Mid-Term Project report

Title: Create a binary classifier to predict quality test results for each capacitor using Logistic Regression

Introduction:

For this model, I have loaded the train and test datasets for capacitors test results with two features x1, x2. I have written more features in the train and test data files to include eight features (x1,x2,x3…,x8) for better classification. We'll train the model on the training dataset and predict the result class on the test dataset. Initialize the weights, the learning rate and use the sigmoid function to predict the probability of the resulting class value. By the gradient descent algorithm, we tried to minimize the cost function by changing the values of W, so the function converges at the least error value.

Initial Values:

For this task, I have selected initial values of weights as 0, alpha or learning rate as 0.2, and intial J was 0.6931471805599454

Final Values:

The final values of weights were [[-0.73133282]

[0.22370008]

[-0.45965325]

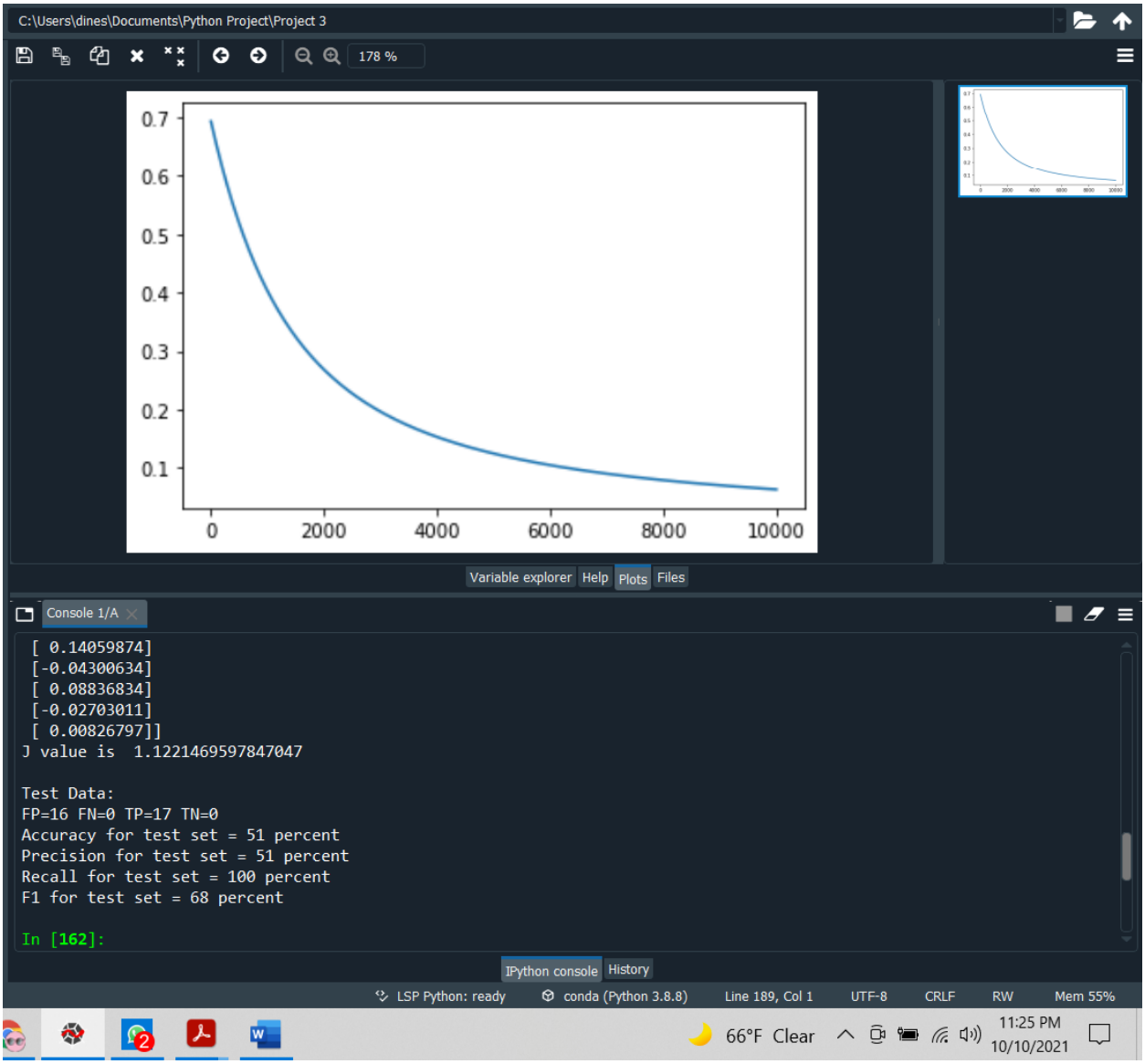
[0.14059874]

[-0.04300634]

[0.08836834]

[-0.02703011]

[0.00826797]], alpha or learning rate was 0.0015, and final J was 0.06982795511319106, which were derived after doing 10,000 iterations.



