Bias.shape)

# Initialize coefficients (including the intercept)
m, n = X2_train_Bias.shape # no_of_training_examples, no_of_features
Beta = np.zeros(n)   # Regression Coefficients
Beta = np.resize(Beta, (len(Beta), 1))
#print("Beta:",Beta.shape)
# Add a column of ones for the Bias/intercept term in X2_train and X2_test

# Sigmoid function (logistic function)
def sigmoid(z):
```

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

# Gradient Descent
for _ in range(num_iterations):
    # Calculate the linear combination of features and coefficients
    linear_combination = np.dot(X2_train_Bias, Beta)
# Apply the sigmoid function to get probabilities
    probabilities = sigmoid(linear_combination)

# Compute errors (log loss)
    errors = -(Y2_train - probabilities)

# Compute gradients
    gradient = (1/m) * np.dot(X2_train_Bias.T, errors)
#    print(gradient.shape)
# Update coefficients (including the intercept)
    Beta -= learning_rate * gradient

# Extract the intercept and feature coefficients
intercept = Beta[0]
Logistic_regression_coefficients = Beta[1:]
#print(Beta.shape)

# Print the logistic regression coefficients
#print("Logistic Regression Coefficients:")
#print("Intercept:", intercept)
#print("Coefficients:", Logistic_regression_coefficients)

## Compute f1 score, accuracy and confusion matrix for training

Y2_train_pred = np.dot(X2_train_Bias, Beta)
#print(Y2_train_pred.shape)
#print("Y2_train_pred:",Y2_train_pred[0:20])

# Set a threshold (usually 0.5 for binary classification)
threshold = 0.5

# Convert probabilities to binary class labels
predicted_labels = (Y2_train_pred >= threshold).astype(int)
#print(predicted_labels[0:20])

# Compute true positives (TP), true negatives (TN), false positives (FP), and false negati
# For Training data
TP = np.sum((predicted_labels == 1) & (Y2_train == 1))
TN = np.sum((predicted_labels == 0) & (Y2_train == 0))
FP = np.sum((predicted_labels == 0) & (Y2_train == 1))
FN = np.sum((predicted_labels == 1) & (Y2_train == 0))

# Compute accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)
```

```
# Compute confusion matrix
confusion_matrix = np.array([[TN, FP], [FN, TP]])

# Compute precision, recall, and F1 score
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * (precision * recall) / (precision + recall)

## Print the above computed metrics
accuracy_train = accuracy
f1_train = f1
cf_train = confusion_matrix # confusion matrix


print("Logistic Regression")
print("Training Accuracy: ", accuracy_train)
print("Training F1 score: ", f1_train)
print("Training confusion matrix: ")
print(cf_train)
```

```
Logistic Regression
Training Accuracy:  0.7285714285714285
Training F1 score:  0.6545454545454547
Training confusion matrix:
[[33 19]
 [ 0 18]]
```

```
## Compute f1 score, accuracy and confusion matrix for testing

Y2_test_pred = np.dot(X2_test_Bias, Beta)
#print(Y2_train_pred.shape)
#print("Y2_train_pred:",Y2_train_pred[0:20])

# Set a threshold (usually 0.5 for binary classification)
threshold = 0.5

# Convert probabilities to binary class labels
predicted_labels = (Y2_test_pred >= threshold).astype(int)
#print(predicted_labels[0:20])

# Compute true positives (TP), true negatives (TN), false positives (FP), and false negati
# For Training data
TP = np.sum((predicted_labels == 1) & (Y2_test == 1))
TN = np.sum((predicted_labels == 0) & (Y2_test == 0))
FP = np.sum((predicted_labels == 0) & (Y2_test == 1))
FN = np.sum((predicted_labels == 1) & (Y2_test == 0))

# Compute accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)
```

```
# Compute confusion matrix
confusion_matrix = np.array([[TN, FP], [FN, TP]])

# Compute precision, recall, and F1 score
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * (precision * recall) / (precision + recall)

## Print the above computed metrics
accuracy_test = accuracy
f1_test = f1
cf_test = confusion_matrix # confusion matrix

print("Logistic Regression")
print("Testing  Accuracy: ", accuracy_test)
print("Testing  F1 score: ", f1_test)
print("Testing  confusion matrix: ")
print(cf_test)
```

```
Logistic Regression
Testing  Accuracy:  0.7666666666666667
Testing  F1 score:  0.7200000000000001
Testing  confusion matrix:
[[14  6]
 [ 1  9]]
```