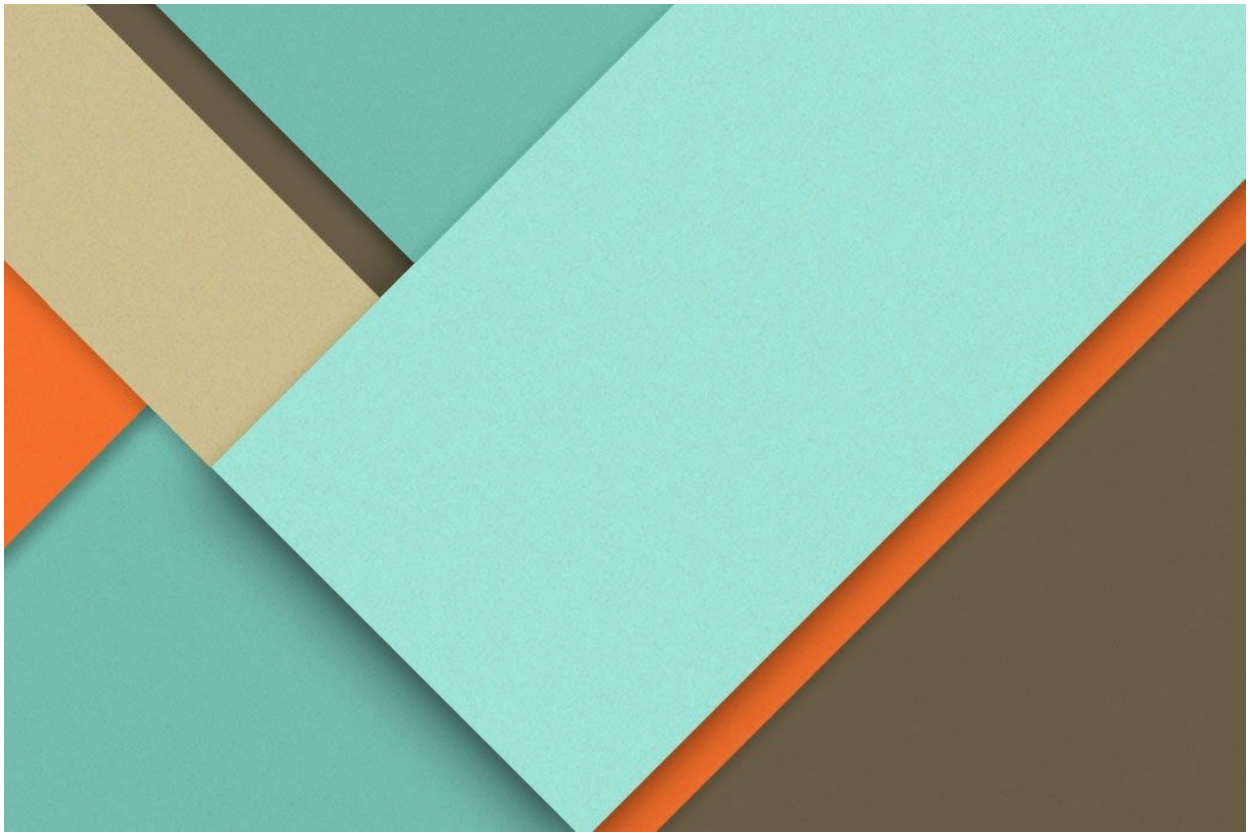


PRML Quiz 1

08.10.2020

Vipul Yuvraj Nikam
180010041
3rd year, CSE

Question 1:

Observe the data by plotting target variable w.r.t each feature and based on your observation comment whether you will use linear regression or polynomial regression. If you choose polynomial regression then what is it's degree? Now, validate your observation computationally by performing the task given below.

Perform univariate linear regression task w.r.t each feature and tabulate the following:

1. Estimated parameters and errors for each fold.
2. Comment on the variation obtained in the values of estimated parameters and error in each fold.

Similarly, perform polynomial regression and tabulate the results. Compare linear and polynomial regression and comment on the order of polynomial regression which perfectly fits the data. Plot the best fit line w.r.t each feature.

Answers:

I. Code:

```
# -*- coding: utf-8 -*-
"""Prml_Q1.ipynb
Automatically generated by Colaboratory.
Original file is located at
https://colab.research.google.com/drive/1rMyiDlB2VyqBPdbjkxYnPy4TJoYVe4p9
"""
# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import panda as pd
import matplotlib.pyplot as plt
import csv
```

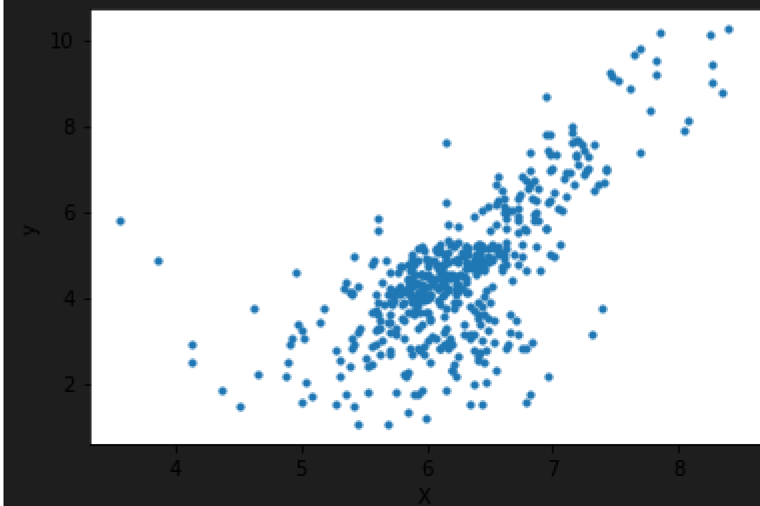
```
from mpl_toolkits.mplot3d import axes3d
# %matplotlib notebook

from google.colab import files
uploaded = files.upload()

data = np.genfromtxt('quiz1_dataset.csv', delimiter=',')
data.shape

y = data[:, -1:]
X = data[:, :-1]
X.shape, y.shape

class regression:
 # Constructor
   def __init__(self, name='reg'):
       self.name = name  # Create an instance variable

   def grad_update(self,w_old,lr,y,x):
       w = w_old - ((2*lr*(np.dot(x.T, (np.dot(x, w_old)-y)))/x.shape[0]))
       return w

   def error(self,w,y,x):
       return np.sum((y- np.dot(x, w))**2)/x.shape[0]

   def mat_inv(self,y,x_aug):
       return np.linalg.inv((x_aug.T).dot(x_aug)).dot(x_aug.T).dot(y)
```

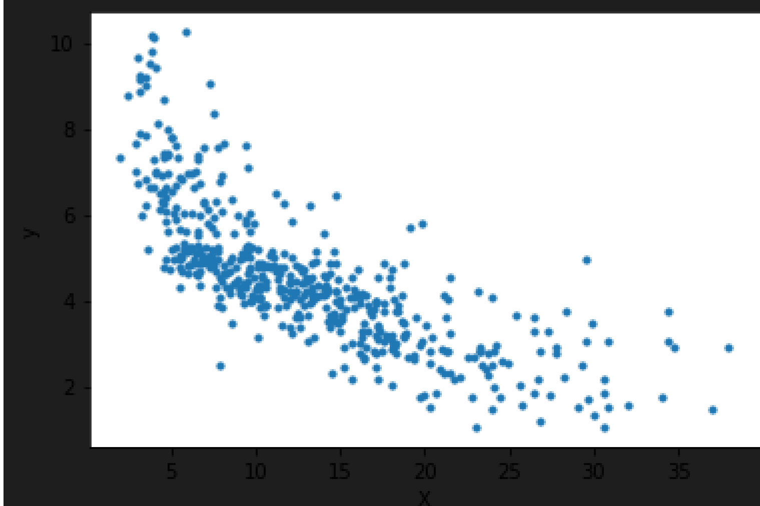
```
def Regression_grad_des(self,x,y,lr):
    w_pred = np.zeros((x.shape[1], 1))
    arry = []
    err = []
    for i in range(1000):
        w_pred = self.grad_update(w_pred, lr, y, x)
        err.append(self.error(w_pred, y, x))
         if(i == 999):
               return w_pred,err

def single_featured_plot(self, x, y):
    plt.figure()
    plt.scatter(x, y, s=10)
    plt.xlabel('X')
    plt.ylabel('y')  # Axis labeling
    plt.show()

def two_featured_plot(self, x, y):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.scatter(x[:, 0], x[:, 1], y)
    plt.ylabel('y1')    # Axis labeling
    plt.xlabel('x1')
    plt.show()

def feature_scale(self, x):
```

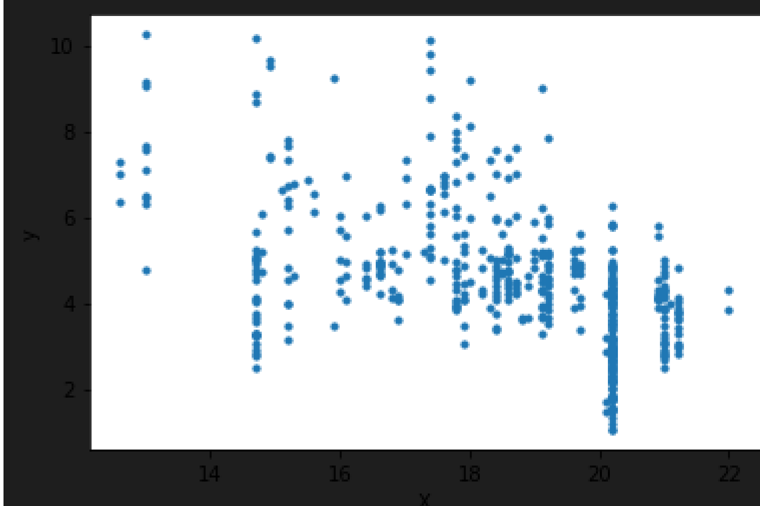
```
        x_mean = x.mean(axis=0)

        x_stdev = x.std(axis=0)

        x_normal = (x - x_mean)/(x_stdev)

        return x_normal, x_mean, x_stdev


    def featurewise_regression(self, x, y, x_test, y_test,
featurewise_folding_dict, fold_number, learning_rate= 0.01):

        for i in range(x.shape[1]):

            x_normal, x_mean, x_stdev = self.feature_scale(x[:, i])

            X_aug = np.c_[np.ones((x.shape[0], 1)), x_normal]

            featurewise_folding_dict['w_pred_mat_inv'][i+1][fold_number] =
self.mat_inv(y, X_aug)

            featurewise_folding_dict['w_pred_grad_desc'][i+1][fold_number],
error = self.Regression_grad_des(X_aug, y, learning_rate)

            featurewise_folding_dict['Training_error'][i+1][fold_number] =
error[-1]


            x_test_normal = (x_test[:, i] - x_mean)/(x_stdev)

            x_test_aug = np.c_[np.ones((x_test.shape[0], 1)),
x_test_normal]


            featurewise_folding_dict['Testing_error'][i+1][fold_number] =
self.error(featurewise_folding_dict['w_pred_grad_desc'][i+1][fold_number],
y_test, x_test_aug)


    def data_transform(self, X,degree):

        X_new = np.ones((X.shape[0], 1))

        for i in range(1, degree+1):

            lup_var = i
```

```
            X_new = np.c_[X_new, X**lup_var]
        return X_new.T


    def feature_poly_regression(self, x, y, x_test, y_test,
featurewise_folding_dict, fold_number, feature, learning_rate= 0.01):
        lr = 0.01
        x_normal, x_mean, x_stdev = self.feature_scale(x)
        X_aug = np.c_[np.ones((x.shape[0], 1)), x_normal]
        featurewise_folding_dict['w_pred_mat_inv'][feature][fold_number] =
self.mat_inv(y, X_aug).round(4)
        featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number],
error = self.Regression_grad_des(X_aug, y, learning_rate)
        featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number]
=
featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number].round(4
)
        featurewise_folding_dict['Training_error'][feature][fold_number] =
error[-1]


        x_test_normal = (x_test - x_mean)/(x_stdev)
        x_test_aug = np.c_[np.ones((x_test.shape[0], 1)), x_test_normal]


        featurewise_folding_dict['Testing_error'][feature][fold_number] =
self.error(featurewise_folding_dict['w_pred_grad_desc'][feature][fold_numb
er], y_test, x_test_aug)


    def poly_regression_bifeatured(self, x, y, x_test, y_test,
featurewise_folding_dict, fold_number, feature, learning_rate= 0.01):
        x_normal, x_mean, x_stdev = self.feature_scale(x)
        X_aug = np.c_[np.ones((x.shape[0], 1)), x_normal]
```

```
       featurewise_folding_dict['w_pred_mat_inv'][feature][fold_number] = self.mat_inv(y, X_aug).round(4)
       featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number], error = self.Regression_grad_des(X_aug, y, learning_rate)
       featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number] = featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number].round(4)
       featurewise_folding_dict['Training_error'][feature][fold_number] = error[-1]


       x_test_normal = (x_test - x_mean)/(x_stdev)
       x_test_aug = np.c_[np.ones((x_test.shape[0], 1)), x_test_normal]


       featurewise_folding_dict['Testing_error'][feature][fold_number] = self.error(featurewise_folding_dict['w_pred_grad_desc'][feature][fold_number], y_test, x_test_aug)


reg = regression()
for i in range(X.shape[1]):
   reg.single_featured_plot(data[:, i], y)
```



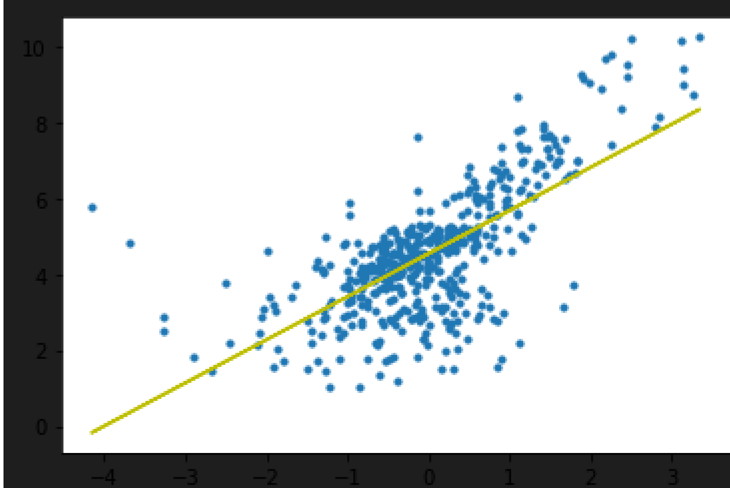




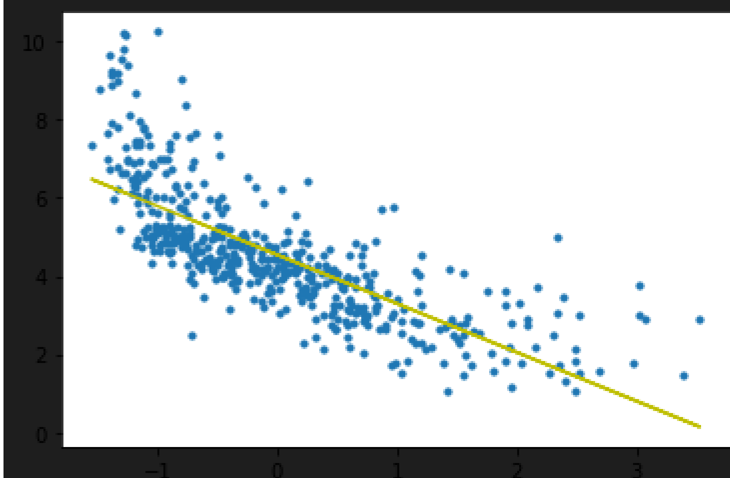

```
test_size = y.shape[0]//5
```

```
featurewise_folding_dict = {}

featurewise_folding_dict['w_pred_mat_inv'] = {}

featurewise_folding_dict['w_pred_grad_desc'] = {}

featurewise_folding_dict['Training_error'] = {}

featurewise_folding_dict['Testing_error'] = {}


for key in featurewise_folding_dict.keys():

   for i in range(X.shape[1]):

       featurewise_folding_dict[key][i+1] = {}

       for j in range(5):

           featurewise_folding_dict[key][i+1][j+1] = {}


for i in featurewise_folding_dict:

   print(f"i")


test_slice, remainder = np.split(X.copy(), [test_size], axis=0)

test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)

reg.featurewise_regression(remainder, remainder_y, test_slice,
test_slice_y, featurewise_folding_dict, 1, 0.01)



for i in range(0, 3):

   rmbr = 2

   lr = 0.01

   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy() #in for loop statement
```

```
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy() #in for loop statement
   reg.featurewise_regression(remainder, remainder_y, test_slice,
test_slice_y, featurewise_folding_dict, i+rmbr, lr)


reg.featurewise_regression(X[:4*test_size], y[:4*test_size],
X[4*test_size:], y[4*test_size:], featurewise_folding_dict, 5, lr)


w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])
w_pred_mat_inv.head()


w_pred_mat_inv[1].mean()


X_normal, X_mean, X_stdev = reg.feature_scale(X)
X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal[:, 0]]
plt.figure()
plt.scatter(X_normal[:, 0], y, s=10)
plt.plot(X_normal[:, 0], np.dot(X_aug, (w_pred_mat_inv[1].mean())), 'y-')
plt.show()  #plot show
```



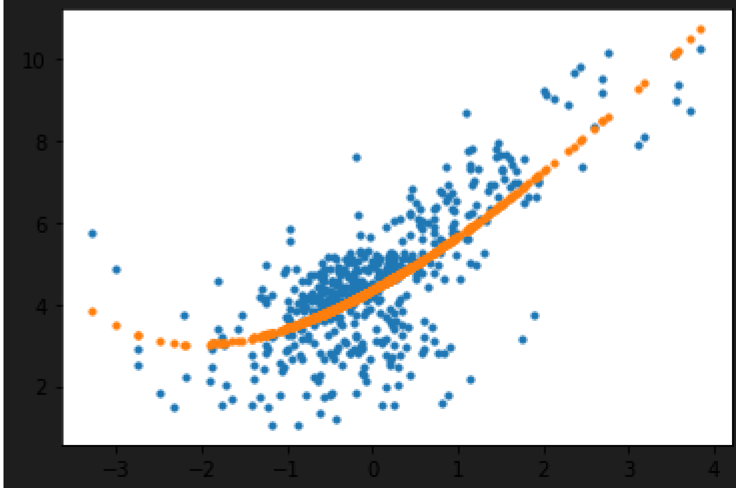
```
X_normal, X_mean, X_stdev = reg.feature_scale(X)
X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal[:, 1]]
plt.figure()
plt.scatter(X_normal[:, 1], y, s=10)
plt.plot(X_normal[:, 1], np.dot(X_aug, (w_pred_mat_inv[2].mean())), 'y-')
plt.show()
```




```
w_pred_grad_desc =
pd.DataFrame(featurewise_folding_dict['w_pred_grad_desc'])
w_pred_grad_desc.head()


Training_error = pd.DataFrame(featurewise_folding_dict['Training_error'])
Training_error.head()


reg = regression()


featurewise_folding_dict = {}
featurewise_folding_dict['w_pred_mat_inv'] = {}
featurewise_folding_dict['w_pred_grad_desc'] = {}
```

