**Report 1**

**Problem 1:**

We want to solve the regression problem in Python using the “Auto” dataset.

We  will  focus  on  the  continuous  predictors  “mpg,  cylinders,  displacement,

horsepower, weight, acceleration, year, origin, name”.  and also remove the

observations with missing values.  This can be done in Python as:

Auto = pd.read\_csv(’Auto.csv’, header=0, na\_values=’?’).dropna()

df = pd.DataFrame(Auto)

1.  Produce a scatterplot matrix.

To produce the scatterplot matrix I first used Python to read the data file and then put that information into a data frame. I then used the python command to produce the scatterplot matrix as shown below:

import numpy as np

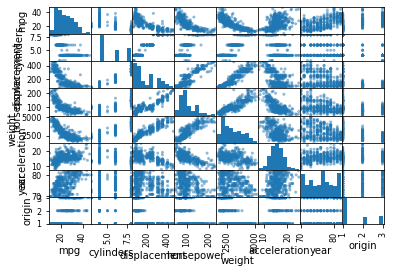
import pandas as pd

import sklearn as sk

Auto = pd.read\_csv('Auto.csv', header=0, na\_values='?').dropna()

df = pd.DataFrame(Auto)

pd.plotting.scatter\_matrix(df)



2. Estimate the regression coefficients B^

To calculate the regression coefficients manually I first subset the data frame by each column and placed them into corresponding data frames with the target column ‘mpg’. I then calculated the numerator of the beta equation and stored it into variables called betax\_num (x representing the number of beta). I then did the similar action for the denominators. Once having the numerator and denominator of the equation I simply divided the values resulting in the beta value. See code below:

xbar1 = np.mean(df["cylinders"])

#print(xbar1)

ybar = np.mean(df["mpg"])

#print(ybar)

cylinders = df["cylinders"]

one\_vec = np.ones(392)

cyl\_bar = xbar1 \* one\_vec

mpg = df["mpg"]

ybar1 = ybar \* one\_vec

test1 =(cylinders-cyl\_bar)

test2 =(mpg-ybar1)

beta1\_num = sum(test1\*test2)

#print(beta1\_num)

test3 = (np.square(cylinders-cyl\_bar))

#print(test3)

beta1\_den = sum(test3)

#print(beta1\_den)

beta1= beta1\_num/beta1\_den

print(beta1)

displacement = df["displacement"]

displacement\_bar = np.mean(df["displacement"])\*one\_vec

beta2\_num = sum((displacement - displacement\_bar)\*(mpg - ybar1))

beta2\_den =sum(np.square(displacement-displacement\_bar))

beta2 = beta2\_num/beta2\_den

print(beta2)

horsepower = df["horsepower"]

horsepower\_bar = np.mean(df["horsepower"])\*one\_vec

beta3\_num = sum((horsepower - horsepower\_bar)\*(mpg - ybar1))

beta3\_den = sum(np.square(horsepower-horsepower\_bar))

beta3 = beta3\_num/beta3\_den

print(beta3)

weight = df["weight"]

weight\_bar = np.mean(df["weight"])\*one\_vec

beta4\_num = sum((weight - weight\_bar)\*(mpg - ybar1))

beta4\_den = sum(np.square(weight - weight\_bar))

beta4 = beta4\_num/beta4\_den

print(beta4)

acceleration = df["acceleration"]

acceleration\_bar = np.mean(acceleration) \* one\_vec

beta5\_num = sum((acceleration - acceleration\_bar) \* (mpg - ybar1))

beta5\_den= sum(np.square(acceleration - acceleration\_bar))

beta5 = beta5\_num/beta5\_den

print(beta5)

year = df["year"]

year\_bar = np.mean(df["year"]) \* one\_vec

beta6\_num = sum((year - year\_bar) \* (mpg - ybar1))

beta6\_den = sum(np.square(year - year\_bar))

beta6 = beta6\_num/beta6\_den

print(beta6)

origin = df["origin"]

origin\_bar = np.mean(origin) \* one\_vec

beta7\_num = sum((origin - origin\_bar) \*(mpg - ybar1))

beta7\_den = sum(np.square(origin - origin\_bar))

beta7 = beta7\_num/beta7\_den

print(beta7)

betas = np.array(beta1, beta2, beta3, beta4, beta5, beta6, beta7)

print(betas)

Results:

-3.5580783676215013

-0.06005142781220627

-0.15784473335365318

-0.007647342535779581

1.197624187732056

1.2300354634480293

5.476547480191458

3. Compute the fitted values and residuals. Then make a plot of theresiduals versus the fitted values.

To compute the fitted values I first computed the H matrix manually and then performed matrix multiplication between that and the real y values. See code below:

df1 = (df[['cylinders','displacement','horsepower','weight','acceleration','year','origin']])

X = np.matrix(df1)

Xt = np.matrix.transpose(X)

XtX = np.matmul(Xt,X)

XtX\_inv= np.linalg.inv(XtX)

XXtX\_inv = np.matmul(X,XtX\_inv)

H = np.matmul(XXtX\_inv,Xt)

y\_vec = (df['mpg'])

yhat = np.matmul(H,y\_vec)

print(yhat)

res = y\_vec - yhat

print(res)

import matplotlib.pyplot as plt

from statsmodels.nonparametric.smoothers\_lowess import lowess as sm\_lowess

top3 = abs(res).sort\_values(ascending = False)[:3]

smoothed = sm\_lowess(res,yhat)

plt.rcParams.update({'font.size': 16})

plt.rcParams["figure.figsize"] = (8,7)

fig, ax = plt.subplots()

ax.scatter(yhat, res, edgecolors = 'k', facecolors = 'none')

ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')

ax.set\_ylabel('Residuals')

ax.set\_xlabel('Fitted Values')

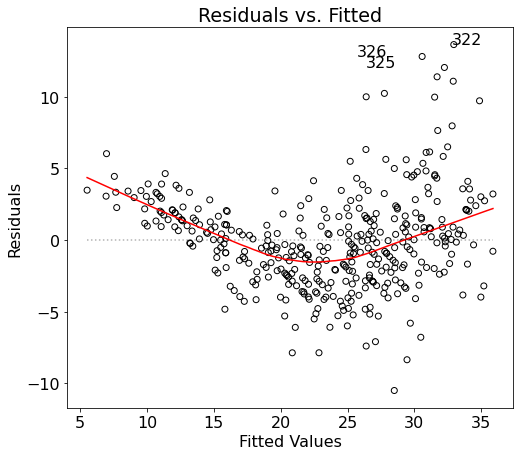
ax.set\_title('Residuals vs. Fitted')

ax.plot([min(yhat),max(yhat)],[0,0],color = 'k',linestyle = ':', alpha = .3)

for i in top3.index:

ax.annotate(i,xy=(yhat[i],res[i]))

plt.show()



4. Perform a test to check if the predictor ‘cylinders’ is needed in themodel. This is done by testingH0:βcyl= 0 vsHa:βcyl6= 0.

By looking at my previously included values for ‘betas’ you can see that the Beta value for ‘cylinders’ does not equal 0. Therefore, the null hypothesis is rejected.

print(betas)

Results:

-3.5580783676215013

-0.06005142781220627

-0.15784473335365318

-0.007647342535779581

1.197624187732056

1.2300354634480293

5.476547480191458

5. Evaluate the performance of your model by finding the mean square error (MSE) and root mean square error (RMSE) of your model.

To calculate MSE and RMSE I first took the sum of squares of the values in mpg minus the values in yhat. Then dividing that result by the number of observations (392) I obtained the MSE. To obtain the RMSE I took the square root of the MSE. See code below:

mse = sum(np.square(mpg - yhat))/392

print(mse)

rmse = np.sqrt(mse)

print(rmse)

Results:

11.235761433092202 (mse)

3.351978733985674 (rmse)

6. Compare your results with Python commands from ‘smf.OLS()’.

Comparing my results to the python commands, there seems to be some variation between my regression coefficients. However, they tend to be somewhat consistent with positive and negative values. Doing all of these calculations leaves much room for error so it is important to be as meticulous as possible:

OLS Regression Results

==============================================================================

Dep. Variable: mpg R-squared: 0.821

Model: OLS Adj. R-squared: 0.818

Method: Least Squares F-statistic: 252.4

Date: Mon, 31 May 2021 Prob (F-statistic): 2.04e-139

Time: 16:29:25 Log-Likelihood: -1023.5

No. Observations: 392 AIC: 2063.

Df Residuals: 384 BIC: 2095.

Df Model: 7

Covariance Type: nonrobust

================================================================================

coef std err t P>|t| [0.025 0.975]

--------------------------------------------------------------------------------

Intercept -17.2184 4.644 -3.707 0.000 -26.350 -8.087

cylinders -0.4934 0.323 -1.526 0.128 -1.129 0.142

displacement 0.0199 0.008 2.647 0.008 0.005 0.035

horsepower -0.0170 0.014 -1.230 0.220 -0.044 0.010

weight -0.0065 0.001 -9.929 0.000 -0.008 -0.005

acceleration 0.0806 0.099 0.815 0.415 -0.114 0.275

year 0.7508 0.051 14.729 0.000 0.651 0.851

origin 1.4261 0.278 5.127 0.000 0.879 1.973

==============================================================================

Omnibus: 31.906 Durbin-Watson: 1.309

Prob(Omnibus): 0.000 Jarque-Bera (JB): 53.100

Skew: 0.529 Prob(JB): 2.95e-12

Kurtosis: 4.460 Cond. No. 8.59e+04

**Problem 2:**

We will use the “Breast Cancer Wisconsin (Original) Dataset” and is avail-able athttps://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original). More details about this data can found on pages 218-219 of thetextbook.1

1. Split the data into training and testing sets. Use 80% of the data for training and 20% for testing. You can perform this in Python using the commands on page 220

To split the training and testing sets I had to import the sklearn package in order to have access to the command train\_test\_split. I then loaded in my dataset and placed the data and target into two separate variables x and y. Using those variables I created 4 subsets X\_train, X\_test, y\_train, y\_test.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn import datasets

data\_set = datasets.load\_breast\_cancer()

X=data\_set.data

y=data\_set.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

1. Using the first observation of the test set as our new observation xnew, classify that observation based onδk(x) using LDA.

To manually calculate LDA for the first observation

K1 = pd.DataFrame(X\_test[y\_test == 0])

K2 = pd.DataFrame(X\_test[y\_test == 1])

X\_train\_df = pd.DataFrame(X\_train)

X\_test\_df = pd.DataFrame(X\_test)

y\_train\_df = pd.DataFrame(y\_train)

y\_test\_df = pd.DataFrame(y\_test)

xb1= np.mean(K1)

xb2 = np.mean(K2)

x1t = (K1.T)

x2t = (K2.T)

s1 = np.cov(x1t)

s2 = np.cov(x2t)

p1=p2=0.5

n1=n2=2

n = n1+n2

k=2

sp = (((n1-1)\*s1 +(n2-1)\*s2)/(n-k))

from numpy.linalg import inv

spi = inv(sp)

xnew = X\_test[0, :]

t1 = np.matmul(xb1.T, spi)

cst1 = np.matmul(t1, xb1)

e1=np.matmul(t1, xnew)

det1= e1-(1/2)\*cst1+np.log(p1)

print(det1)

t2 = np.matmul(xb2.T, spi)

cst2 = np.matmul(t2, xb2)

e2=np.matmul(t2, xnew)

det2= e2-(1/2)\*cst2+np.log(p2)

print(det2)

print(np.max([det1,det2]))

Results:

1511.5299340739195

1503.255941190667

1511.5299340739195

#therefore, classify as 0

1. Write a function ‘ldadelta’ to find the discriminant functionˆδk(x) fora new observationx. Check your function with the observation from the previous question

To create this function Idadelta I essentially took my work from the previous question and placed it into a function of itself. The only difference is that I added loop characteristics so that it evaluated each row individually until it reached the value of the row count. See the code below: \

def lda\_delta(xnew, ynew):

x = 0

while x < len(xnew):

x0 = xnew[x, :]

K1 = pd.DataFrame(xnew[ynew == 0])

K2 = pd.DataFrame(xnew[ynew == 1])

xb1= np.mean(K1)

xb2 = np.mean(K2)

x1t = (K1.T)

x2t = (K2.T)

s1 = np.cov(x1t)

s2 = np.cov(x2t)

p1=p2=0.5

n1=n2=2

n = n1+n2

k=2

sp = (((n1-1)\*s1 +(n2-1)\*s2)/(n-k))

from numpy.linalg import inv

spi = inv(sp)

t1 = np.matmul(xb1.T, spi)

cst1 = np.matmul(t1, xb1)

e1=np.matmul(t1, x0)

det0= e1-(1/2)\*cst1+np.log(p1)

# print(det0)

t2 = np.matmul(xb2.T, spi)

cst2 = np.matmul(t2, xb2)

e2=np.matmul(t2, x0)

det1= e2-(1/2)\*cst2+np.log(p2)

#print(det1)

if np.max([det0,det1]) == det0:

print(0)

elif np.max([det0,det1]) == det1:

print(1)

else:

print("error")

x += 1

1. Apply your function ‘ldadelta’ to the training set and produce theconfusion matrix. Use the confusion matrix to find ‘Recall, Specificity, Fallout, Positive predictive value, and the accuracy’ for the training set. Also, evaluate the training error

To create the confusion matrix as well as calculating the corresponding values I had Python write the results as well as the real values to an excel file. I then used excel to classify each row as either a TP, TN, FP, or FN. Once I had the count of each of these I was able to produce the matrix. Once I had the matrix produced I was able to calculate each of the values. See code/excel below:

n = np.matrix(lda\_delta(X\_train, y\_train))

n.to\_excel('./confusion0.xlsx', sheet\_name = 'train\_results', index=False)

y\_train\_mat = pd.DataFrame(y\_train)

y\_train\_mat.to\_excel('./confusion1.xlsx', sheet\_name = 'train', index=False)

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 157 | 2 |
| 1 | 8 | 288 |
|  |  |  |
| P | 165 |  |
| N | 290 |  |
| Recall | 0.987421 |  |
| Specificity | 0.993103 |  |
| Fallout | 0.012121 |  |
| PPV | 0.951515 |  |
| ACC | 0.978022 |  |
| MSE | 0.021978 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Apply your function ‘ldadelta’ to the test set and produce the con-fusion matrix. Use the confusion matrix to find ‘Recall, Specificity,Fallout, Positive predictive value, and the accuracy’ for the test set.Find the test error.

I repeated the same procedure that I used for question 4 see code/excel below:

z = np.matrix(lda\_delta(X\_test, y\_test))

y\_test\_mat = pd.DataFrame(y\_test)

z.to\_excel('./confusion2.xlsx', sheet\_name = 'test\_results')

y\_test\_mat.to\_excel('./confusion3.xlsx', sheet\_name = 'test', index=False)

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 45 | 1 |
| 1 | 2 | 67 |
|  |  |  |
| P | 47 |  |
| N | 67 |  |
| Recall | 0.978261 |  |
| Specificity | 0.985294 |  |
| Fallout | 0.021277 |  |
| PPV | 0.957447 |  |
| ACC | 0.982456 |  |
| MSE | 0.026316 |  |

1. Compare your results with the ones obtained using the commands from Python.

The results produced by the commands from Python are close to that of the manually calculated values. See code and results below:

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn import neighbors

lda = LinearDiscriminantAnalysis()

pred = lda.fit(X\_train, y\_train).predict(X\_test)

lda.priors\_

lda.means\_

lda.coef\_

confusion\_matrix(y\_test, pred).T

print(classification\_report(y\_test, pred, digits=3))

pred\_p = lda.predict\_proba(X\_test)

a1=np.unique(pred\_p[:,1]>0.5, return\_counts=True)

a1

np.unique(pred\_p[:,1]>0.9, return\_counts=True)

Results:

precision recall f1-score support

0 1.000 0.915 0.956 47

1 0.944 1.000 0.971 67

accuracy 0.965 114

macro avg 0.972 0.957 0.963 114

weighted avg 0.967 0.965 0.965 114

1. Provide the ROC curve and AUC for your model