

General Sir John Kotelawala Defense University

Department of Electrical, Electronics & Telecommunication Engineering

Machine Learning

ET 4103

Assignment – 03

Unregularized Logistic Regression

Index No : D/ENG/22/0120/ET

Name : M. A. E. Wijesuriya

Intake : 39

Submission Date : 20/06/2025

**Q1. Utilize the given Jupyter notebook[1] for Unregularized Logistic Regression. Comment on the code and the output of the program, explaining utilized Machine Learning concepts where necessary**

The following code is a python program that demonstrates Unregularized Logistic Regression. Logistic Regression is a type of statistical model used to classify data into binary outcomes. It is a supervised learning algorithm that used a sigmoid function to generate probability values for a set of linear inputs. This probability value is then used to classify the data into one of two classes.

# File Location: The file we want to access is currently placed in the current working directory of Python.

from google.colab import drive

drive.mount('/content/drive') # Grants Colab access to Google Drive in order to retrieve the data files

%cd "/content/drive/MyDrive/ML\_files"

/content/drive/MyDrive/ML\_files

# Importing Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Reading the data file

data\_path = 'ex2data1.txt'

data = pd.read\_csv(data\_path, header=None, names = ["x1","x2","y"])

data.head()

|  |  |  |  |
| --- | --- | --- | --- |
|  | x1 | x2 | y |
| 0 | 34.623660 | 78.024693 | 0 |
| 1 | 30.286711 | 43.894998 | 0 |
| 2 | 35.847409 | 72.902198 | 0 |
| 3 | 60.182599 | 86.308552 | 1 |
| 4 | 79.032736 | 75.344376 | 1 |

# Plots the data on a scatter plot

neg = data['y'] == 0

pos = data['y'] == 1

# Marks positive data with a yellow cross

plt.scatter(data[pos]['x1'],data[pos]['x2'], marker='+', c='orange', s=60, linewidth=2, label = "Admitted")

# Marks negative data with a blue dot

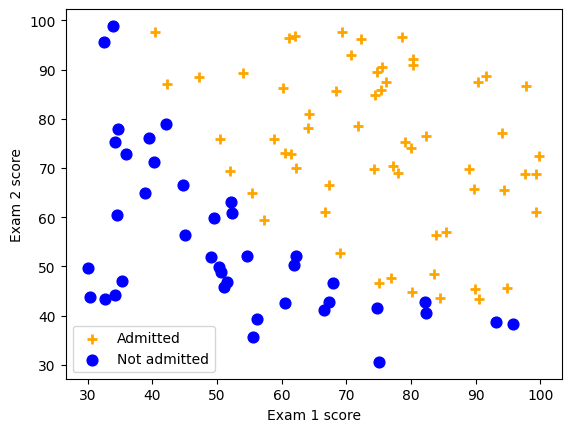
plt.scatter(data[neg]['x1'], data[neg]['x2'], c='blue', s=60, label = "Not admitted" )

plt.xlabel('Exam 1 score')

plt.ylabel('Exam 2 score')

plt.legend(loc='best')

plt.show()



# Converts data

n = data.shape[1]-1

x = data[data.columns[0:n]]

y = data[data.columns[n:n+1]]

# convert to np.array

X = x.values

# insert 1's (x\_0)

X = np.insert(X, 0, 1, axis=1)

y = y.values

### Sigmoid function

def sigmoid(z):

    return(1 / (1 + np.exp(-z)))

### Hypothesis and cost function

m = X.shape[0]

def Cost(theta, X, y):

    h = sigmoid(X.dot(theta))

    J = -1\*(1/m)\*(np.log(h).T.dot(y)+np.log(1-h).T.dot(1-y))

    if np.isnan(J.item()):

        return(np.inf)

    return(J.item())

# Calculation of the cost function for an initial (zero) value of theta

theta\_initial = np.zeros(X.shape[1]).reshape(-1,1)

Cost(theta\_initial,X,y)

*0.6931471805599453*

# Gradient function for regression

def gradient(theta, X, y):

    h = sigmoid(X.dot(theta))

    grad =(1/m)\*X.T.dot(h-y)

    return grad

# Calculating cost and gradient for theta\_initial

theta\_initial = np.zeros(X.shape[1]).reshape(-1,1)

cost = Cost(theta\_initial, X, y)

grad = gradient(theta\_initial, X, y)

print('Cost: \n', cost)

print('Grad: \n', grad)

*Cost:*

*0.6931471805599453*

*Grad:*

*[[ -0.1 ]*

*[-12.00921659]*

*[-11.26284221]]*

# Gradient descent function

def gradientDescent(X, y, theta, alpha, num\_iters):

    J\_history = np.zeros(num\_iters)

    for iter in np.arange(num\_iters):

        theta = theta - alpha\*gradient(theta,X,y)

        J\_history[iter] = Cost(theta,X,y)

    return(theta, J\_history)

theta\_initial = np.zeros(X.shape[1]).reshape(-1,1)

alpha = 0.005 # Learning Rate

iterations = 200000 # Number of gradient descent steps

theta, cost\_history = gradientDescent(X,y,theta\_initial,alpha,iterations)

theta

*array([[-29.86812752],*

*[ 0.26028092],*

*[ 0.25275129]])*

# Plot of cost history vs Iterations

plt.plot(cost\_history)

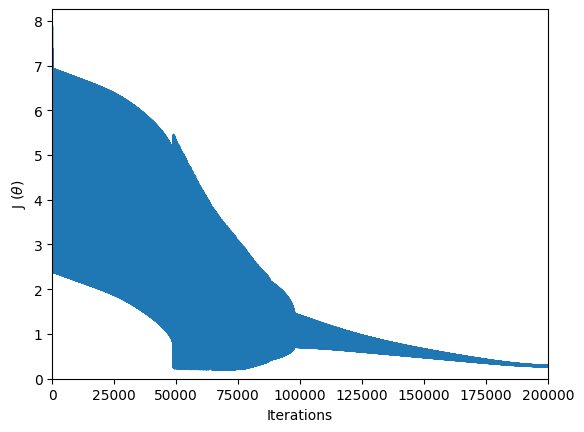
plt.ylabel('J' + ' (' + r'$\theta$' +')')

# or plt.ylabel('J' + ' (\u0398)' )

plt.xlabel('Iterations')

plt.ylim(ymin = 0)

plt.xlim(0,iterations)



## Optimization (using Scipy)

import scipy.optimize as sp

theta\_opt = sp.fmin( Cost, x0=theta\_initial, args=(X, y), maxiter=500, full\_output=True)

*Optimization terminated successfully.*

*Current function value: 0.203498*

*Iterations: 157*

*Function evaluations: 287*

# Prediction function for binary classification

def predict(theta, X, threshold=0.5):

    p = sigmoid(X.dot(theta.T)) >= threshold

    return(p.astype('int'))

theta\_opt = theta\_opt[0]

theta\_opt

*array([-25.16130062, 0.20623142, 0.20147143])*

# Calculating probability for values of scikit fmin theta

sigmoid(np.array([1, 45, 85]).dot(theta\_opt.T))

*np.float64(0.7762915904112411)*

# Calculates the accuracy of fmin model

p = predict(theta\_opt, X)

print('Train accuracy {}%'.format(100\*sum(p == y.ravel())/p.size))

*Train accuracy 89.0%*

###### Using theta obtained from gradient descent

def predict1(theta1, X, threshold=0.5):

    p1 = sigmoid(X.dot(theta1)) >= threshold

    return(p1.astype('int'))

# Calculates accuracy of manually derived model

p1 = predict1(theta,X)

print('Train accuracy {}%'.format(100\*sum(p1.ravel() == y.ravel())/p1.size))

*Train accuracy 92.0%*

#### sklearn

from sklearn import linear\_model

reg = linear\_model.LogisticRegression()

reg.fit (X[:,[1,2]],y.ravel());

from sklearn.metrics import accuracy\_score

y\_pred = reg.predict(X[:,[1,2]])

print('Train accuracy: ' + str(100\*accuracy\_score(y\_pred,y))+'%')

*Train accuracy: 89.0%*

# Plots the decision boundaries along with the scatter plot of data

## Decision boundary

neg = data['y'] == 0

pos = data['y'] == 1

plt.scatter(data[pos]['x1'],data[pos]['x2'], marker='+', c='orange', s=60, linewidth=2, label = "Admitted")

plt.scatter(data[neg]['x1'], data[neg]['x2'], c='blue', s=60, label = "Not admitted" )

xx = np.linspace(30,100,100)

yy = (-1./theta[2])\*(theta[0] + theta[1]\*xx)

plt.plot(xx,yy,color='r',label='decision boundary')

yy\_opt = (-1./theta\_opt[2])\*(theta\_opt[0] + theta\_opt[1]\*xx)

plt.plot(xx,yy\_opt,color='k',label='decision boundary (using optimization)',alpha=0.7)

coef = reg.coef\_

intercept = reg.intercept\_

ex2 = -(coef[:, 0] \* xx + intercept.item()) / coef[:,1]

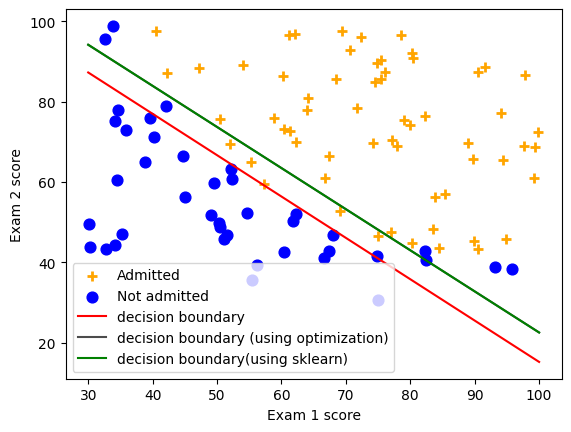
plt.plot(xx,ex2,color='g',label='decision boundary(using sklearn)')

plt.xlabel('Exam 1 score')

plt.ylabel('Exam 2 score')

plt.legend(loc='best')

plt.show()



### Alternate way to plot decision boundary (using contour)

x1\_min, x1\_max = data['x1'].min(), data['x1'].max(),

x2\_min, x2\_max = data['x2'].min(), data['x2'].max(),

xx1, xx2 = np.meshgrid(np.linspace(x1\_min, x1\_max), np.linspace(x2\_min, x2\_max))

B0 = np.linspace(x1\_min,x1\_max)

B1 = np.linspace(x2\_min,x2\_max)

#xx, yy = np.meshgrid(B0, B1, indexing='xy')

Z = np.zeros((B0.size,B1.size))

def h(x1,x2):

    stacked = np.hstack((x1,x2))

    a = np.insert(stacked,0,1)

    return a.dot(theta)

for i in range (B0.size):

    for j in range(B1.size):

        Z[i,j] = h(xx1[i,j],xx2[i,j])

plt.contour(xx1,xx2,Z,[0.5],colors='r')

neg = data['y'] == 0

pos = data['y'] == 1

plt.scatter(data[pos]['x1'],data[pos]['x2'], marker='+', c='orange', s=60, linewidth=2, label = "Admitted")

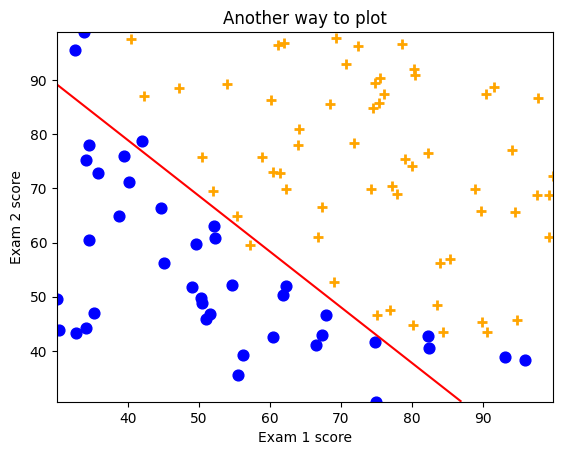
plt.scatter(data[neg]['x1'], data[neg]['x2'], c='blue', s=60, label = "Not admitted" )

plt.xlabel('Exam 1 score')

plt.ylabel('Exam 2 score')

plt.title('Another way to plot')

plt.show()



The above code demonstrates Unregularized Logistic Regression being performed on a common dataset of student scores on two exams that aims to classify the scores into two categories: admitted and not admitted.

The dataset is loaded from a text file, visualized using a scatter plot, and then converted into NumPy arrays for mathematical operations.

First, a custom Sigmoid function, cost function, and gradient function are designed based on the mathematical theory behind them. This allows us to understand the actual mathematical calculations performed during logistic regression. Two different methods are used in this program, in order to find the best parameters (theta): First, a custom Gradient Descent function is run over 20,000 iterations in order to calculate the value of theta that gives the minimal cost function. The other method uses the built-in scipy.optimize.fmin function to achieve the same goal, albeit through different means.

After training these models, the program compares the accuracy of each, with the manually derived model giving an accuracy of 92.0%, while the scipy fmin model gives us an accuracy of 89.0%. This allows us to compare how manual and library-based optimizations work side by side. Additionally, the program uses sklearn’s built in Logistic Regression tool as a third model to compare on the data. This too, gives us an accuracy of 89.0%

Finally, the decision boundaries of all three models are plotted over the scatter plot of the data in order to visualize how well each method separates the two classes. This gives us a clear picture of how logistic regression draws a line (or plane) to divide different outcomes, and how changing the optimization approach can affect the final boundary. Overall, this program ties together data handling, math, and visualization to demonstrate the inner workings of logistic regression in a very hands-on way.