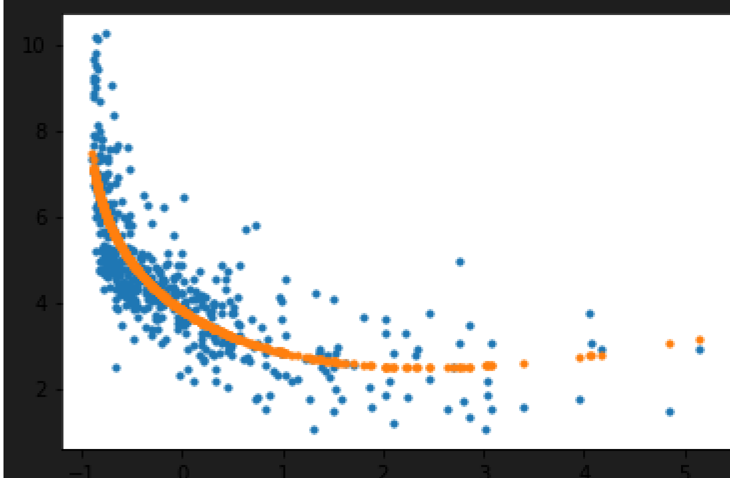
```
#Error calculations.
featurewise_folding_dict['Training_error'] = {}
featurewise_folding_dict['Testing_error'] = {}

for key in featurewise_folding_dict.keys():
   for i in range(X.shape[1]):
       Fear = i
       featurewise_folding_dict[key][Fear+1] = {}
       for j in range(5):
           featurewise_folding_dict[key][Fear+1][j+1] = {}

for i in featurewise_folding_dict:
   print(i)

X_transform = (reg.data_transform(X[:, 0], 2)).T
X_transform.shape

test_slice, remainder = np.split(X_transform.copy(), [test_size], axis=0)
test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)

reg.feature_poly_regression(remainder[:, 1:], remainder_y, test_slice[:,
1:], test_slice_y, featurewise_folding_dict, 1, 1, 0.01)

for i in range(0, 3):
   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy()
```

```
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy()
   reg.feature_poly_regression(remainder[:, 1:], remainder_y,
test_slice[:, 1:], test_slice_y, featurewise_folding_dict, i+2, 1, 0.01)


lr = 0.01
reg.feature_poly_regression(X_transform[:4*test_size, 1:],
y[:4*test_size], X_transform[4*test_size:, 1:], y[4*test_size:],
featurewise_folding_dict, 5, 1, lr)


w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])
w_pred_mat_inv.head()


w_pred_mat_inv[1].mean()




X_normal, X_mean, X_stdev = reg.feature_scale(X_transform[:, 1:])
X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]
plt.figure()


plt.scatter(X_normal[:, 1], y, s=10)
plt.scatter(X_normal[:, 1], np.dot(X_aug, (w_pred_mat_inv[1].mean())),
s=10)
plt.show()
```




```
X_transform = (reg.data_transform(X[:, 1], 2)).T
X_transform.shape

test_slice, remainder = np.split(X_transform.copy(), [test_size], axis=0)
test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)

reg.feature_poly_regression(remainder[:, 1:], remainder_y, test_slice[:,
1:], test_slice_y, featurewise_folding_dict, 1, 2, 0.01)

for i in range(0, 3):
   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy()
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy()
   reg.feature_poly_regression(remainder[:, 1:], remainder_y,
test_slice[:, 1:], test_slice_y, featurewise_folding_dict, i+2, 2, 0.01)
```

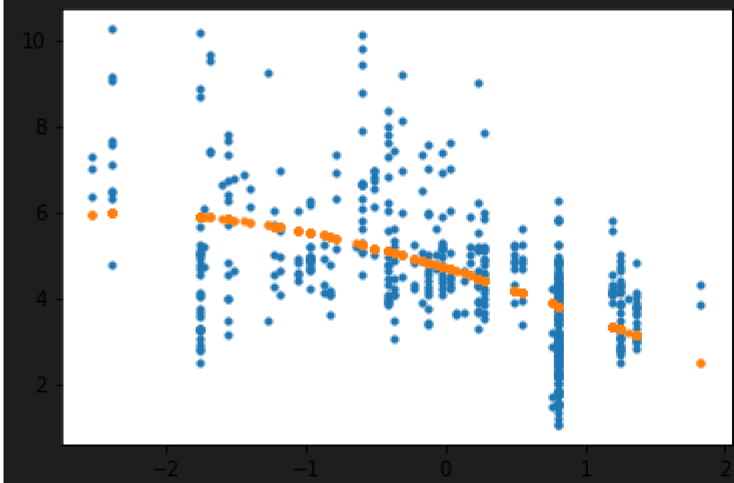
```
reg.feature_poly_regression(X_transform[:4*test_size, 1:],
y[:4*test_size], X_transform[4*test_size:, 1:], y[4*test_size:],
featurewise_folding_dict, 5, 2, 0.01)


w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])

w_pred_mat_inv.head()


w_pred_mat_inv[2].mean()


X_normal, X_mean, X_stdev = reg.feature_scale(X_transform[:, 1:])

X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]

plt.figure()

plt.scatter(X_normal[:, 1], y, s=10)

plt.scatter(X_normal[:, 1], np.dot(X_aug, (w_pred_mat_inv[2].mean())),
s=10)

plt.show()
```




```
X_transform = (reg.data_transform(X[:, 2], 2)).T

X_transform.shape
```

```
test_slice, remainder = np.split(X_transform.copy(), [test_size], axis=0)
test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)

reg.feature_poly_regression(remainder[:, 1:], remainder_y, test_slice[:, 1:], test_slice_y, featurewise_folding_dict, 1, 3, 0.01)

for i in range(0, 3):
   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice, remainder[i*test_size:(i+1)*test_size].copy()
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y, remainder_y[i*test_size:(i+1)*test_size].copy()
   reg.feature_poly_regression(remainder[:, 1:], remainder_y, test_slice[:, 1:], test_slice_y, featurewise_folding_dict, i+2, 3, 0.01)

reg.feature_poly_regression(X_transform[:4*test_size, 1:], y[:4*test_size], X_transform[4*test_size:, 1:], y[4*test_size:], featurewise_folding_dict, 5, 3, 0.01)

w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])
w_pred_mat_inv.head()

w_pred_mat_inv[3].mean()

X_normal, X_mean, X_stdev = reg.feature_scale(X_transform[:, 1:])
X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]
plt.figure()
plt.scatter(X_normal[:, 1], y, s=10)
plt.scatter(X_normal[:, 1], np.dot(X_aug, (w_pred_mat_inv[3].mean())), s=10)
```

```
plt.show()
```



II. Graphs & Plots:

All graphs are attached above in the code file.

