Machine Learning Homework #2

Farida Aliyeva

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

#importing the file to read

data = pd.read\_csv(r'C:\Users\ASUS\Desktop\python\exams.csv')

copied\_data = data.copy()

Importing Data and Libraries

#classifying the data into admitted and not admitted student based on exam results

e1a = data[data['admitted']==1]['exam\_1']

e2a = data[data['admitted']==1]['exam\_2']

e1na = data[data['admitted']==0]['exam\_1']

e2na = data[data['admitted']==0]['exam\_2']

#scattering the graph of admitted and not admitted students

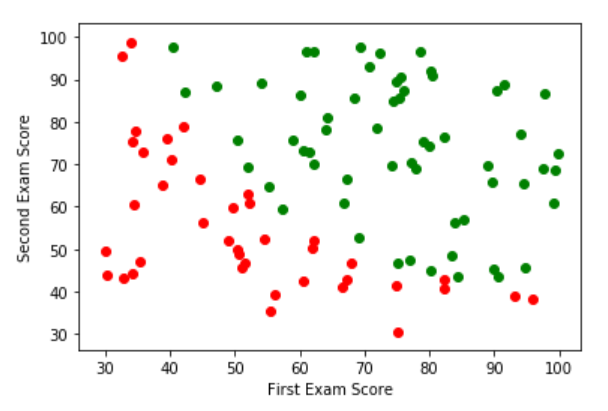
plt.xlabel("First Exam Score")

plt.ylabel("Second Exam Score")

plt.scatter(e1a, e2a, color="green")

plt.scatter(e1na, e2na, color="red")

Output:



#normalization of data (minmax normalization)

data['exam\_1'] = (data['exam\_1']-data['exam\_1'].min())/(data['exam\_1'].max()-data['exam\_1'].min())

data['exam\_2'] = (data['exam\_2']-data['exam\_2'].min())/(data['exam\_2'].max()-data['exam\_2'].min())

x = data[['exam\_1', 'exam\_2']].values

y = data['admitted'].values

#logistic sigmoid function

def sigmoid(z):

    return 1/(1+np.exp(-z))

#cost function of logistic regression

def costFunction(h, y):

    return (-y \* np.log(h)-(1-y)\*np.log(1-h)).mean()

#probability prediction

def predProb (x, theta):

    return sigmoid(np.dot(x, theta))

#prediction function

def pred(x, theta, threshold = 0.5):

    prob = predProb(x, theta)

    prob[prob>=0.5] = 1

    prob[prob<0.5] = 0

    return prob

intercept = np.ones((x.shape[0], 1))

x = np.concatenate((intercept, x), axis = 1)

theta = np.zeros(x.shape[1])

learning\_rate = 0.01

numOfits = 100000

#gradient descent

cost\_history = []

for i in range(numOfits):

    z = np.dot(x, theta)

    h = sigmoid(z)

    gradient = np.dot(x.T, (h-y))/ y.size

    theta -= learning\_rate\*gradient

    if i %1000 == 0: print(costFunction(h, y))

    cost\_history.append(costFunction(h, y))

Output:

0.6931471805599453

0.5908778825379103

0.5448305466419671

0.5074938468946265

0.47688781019259757

0.45150408980588247

0.4301990908090169

0.4121107665725063

0.39658771701610085

0.38313421082464416

0.37136936482059973

0.3609973996820093

0.3517860883426499

0.34355111346656747

0.3361446468624711

0.32944694295395643

0.32336009278123085

0.31780333736092314

0.3127095159807164

0.30802234811863377

0.303694333487735

0.29968511477637505

0.29596018997599843

0.29248989123592667

0.2892485686963154

0.28621393328480765

0.2833665237784718

0.2806892717472795

0.2781671441566069

0.27578684800944137

0.2735365848755757

0.2714058457858185

0.26938523898041555

0.26746634454931095

0.2656415912021355

0.2639041513422584

0.26224785135440043

0.2606670945959111

0.2591567950429646

0.25771231991122356

0.2563294398661752

0.25500428567691524

0.25373331036055785

0.252513256021988

0.25134112472255055

0.2502141528171634

0.24912978828669594

0.24808567066480933

0.2470796132186056

0.24610958709262779

0.24517370716777723

0.24427021942203064

0.24339748960960564

0.2425539931003887

0.24173830574279692

0.24094909563139552

0.24018511567609888

0.23944519688302257

0.2387282422684319

0.23803322133700255

0.2373591650640378

0.23670516132856548

0.2360703507505518

0.23545392289093608

0.2348551127779628

0.23427319772743466

0.2337074944281434

0.23315735626691098

0.23262217087045825

0.2321013578437734

0.23159436668680183

0.23110067487318595

0.23061978607646208

0.2301512285306095

0.229694553513165

0.22924933394028993

0.22881516306421115

0.2283916532643953

0.22797843492463252

0.22757515538895542

0.2271814779899704

0.22679708114377875

0.22642165750619248

0.22605491318542958

0.22569656700689852

0.2253463498260755

0.22500400388581984

0.22466928221479182

0.2243419480639195

0.22402177437812132

0.22370854330071996

0.22340204570819952

0.22310208077314647

0.22280845555338924

0.22252098460551412

0.22223948962107382

0.22196379908394004

0.22169374794737473

0.221429177329495

0.22116993422591802

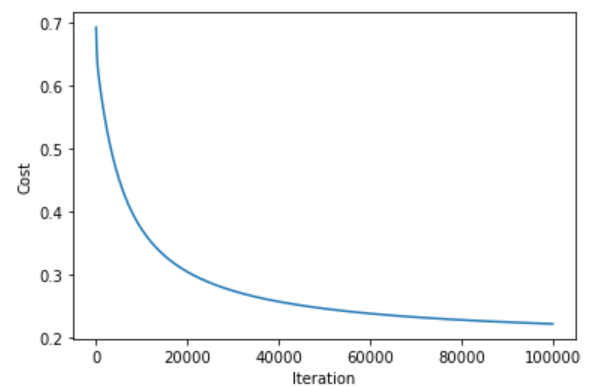
#Cost vs Iteration graph

plt.xlabel("Iteration")

plt.ylabel("Cost")

plt.plot(range(len(cost\_history)), cost\_history)

Output:



#scattering after data is normalized plus a prediction line is plotted

plt.scatter(data[data['admitted']==1]['exam\_1'], data[data['admitted']==1]['exam\_2'], color="green")

plt.scatter(data[data['admitted']==0]['exam\_1'], data[data['admitted']==0]['exam\_2'], color="red")

x\_values =  [np.min(data['exam\_1']), np.max(data['exam\_1'])]

y\_values = - (theta[0] + np.dot(theta[1], x\_values))/theta[2]

print(x\_values)

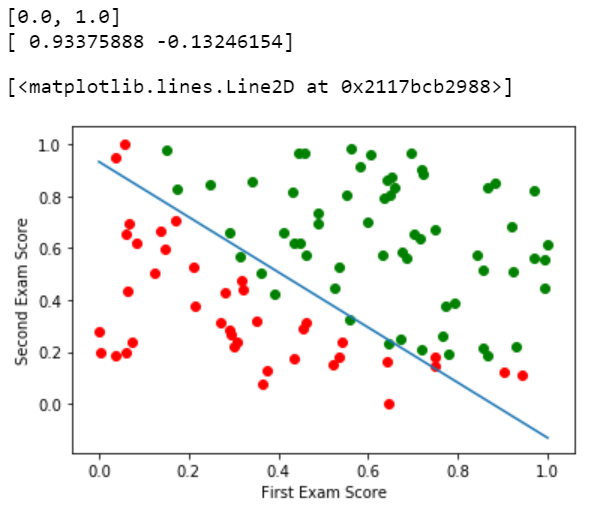
print(y\_values)

plt.xlabel("First Exam Score")

plt.ylabel("Second Exam Score")

plt.plot(x\_values, y\_values)

Output:



#using sklearn library

preds = pred(x, theta)

print('Prediction on train data')

print(preds)

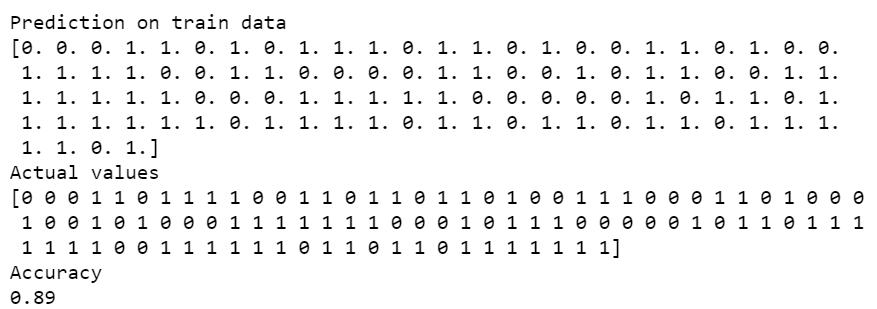
print('Actual values')

print(y)

print('Accuracy')

print(accuracy\_score(y, preds))

Output:



#Testing1

testX\_1 = np.array([55, 70])

testY\_1 = 1

testX\_1 = (testX\_1 - copied\_data['exam\_1'].min())/(copied\_data['exam\_1'].max() - copied\_data['exam\_1'].min())

testX\_1 = np.insert(testX\_1, 0, 1.0)

print('Test 1 prediction:')

print(predProb(testX\_1, theta))

Output:



#Testing2

testX\_2 = np.array ([40, 60])

testY\_2 = 0

testX\_2 = (testX\_2 - copied\_data['exam\_1'].min())/(copied\_data['exam\_1'].max() - copied\_data['exam\_1'].min())

testX\_2 = np.insert(testX\_2, 0, 1.0)

print('Test 2 prediction:')

print(predProb(testX\_2, theta))

Output:

