PROJECT 3 - MIDTERM
(a) Create a binary classifier to predict the quality test result using Logistic Regression

Problem Statement: The task is to develop a Logistic Regression binary classifier from scratch to predict whether each capacitor in the test set will pass QC using what you've learned from the class presentations and homework, rather than using a Logistic Regression library inbuilt function.

I loaded the train and test datasets for capacitors test results having two features x1, x2 for this model. For improved classification, I added more features to the train and test data files, totaling eight (x1, x2,x3,...x8). On the training dataset, we'll train the model and predict the outcome class on the test dataset. Set the weights and learning rate, then use the sigmoid function to forecast the probability of the resulting class value. I tried to minimize the cost function using the gradient descent approach by varying the values of W until the function converged at the least error value.

Method of Approach: Declared the dependencies and used the variable 'f_in' to access the training dataset file. Then I declared a string variable to read every line from the file and split it with the split function into a variable called 'd'. Then, using a for loop, I read all of the data into an empty NumPy array with the dimensions of the file's rows and columns. Then, to increase the amount of features on the file, I wrote another .txt file that reads the existing rows and features from the file and then adds new features using the for loop code provided in the problem statement.After finishing the feature addition, I opened the new 'P3train1.txt' file and divided it again using the '\t' delimiter. The issue I had was that the for loop couldn't reach the last row of the training dataset, therefore I had to manually adjust the last datapoint's values. After that, I transformed them to a data frame and created two variables, TX and TY, to separate the training dataset into inputs (TX) and labels (TY) (TY). For the testing dataset, the entire approach was followed. The hypothesis function and logistic regression model were then defined as a function that performs the complete cost computation, partial derivatives of weight and bias, and gradient descent.

The initial values of w, alpha, number of iterations and cost(J) were –

(i) w = 0
(ii) alpha = 0.15
(iii) iterations = 25000
(iv) J = 0.6931471805599454

The plot was plotted and the final J value was printed. Finally to evaluate the model the confusion matrix is printed.

The following are the final values:

(i) w = [[2.07834063]
[-10.73567469]
[3.7435557]
[-8.32357438]
[-0.99468771]
[-10.17715364]
[3.01604057]
[-2.00994744]]

(ii) J = 0.48146643176337556

(iii) alpha = 0.15

(iv) Confusion Matrix: FP=3 FN=1 TP=15 TN=14

(v) Accuracy for test set = 87 percent
Precision for test set = 83 percent
Recall for test set = 93 percent
F1 for test set = 88 percent

Output and Screenshot:

jupyter Ravi_Adithya_P3_LR Last Checkpoint: 15 minutes ago (unsaved changes)

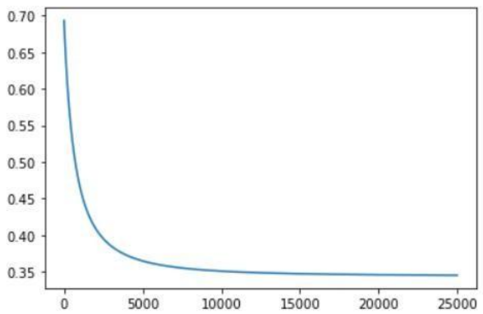
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

Run Code

```
cost after 5000 iteration is :  0.36477706446421865
cost after 7500 iteration is :  0.3551674305823984
cost after 10000 iteration is :  0.3509163258575192
cost after 12500 iteration is :  0.34867706401580817
cost after 15000 iteration is :  0.34736173851360 28
cost after 17500 iteration is :  0.3465306592189422
cost after 20000 iteration is :  0.34597831253648814
cost after 22500 iteration is :  0.3455977501614133
```

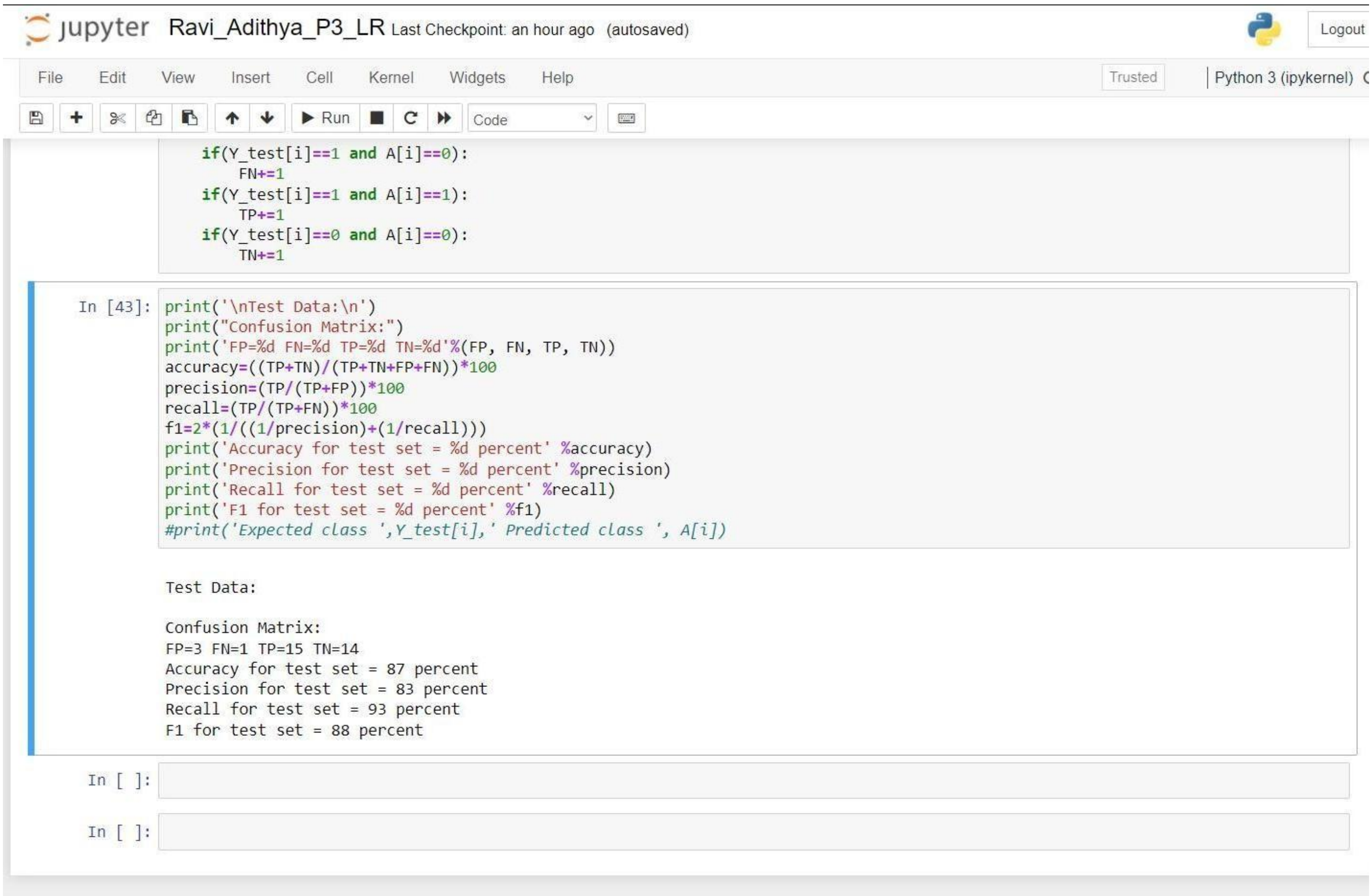
In [35]:
```
plt.plot(np.arange(iteration),cost_list)
```

Out[35]:
```
[<matplotlib.lines.Line2D at 0x12b1370fcd0>]
```



In [36]:
```
print(W)
Z = np.dot(W.T, X_test) + B
A = sigmoid_fn(Z)
m = X_test.shape[1]
n = X_test.shape[0]
```

```
[[  2.07834063]
 [-10.73567469]
 [  3.7435557 ]
```



jupyter Ravi_Adithya_P3_LR Last Checkpoint: an hour ago (autosaved)

```
    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
```

