Entrega Práctica 2: Regresión Logística

Autores	Correo
Clara Daniela Sima	**csima@ucm.es**
Stiven Arias Giraldo	**starias@ucm.es**

Parte 1 - Regresión Logística

Cálculo vectorizado mediante regresión logística del gradiente y de la función de coste con dos variables de X (siendo estos valores notas de éxamenes) para determinar si los alumnos que hicieron los exámenes están aprobados o no. El gradiente calculado se guarda en un vector **Theta** y el coste en **J**. Además, se hace uso de **opt.fmin_tnc** para calcular el valor óptimo de Theta.

Sección de código

```
import numpy as np
from numpy.lib import diff
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy . optimize as opt

def read_data():
    """
    Reads the data of the file and return the result as a float
    """
    valores = read_csv("./src/ex2data1.csv", header=None).to_numpy()
    return valores.astype(float)


def data_exam_graph(X, Y):
    """
        Draw the first graph of the exercise. The results of both exams
        showing the admitted and Not admitted exams
    """
    # Results of admitted
    pos = np.where (Y == 1)
    # Result of not admitted
    neg = np.where (Y == 0)

    fig = plt.figure()
    plt.ylabel('Exam Score 2', c = 'k', size='15')
    plt.xlabel('Exam Score 1', c = 'k', size='15')
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label="Admitted")
    plt.scatter(X[neg, 0], X[neg, 1], c='#c6ce00', label="No Admitted")
    plt.legend(loc='lower left')
```

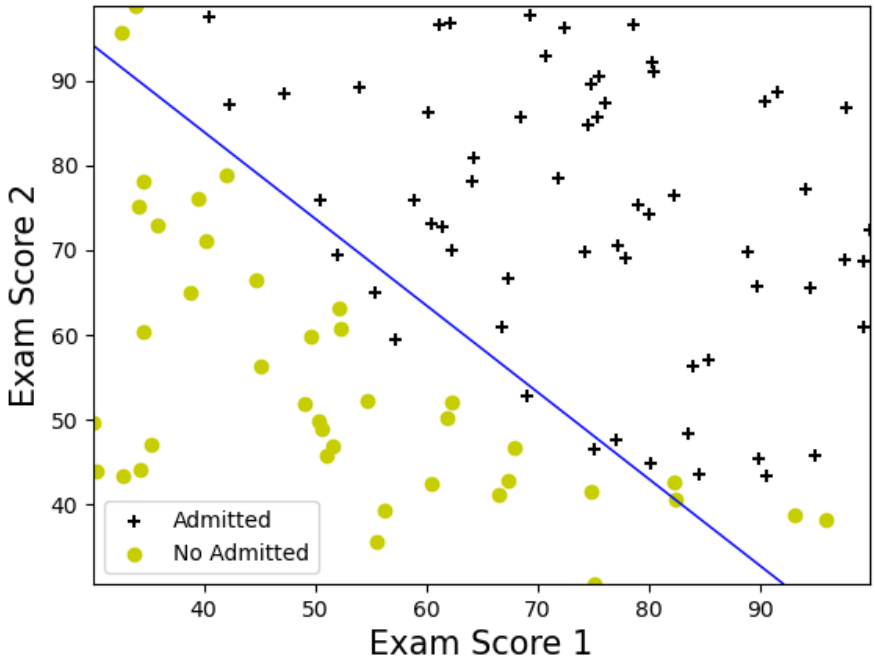
```
def sigmoide_fun(Z):
    G = (1 / (1 + (np.exp(-Z))))

    return G


def new_Theta(Theta, X, Y):
    """
        Calculate the new value of Theta with matrix
    """
    Z = np.matmul(X, Theta)
    S = sigmoide_fun(Z)
    Diff = S - Y

    NewTheta = (1 / len(Y)) * np.matmul(np.transpose(X), Diff)
    return NewTheta


def fun_J(Theta, X, Y):
    """
        Calculates the J function of the cost
        of the Logistic Regresion
    """
    S = sigmoide_fun(np.matmul(X, Theta))
    Sum1 = np.matmul(Y, np.log(S + 0.00001))

    # This add is to dodge the log(0)
    Diff = (1 - S) + 0.00001
    Sum2 = np.matmul((1 - Y), np.log(Diff))

    Sum = Sum1 + Sum2

    return (-1 / len(X)) * Sum


def lineal_fun_graph(X, Y, Theta):
    x1_min, x1_max = X[:, 1].min(), X[:, 1].max()
    x2_min, x2_max = X[:, 2].min(), X[:, 2].max()

    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
                           np.linspace(x2_min, x2_max))

    h = sigmoide_fun(np.c_[np.ones(( xx1.ravel().shape[0], 1)),
                                    xx1.ravel(),
                                    xx2.ravel()].dot(Theta))

    h = h.reshape(xx1.shape)

    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')


def percentage(Theta, X, Y):
    S = sigmoide_fun(np.matmul(X, Theta))
    i = 0
    success = 0
```

```
    for i in range(len(S)):
        if S[i] >= 0.5:
            S[i] = 1
        else:
            S[i] = 0
        if (S[i] == Y[i]):
            success += 1

    return (success / len(S)) * 100


def gradient():
    valores = read_data()
    # Add all the rows and the col(len - 1)
    # Matrix: (m, n)
    X = valores[:, :-1]
    XX = np.hstack([np.ones([np.shape(X)[0], 1]), X]) # [100, 3]
    # The -1 value add the col(len - 1)
    #(m,)
    Y = valores[:, -1]

    # (m, n + 1)
    # Row Y
    n = np.shape(XX)[1]

    Theta = np.zeros(n)

    J = fun_J(Theta, XX, Y)
    #Theta = new_Theta(Theta, XX, Y)
    #print("Función de coste: ", J)
    #print("New Thetas: ", Theta)


    # Graph section
    result = opt.fmin_tnc(func = fun_J, x0 = Theta, fprime = new_Theta, args =
(XX, Y))
    theta_opt = result[0]
    print("Precentage of succes: {} %".format(percentage(theta_opt, XX, Y)))

    data_exam_graph(X, Y)
    lineal_fun_graph(XX, Y, theta_opt)
    return 1

# main
gradient()
plt.show()
```