J vs. Iteration plot

J value for Test Dataset = 1.1221469597847047

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


aFileName = input("Enter the name of your train file: ")
```

```
fin= open(aFileName, "r")
#fin= open('P3train.txt', "r")

aString = fin.readline()
d=aString.split("\t")
rows = int(d[0])
columns = int(d[1])+1

data= np.zeros([rows, columns])

for k in range(rows):
    aString = fin.readline()
    t = aString.split("\t")
    t[-1]=t[-1].strip()

    for j in range(columns):
        data[k,j]= float(t[j])

fin.close()

#writing new features in a train file
fout = open('P3train1.txt', "w")

thePower = 2

for a in range(rows):
    x1=data[a][0]
    x2=data[a][1]
```

```
    y=data[a][2]
    for j in range(thePower+1):
        for i in range(thePower+1):
            temp = (x1**i)*(x2**j)
            if (temp != 1):
                fout.write(str(temp)+"\t")
    fout.write(str(y)+"\n")
fout.close()

fin= open('P3train1.txt', "r")

aString = fin.readline()
d=aString.split("\t")
# print(rows,columns)
# rows = int(d[0])
columns = columns+6
# print(columns)
data1= np.zeros([rows, columns])

for k in range(rows):
    bString = fin.readline()
    t1 = aString.split("\t")
    t1[-1]=t1[-1].strip()

    for j in range(columns):
        data1[k,j]= float(t1[j])

fin.close()
```

```
df = pd.DataFrame(data1, columns =['x1','x2','x3','x4','x5','x6','x7','x8','y'])
# print(df)
TX = df.iloc[:,:8]
TY = df.iloc[:,8:]

X_train = TX.values
Y_train = TY.values
X_train = X_train.T
Y_train = Y_train.reshape(1, X_train.shape[1])

aFileName = input("Enter the name of your test file: ")
fin= open(aFileName, "r")
# fin= open('P3test.txt', "r")

aString = fin.readline()
d=aString.split("\t")
rows = int(d[0])
columns = int(d[1])+1

data= np.zeros([rows, columns])

for k in range(rows):
    aString = fin.readline()
    t = aString.split("\t")
    t[-1]=t[-1].strip()

    for j in range(columns):
```

```
        data[k,j]= float(t[j])

fin.close()
#writing new features in a test file
fout = open('P3test1.txt', "w")

thePower = 2

for a in range(rows):
    x1=data[a][0]
    x2=data[a][1]
    y=data[a][2]
    for j in range(thePower+1):
        for i in range(thePower+1):
            temp = (x1**i)*(x2**j)
            if (temp != 1):
                fout.write(str(temp)+"\t")
    fout.write(str(y)+"\n")
fout.close()

fin= open('P3test1.txt', "r")

aString = fin.readline()
d=aString.split("\t")
# print(rows,columns)
# rows = int(d[0])
columns = columns+6
# print(columns)
```

```
data2= np.zeros([rows, columns])

for k in range(rows-1):
    bString = fin.readline()
    t1 = bString.split("\t")
    t1[-1]=t1[-1].strip()
    # print(t1)
    for j in range(columns):
        data2[k,j]= float(t1[j])

fin.close()

dt = pd.DataFrame(data2, columns =['x1','x2','x3','x4','x5','x6','x7','x8','y'])
# print(dt)
X= dt.iloc[:,:8]
Y= dt.iloc[:,8:]

X_test = X.values
Y_test = Y.values
# print(Y_test)
X_test = X_test.T
Y_test = Y_test.reshape(1, X_test.shape[1])
# print(Y_test)

print("Shape of X_train is ",X_train.shape)
print("Shape of Y_train is ",Y_train.shape)
print("Shape of X_test is ",X_test.shape)
print("Shape of Y_test is ",Y_test.shape)
```

```
def sigmoid(x):
    return 1/(1 + np.exp(-x))
def model(X, Y, learning_rate, iterations):
    m = X_train.shape[1]
    n = X_train.shape[0]

    W = np.zeros((n,1))
    B = 0

    cost_list = []
    for i in range(iterations):

        Z = np.dot(W.T, X) + B
        A = sigmoid(Z)

        cost = -(1/m)*np.sum( Y*np.log(A) + (1-Y)*np.log(1-A))
        dW = (1/m)*np.dot(A-Y, X.T)
        dB = (1/m)*np.sum(A - Y)

        W = W - learning_rate*dW.T
        B = B - learning_rate*dB

        cost_list.append(cost)

        if(i%(iterations/10) == 0):
            print("cost after",i, "iteration is : ",cost)
    return W, B, cost_list
```

```
iteration = 10000
learning_rate =  0.0015
W, B, cost_list = model(X_train, Y_train,learning_rate, iteration)


plt.plot(np.arange(iteration),cost_list)


# print(X_train)
print(W)
Z = np.dot(W.T, X_test) + B
A = sigmoid(Z)
m = X_test.shape[1]
n = X_test.shape[0]
cost = -(1/m)*np.sum( Y_test*np.log(A) + (1-Y_test)*np.log(1-A))
print('J value is ',cost)
A = A > 0.5
A = np.array(A, dtype = 'int64')
A = A.T
# print(A)
# print(Y_test[0])
Y_test=Y_test.T
# print(Y_test[4])
FP=FN=TP=TN=0
for i in range(rows):
    if(Y_test[i]==0 and A[i]==1):
        FP+=1
    if(Y_test[i]==1 and A[i]==0):
        FN+=1
```

```
    if(Y_test[i]==1 and A[i]==1):
        TP+=1
    if(Y_test[i]==0 and A[i]==0):
        TN+=1
print('\nTest Data: \nFP=%d FN=%d TP=%d TN=%d'%(FP, FN, TP, TN))
Model_Acc=((TP+TN)/(TP+TN+FP+FN))*100
Model_Prec=(TP/(TP+FP))*100
Recall=(TP/(TP+FN))*100
F1=2*(1/((1/Model_Prec)+(1/Recall)))
print('Accuracy for test set = %d percent' %Model_Acc)
print('Precision for test set = %d percent' %Model_Prec)
print('Recall for test set = %d percent' %Recall)
print('F1 for test set = %d percent' %F1)
    # print('Expected class ',Y_test[i],' Predicted class ', A[i])
```