III. Explanation:

* Polynomial regression & linear ~~preg~~ regression

→ from the figures & calculations, we can say that polynomial regression of degree 2 better for the data for almost all features.

polynomial regression may take more computation time, (trai it has one extra feature x^2) as compared to linear regression

θ_0 is the most varying in case of polynomial regression (±0.5) b/w the folds whereas, cost varies in terms of (±0.4)

so, we can conclude from this that we use polynomial regression ~~for~~ with degree '2' for features 1&2 & we use liniar regression for features 3 which gives less error than polynomial degree 2

Question 2:

Observe the data by plotting the target variable by taking 2 features at a time and based on your observation comment whether you will use linear regression or polynomial regression. If you choose polynomial regression then what is it's degree? Now, validate your observation computationally by performing the task given below.

Perform bivariate regression task by taking 2 features at a time and tabulate the following:

1. Estimated parameters and errors for each fold.
2. Comment on the variation obtained in the values of estimated parameters and error in each fold.

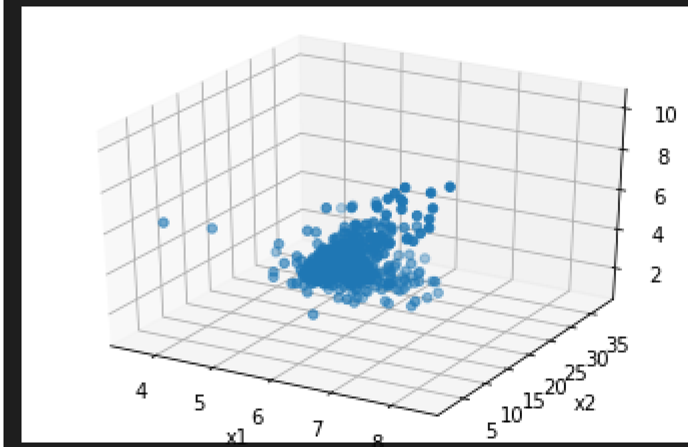
Similarly, perform polynomial bivariate regression and tabulate the results. Compare linear and polynomial regression and comment on the order of polynomial regression which perfectly fit the data.

Plot the best surface w.r.t each feature vector.

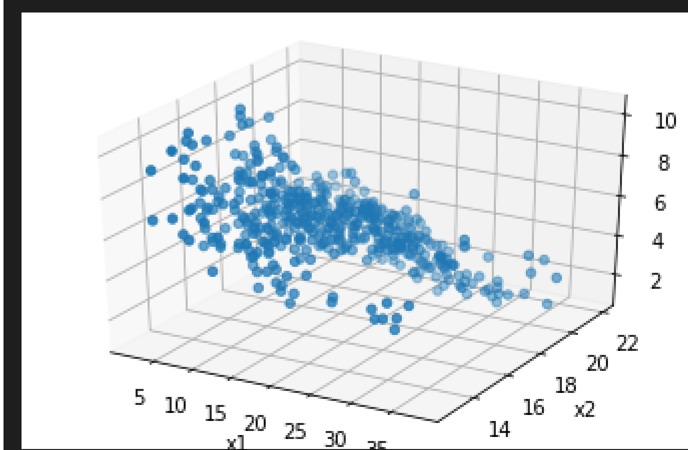
Answers:

I. Code:

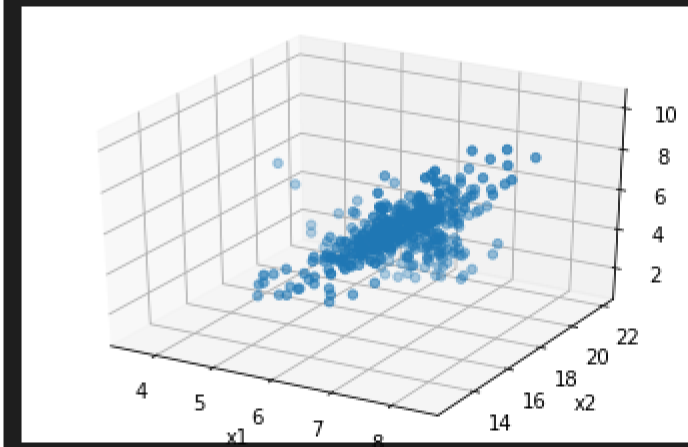
```
#the above code is continued here.

reg = regression()

reg.two_featured_plot(data[:, 0:2], y)
```




```
reg.two_featured_plot(data[:, 1:3], y)
```




```
reg.two_featured_plot(np.c_[data[:, 0], data[:,2]], y)
```