In [43]:
```
print('\nTest Data:\n')
print("Confusion Matrix:")
print('FP=%d FN=%d TP=%d TN=%d'%(FP, FN, TP, TN))
accuracy=((TP+TN)/(TP+TN+FP+FN))*100
precision=(TP/(TP+FP))*100
recall=(TP/(TP+FN))*100
f1=2*(1/((1/precision)+(1/recall)))
print('Accuracy for test set = %d percent' %accuracy)
print('Precision for test set = %d percent' %precision)
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])
```

```
Test Data:

Confusion Matrix:
FP=3 FN=1 TP=15 TN=14
Accuracy for test set = 87 percent
Precision for test set = 83 percent
Recall for test set = 93 percent
F1 for test set = 88 percent
```

In [ ]:

In [ ]:

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

f = input("Enter the name of your train file: ")
f_in= open(f, "r")

s = f_in.readline()
d = s.split("\t")
rows = int(d[0])
columns = int(d[1])+1
data = np.zeros([rows, columns])

for k in range(rows):
        s = f_in.readline()
        t = s.split("\t")
        #t[-1]=t[-1].strip()
        for j in range(columns):
                data[k,j]= float(t[j])
        #print(data[k,j])
#print(data[k])
```

```
f_in.close()
f_out = open('P3train1.txt', "w")

#for loop to increase the number of features
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):
				f_out.write(str(temp)+"\t")
	f_out.write(str(y)+"\n")
f_out.close()

f_in= open('P3train1.txt', "r")
s = f_in.readline()
d=s.split("\t")
columns = columns+6
data1 = np.zeros([rows, columns])
for k in range(rows-1):
	s = f_in.readline()
	t = s.split("\t")
	#t[-1]=t[-1].strip()
	for j in range(columns):
		data1[k,j]= float(t[j])
		#print("k =", k)
		#print("J =", j)
	#print(data[k,j])
#print(data[k])

data1[84,0] = -0.0063364
data1[84,1] = 4.014996496e-05
data1[84,2] = 0.99927
data1[84,3] = -0.006331774428
data1[84,4] = 4.0120655485579195e-05
data1[84,5] = 0.9985405329
data1[84,6] = -0.00632715223266756
data1[84,7] = 4.009136740707473e-05
data1[84,8] = 0.0

print(data1.shape)

f_in.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])

#to work on the test file
file = input("Enter the name of your test file: ")
f_in= open(file, "r")

s = f_in.readline()
d = s.split("\t")
rows = int(d[0])
columns = int(d[1])+1
data = np.zeros([rows, columns])
for k in range(rows):
        s = f_in.readline()
        t = s.split("\t")
        t[-1]=t[-1].strip()

        for j in range(columns):
                data[k,j]= float(t[j])
f_in.close()
f_out = open('P3test1.txt', "w")

#for loop to increase the number of features
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):
                                f_out.write(str(temp)+"\t")
        f_out.write(str(y)+"\n")
f_out.close()

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

s = f_in.readline()
d = s.split("\t")
columns = columns+6
data2= np.zeros([rows, columns])
```

```
for k in range(rows-1):
    s2 = f_in.readline()
    t1 = s2.split("\t")
    t1[-1]=t1[-1].strip()
    #print(t1)
    for j in range(columns):
        data2[k,j] = float(t1[j])

#print(data2) #print(data2.shape)f_in.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
X_test = X_test.T
Y_test = Y_test.reshape(1, X_test.shape[1])

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_fn(x):
    return 1/(1 + np.exp(-x))

def log_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_fn(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 = 25000
alpha =    0.15
W, B, cost_list = log_model(X_train, Y_train, alpha, iteration)
plt.plot(np.arange(iteration),cost_list)

print(W)
Z = np.dot(W.T, X_test) + B
A = sigmoid_fn(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
Y_test=Y_test.T

#for loop to calculate the values of TN,TP,FN,FP
TP=TN=FP=FN=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:\n')
print("Confusion Matrix:")
print('FP=%d FN=%d TP=%d TN=%d'%(FP, FN, TP, TN))
accuracy=((TP+TN)/(TP+TN+FP+FN))*100
precision=(TP/(TP+FP))*100
recall=(TP/(TP+FN))*100
f1=2*(1/((1/precision)+(1/recall)))
print('Accuracy for test set = %d percent' %accuracy)
print('Precision for test set = %d percent' %precision)
print('Recall for test set = %d percent' %recall)
print('F1 for test set = %d percent' %f1)
```