Gráficas - Parte 1

Distribución de los resultados de los dos exámenes de los alumnos determinando si son admitidos o no. Función lineal calculada mediante regresión logística para determinar si un alumno es admitido o no:

if: g(x,y) >= 0.5 : admitido



Resultado de la función de coste con Theta = 0 y de los nuevos valores entrenados de Theta mediante la función new_Theta del código.

```
Función de coste:  0.6931271807599427
New Thetas:  [ -0.1        -12.00921659 -11.26284221]
```

Porcentaje de éxito calculado mediante el valor de Theta óptimo

```
Precentage of succes: 89.0 %
```

Parte 2 - Regresión Logística regularizada

Cálculo vectorizado mediante regresión logística del gradiente y de la función de coste con dos variables de X (siendo estos valores test de microchips en una empresa) para determinar si los chips que son aptos o no para salir al mercado. El gradiente calculado se guarda en un vector **Theta** y el coste en **J**. Además, se hace uso de **opt.fmin_tnc** para calcular el valor óptimo de Theta y se utiliza un valor **lambda** para regularizar la función.

Sección de código

```
import numpy as np
from numpy.lib import diff
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
import scipy


def read_data():
    """
    Reads the data of the file and return the result as a float
    """
    valores = read_csv("./src/ex2data2.csv", header=None).to_numpy()
    return valores.astype(float)

def data_exam_graph(X, Y):
    """
        Draw the first graph of the exercise. The results of both exams
        showing the admitted and Not admitted exams
    """
    # Results of admitted
    pos = np.where (Y == 1)
    # Result of not admitted
    neg = np.where (Y == 0)

    plt.ylabel('Microchip 2', c = 'k', size='15')
    plt.xlabel('Microchip 1', c = 'k', size='15')
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label="y = 1")
    plt.scatter(X[neg, 0], X[neg, 1], c='#c6ce00', label="y = 0")
    plt.legend(loc='lower left')

def sigmoide_fun(Z):
    G = 1 / (1 + (np.exp(-Z)))

    return G

def fun_J(Theta, m, X, Y, lamb):
    """
        Calculates the J function of the cost
        of the Logistic Regresion
```

```
    """
    S = sigmoide_fun(np.dot(X, Theta))
    Sum1 = np.dot(Y, np.log(S))

    # This add is to dodge the log(0)
    Diff = (1 - S) + 0.00001
    Sum2 = np.dot((1 - Y), np.log(Diff))
    # First part
    Sum = Sum1 + Sum2
    Sum = (-1 / m) * Sum
    # Lambda part
    Sum3 = np.sum(np.power(Theta, 2))
    Sum += (lamb / (2 * m)) * Sum3

    return Sum

def new_theta(Theta, m, X, Y, lamb):
    """
        Calculate the new value of Theta with matrix
    """
    Z = np.matmul(X, Theta)
    S = sigmoide_fun(Z)
    Diff = S - Y

    X_t = np.transpose(X)
    NewTheta = (1 / m) * np.matmul(X_t, Diff) + (lamb/m) * Theta
    NewTheta[0] -= (lamb/m) * Theta[0]

    return NewTheta

def lineal_fun_graph(X, Y, Theta, poly):

    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()

    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
                           np.linspace(x2_min, x2_max))

    poly = poly.fit_transform(np.c_[xx1.ravel(),
                         xx2.ravel()])
    h = sigmoide_fun(np.dot(poly, Theta))

    h = h.reshape(xx1.shape)

    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    return 0

def solution(lamb = 1):
    valores = read_data()

    # Add all the rows and the col(len - 1)
    X = valores[:, :-1]
    print("Shape X: ", np.shape(X))
    poly = PolynomialFeatures(6)
    XX = poly.fit_transform(X)
```

```
    # The -1 value add the col(len - 1)
    Y = valores[:, -1]

    print("Shape XX: ", np.shape(XX))
    m = np.shape(XX)[0]
    n = np.shape(XX)[1]
    Theta = np.zeros(n)

    # lambda
    J = fun_J(Theta, m, XX, Y, lamb)
    print("J: ", J)
    T = new_theta(Theta, m, XX, Y, lamb)

    theta_op = scipy.optimize.fmin_tnc(fun_J, Theta, new_theta, args=(m, XX, Y,
lamb))

    # Graph section
    plt.title(r'$\lambda$ = ' + str(lamb))
    data_exam_graph(X, Y)
    lineal_fun_graph(X, Y, theta_op[0], poly)

    return 0

# main
plt.figure()
solution(-0.00005)
plt.show()
```

Gráficas - Parte 2

Distribución de los datos de los test de microchips extraídos del excel junto con la función logística regularizada mediante diversos valores de lambda. Para determinar si son aptos o no:

if: g(x, y) >= 0.5: admitidos