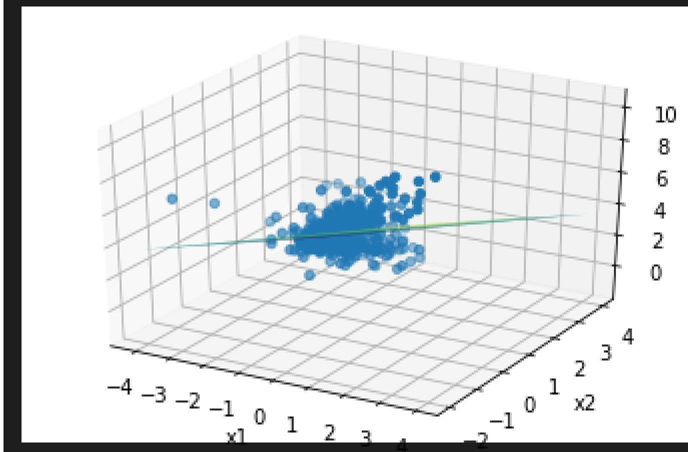



```
featurewise_folding_dict = {}

featurewise_folding_dict['w_pred_mat_inv'] = {}

featurewise_folding_dict['w_pred_grad_desc'] = {}

#errors calculations

featurewise_folding_dict['Training_error'] = {}

featurewise_folding_dict['Testing_error'] = {}


for key in featurewise_folding_dict.keys():

   for i in range(X.shape[1]):

       featurewise_folding_dict[key][i+1] = {}

       for j in range(5):

           str = j

           featurewise_folding_dict[key][i+1][str+1] = {}


for i in featurewise_folding_dict:

   print(i)


test_slice, remainder = np.split(X[:, 0:2].copy(), [test_size], axis=0)

test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)
```

```
reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, 1, 1, 0.01)

for i in range(0, 3):
   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy()
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy()
   reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, i+2, 1, 0.01)

reg.poly_regression_bifeatured(X[:4*test_size, 0:2], y[:4*test_size],
X[4*test_size:, 0:2], y[4*test_size:], featurewise_folding_dict, 5, 1,
0.01)

w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])
w_pred_mat_inv.head()

test_slice, remainder = np.split(X[:, 1:3].copy(), [test_size], axis=0)
test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)

reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, 1, 2, 0.01)

for i in range(0, 3):
   remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy()
   remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy()
```

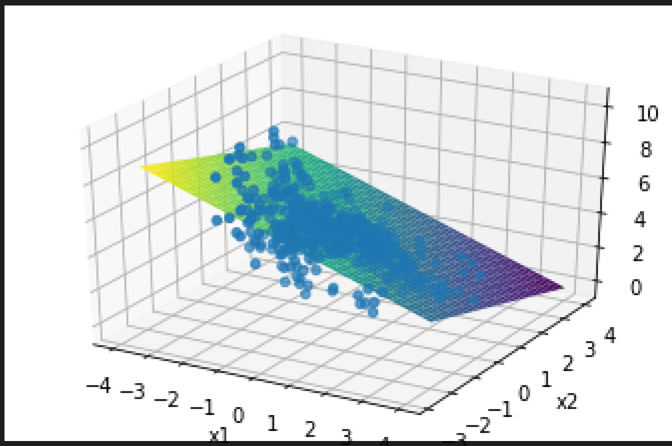
```
    reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, i+2, 2, 0.01)


reg.poly_regression_bifeatured(X[:4*test_size, 0:2], y[:4*test_size],
X[4*test_size:, 0:2], y[4*test_size:], featurewise_folding_dict, 5, 2,
0.01)


w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])

w_pred_mat_inv.head()


test_slice, remainder = np.split(np.c_[X[:, 0], X[:, 2]].copy(),
[test_size], axis=0)

test_slice_y, remainder_y = np.split(y.copy(), [test_size], axis=0)


reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, 1, 3, 0.01)


for i in range(0, 3):

    remainder[i*test_size:(i+1)*test_size], test_slice = test_slice,
remainder[i*test_size:(i+1)*test_size].copy()

    remainder_y[i*test_size:(i+1)*test_size], test_slice_y = test_slice_y,
remainder_y[i*test_size:(i+1)*test_size].copy()

    reg.poly_regression_bifeatured(remainder[:, 0:2], remainder_y,
test_slice[:, 0:2], test_slice_y, featurewise_folding_dict, i+2, 3, 0.01)


reg.poly_regression_bifeatured(X[:4*test_size, 0:2], y[:4*test_size],
X[4*test_size:, 0:2], y[4*test_size:], featurewise_folding_dict, 5, 3,
0.01)


w_pred_mat_inv = pd.DataFrame(featurewise_folding_dict['w_pred_mat_inv'])
```

```
w_pred_mat_inv.head()


print(X.shape)


X_normal, X_mean, X_stdev = reg.feature_scale(X[:, 0:2])

X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]


m = 30

mar1 = 30

x_1 = np.linspace(-4,4,mar1)

y_1 = np.linspace(-2,4,mar1)


X_1,Y_1 = np.meshgrid(x_1,y_1)

Z= w_pred_mat_inv[1].mean()[1, 0]*X_1 + w_pred_mat_inv[1].mean()[2,0]*Y_1
+ w_pred_mat_inv[1].mean()[0,0]


fig = plt.figure()

ax = fig.gca(projection='3d')

ax.scatter(X_normal[:, 0], X_normal[:, 1], y)

surf = ax.plot_surface(X_1, Y_1, Z, cmap='viridis')

# ax.plot_surface(X_normal[:, 0], X_normal[:, 1],  np.dot(X_aug,
(w_pred_mat_inv[1].mean())), color='y')

plt.xlabel('x1')

plt.ylabel('y1')

plt.show()
```



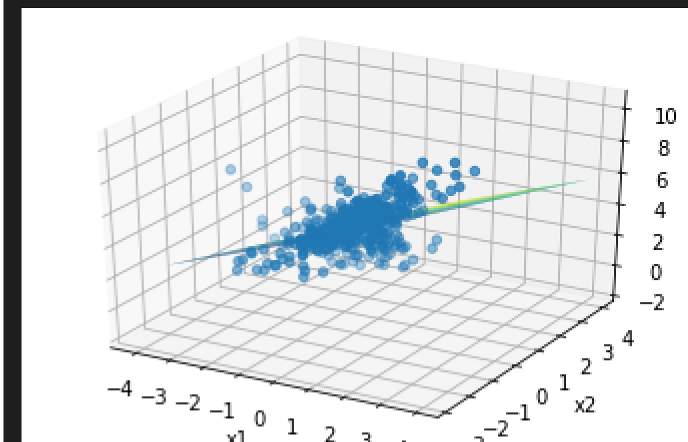

```
X_normal, X_mean, X_stdev = reg.feature_scale(X[:, 1:3])

X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]


m = 30

mar2 = 30

x_1 = np.linspace(-4,4,mar2)

y_1 = np.linspace(-2,4,mar2)


X_1,Y_1 = np.meshgrid(x_1,y_1)

Z= w_pred_mat_inv[2].mean()[1, 0]*X_1 + w_pred_mat_inv[2].mean()[2,0]*Y_1
+ w_pred_mat_inv[2].mean()[0,0]


fig = plt.figure()

ax = fig.gca(projection='3d')

ax.scatter(X_normal[:, 0], X_normal[:, 1], y)

surf = ax.plot_surface(X_1, Y_1, Z, cmap='viridis')

# ax.plot_surface(X_normal[:, 0], X_normal[:, 1],  np.dot(X_aug,
(w_pred_mat_inv[1].mean())), color='y')

plt.xlabel('x1')

plt.ylabel('x2')
```

```
plt.show()
```




```
X_normal, X_mean, X_stdev = reg.feature_scale(np.c_[X[:, 0], X[:, 2]])

X_aug = np.c_[np.ones((X.shape[0], 1)), X_normal]


m = 30

strl = 0

x_1 = np.linspace(-4,4,mar2)

y_1 = np.linspace(-2,4,mar2)


X_1,Y_1 = np.meshgrid(x_1,y_1)

Z= w_pred_mat_inv[3].mean()[1, 0]*X_1 + w_pred_mat_inv[3].mean()[2,0]*Y_1
+ w_pred_mat_inv[3].mean()[strl,strl]


fig = plt.figure()

ax = fig.gca(projection='3d')

ax.scatter(X_normal[:, 0], X_normal[:, 1], y)

surf = ax.plot_surface(X_1, Y_1, Z, cmap='viridis')

plt.xlabel('x1')

plt.ylabel('x2')
```

```
plt.show()   #plot will be shown here
```




```
X.shape, y.shape
```

II. Graphs & Plots:

All graphs are attached above in the code file.

III. Explanation:

→ As values are not compared to linear model so linear model is not initiable for feature.

error values are for less than linear models polynomial regression model in the choice.

in this too, error values are for less compared to linear models fold 2 has ~~been~~ ~~least~~ least error. so it was right choice.

Question 3:

Perform regression analysis considering all the 3 features and tabulate the following:

1. Estimated parameters and errors for each fold.
2. Comment on the variation obtained in the values of estimated parameters and error in each fold.

Answers:

I. Code:

```
#This is a separate file from the first two so run this separately.

import random

import csv

import numpy as np

import matplotlib.pyplot as plt

from Regression import Regression


def errorUsingW(w, x, y):

   predictions = x @ w     #matrix multiplication

   return (np.sum(np.square(y - predictions))) / (2 * y.shape[0])

def addOnesColumn(x):

   x_var1 = 10

   const_ar = np.array([[1] for i in range(x.shape[0])])

   x = np.append(const_ar, x, axis=1)

   return x

def performMatrixInversion(x, y):

   x = addOnesColumn(x)
```

```
    w = ((np.linalg.inv(x.T @ x)) @ x.T @ y)
    newY = x @ w      #matrix
    return w, newY, errorUsingW(w, x, y)
def multivar_reg(x, y, KFOLD):
    for fold_No in range(KFOLD):
        print("K_no :: ", fold_No)
        blockSize = int(((x.shape[0] + KFOLD - 1) / KFOLD))

        blockStartPoint = fold_No * blockSize
        blockEndPoint = blockStartPoint + blockSize

        trainingX = np.concatenate((x[:blockStartPoint, :],
x[blockEndPoint:, :]))
        trainingY = np.concatenate((y[:blockStartPoint, :],
y[blockEndPoint:, :]))

        testX = x[blockStartPoint:blockEndPoint, :]
        testY = y[blockStartPoint:blockEndPoint, :]

        #transform testX (add column of ones)
        testX = addOnesColumn(testX)
        print("Using Matrix Inverse Method :")
        W, newY, leastError = performMatrixInversion(trainingX, trainingY)
        testError = errorUsingW(W, testX, testY)
        print("W : ", W.T, "  error : ", testError)
        print("\n")
```

```
with open('quiz1_dataset.csv') as file:

   data = list(csv.reader(file, delimiter=','))

data = np.array(data)

data = data.astype(np.float)


KFOLD = 5


x  = data[:, 0:3]

y  = data[:, 3:4]


multivar_reg(x,y,5)
```

II. Graphs & Plots:

There are no graphs for this question.

III. Explanation:

→ Fold 4 has least errors from table so $w = (4.22, -0.32, 0.272, 0.57)$ is to be contidered chech polynomial regression to see which is best.

fold 5 has least errors compared to rest of dataset & even linear model so polynomial regression is best when 3 feautures are considered.