Aprendizaje automático y minería de datos: Práctica 2

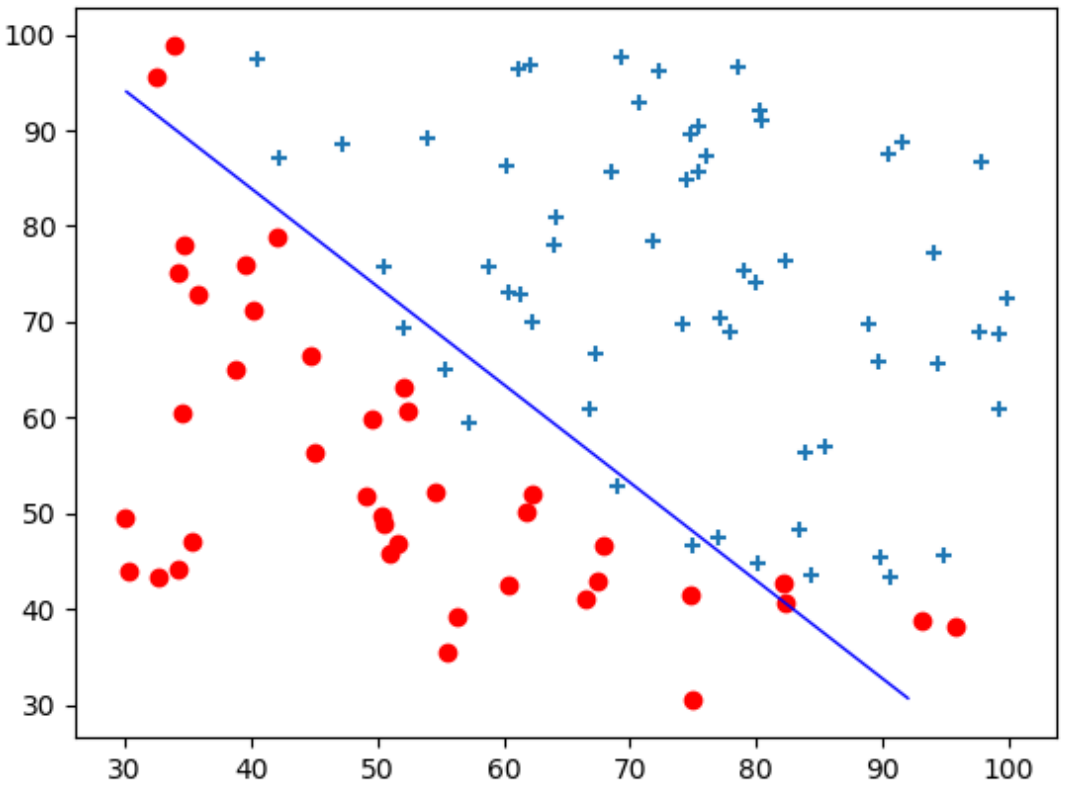
Jorge Rodríguez García y Gonzalo Sanz Lastra

Código de la práctica:

PARTE 1: REGRESIÓN LOGÍSTICA

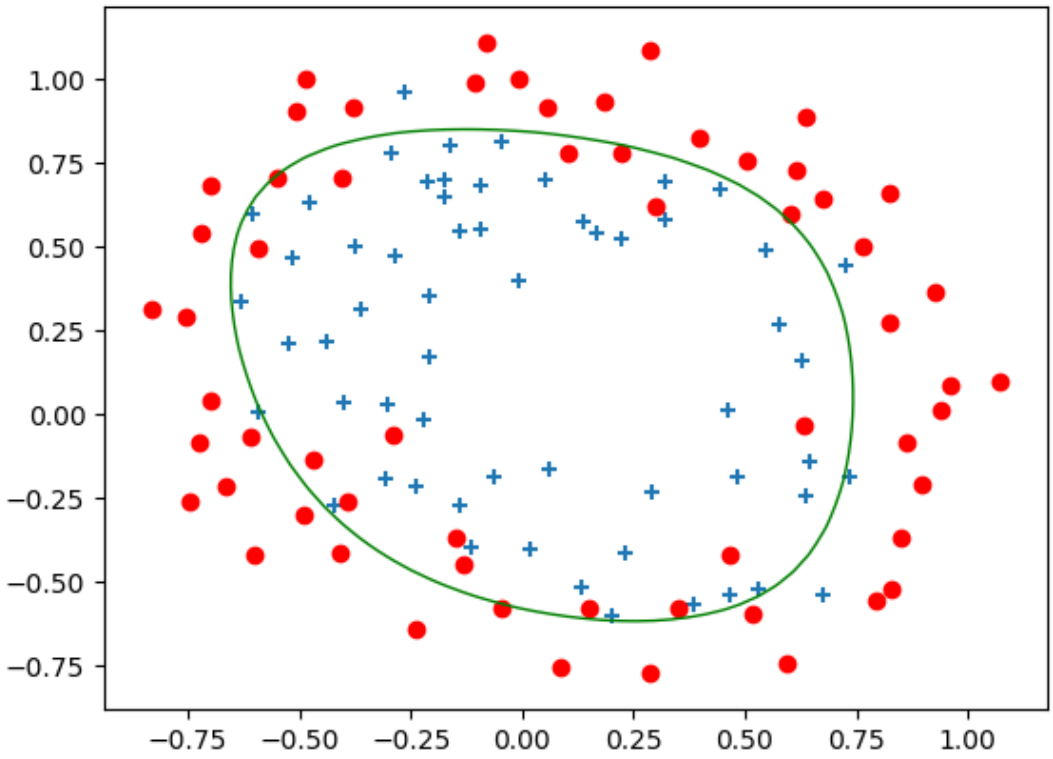
```python
import numpy as np
import scipy.optimize as opt
from pandas.io.parsers import read_csv   # para leer .csv
from matplotlib import pyplot as plt     # para dibujar las graficas

def load_csv(file_name):
    """carga el fichero csv especificado y lo devuelve en un array de numpy"""
    valores = read_csv(file_name, header=None).values

    return valores.astype(float)

def graphics(X, Y, O):
    pinta_frontera_recta(X, Y, O)

    # Obtiene un vector con los índices de los ejemplos positivos
    pos = np.where(Y == 1)
    # Dibuja los ejemplos positivos
    plt.scatter(X[pos, 1] ,X[pos, 2] ,marker= '+')

    # Obtiene un vector con los índices de los ejemplos negativos
    pos = np.where(Y == 0)
    # Dibuja los ejemplos negativos
    plt.scatter(X[pos, 1] ,X[pos, 2] , c = 'red')

    plt.savefig("frontera1.png")
    plt.show()

def pinta_frontera_recta(X, Y, O):
    """pinta la recta que separa los datos entre los que cumplen el requisito y los que no"""
    plt.figure()
    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)) # grid de cada columna de Xs

    h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)),
    xx1.ravel(), xx2.ravel()].dot(O)) # ravel las pone una tras otra

    h = h.reshape(xx1.shape)

    # el cuarto parámetro es el valor de z cuya frontera se
    # quiere pintar, en este caso el punto medio entre 0 y 1 (los que cumplen y los que no)
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')

# g(X*Ot) = h(x)
def sigmoid(Z):
    return 1/(1+np.exp(-Z))
```

```python
def cost(O, X, Y):
    """devuelve un valor de coste"""
    return -((np.log(sigmoid(X.dot(O)))).T.dot(Y) + (np.log(1-sigmoid(X.dot(O)))).T.dot(1-Y))/X.shape[0]

def gradient(O, X, Y):
    """la operacion que hace el gradiente por dentro -> devuelve un vector de valores"""
    return (X.T.dot(sigmoid(X.dot(O))-Y))/X.shape[0]

def successPercentage(X, Y, O):
    """determina el porcentaje de aciertos comparando los resultados estimados con los resultados reales"""
    results = (sigmoid(X.dot(O.T)) >= 0.5)
    results = (results == Y)
    return results.sum()/results.shape[0]

def main():
    valores = load_csv("ex2data1.csv")

    X = valores[:, :-1] # matriz X, con todas las filas y todas las columnas menos la ultima (ys)
    Y = valores[:, -1]  # matriz Y, con todas las filas y la ultima columna

    m = X.shape[0]      # numero de muestras de entrenamiento
    n = X.shape[1] + 1  # numero de variables x que influyen en el resultado y, mas la columna de 1s

    X = np.hstack([np.ones([m, 1]), X])

    O = np.zeros(n)

    # por dentro hace la funcion de gradiente, usando nuestras funciones y variables
    result = opt.fmin_tnc(func = cost, x0 = O, fprime = gradient, args=(X, Y))
    O_opt = result[0] # entre todos los resultados devueltos, este ofrece las thetas optimas

    success = successPercentage(X, Y, O_opt)
    print("Porcentaje de acierto: " + str(success*100) + "%")

    #GRAFICAS
    graphics(X, Y, O_opt)

main()
```



PARTE 2: REGULARIZACIÓN

```python
import numpy as np
import scipy.optimize as opt
from pandas.io.parsers import read_csv   # para leer .csv
from matplotlib import pyplot as plt     # para dibujar las graficas
from sklearn import preprocessing        # para polinomizar las Xs

def load_csv(file_name):
    """carga el fichero csv especificado y lo devuelve en un array de numpy"""
    valores = read_csv(file_name, header=None).values

    return valores.astype(float)

def graphics(X, Y, O, poly):
    plot_decisionboundary(X, Y, O, poly)

    # Obtiene un vector con los índices de los ejemplos positivos
    pos = np.where(Y == 1)
    # Dibuja los ejemplos positivos
    plt.scatter(X[pos, 0] ,X[pos, 1] ,marker= '+')

    # Obtiene un vector con los índices de los ejemplos negativos
    pos = np.where(Y == 0)
    # Dibuja los ejemplos negativos
    plt.scatter(X[pos, 0] ,X[pos, 1] , c = 'red')

    plt.savefig("frontera2.png")
    plt.show()

def plot_decisionboundary(X, Y, theta, poly):
    """pinta el polinomio que separa los datos entre los que cumplen el requisito y los que no"""
    plt.figure()

    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)) # grid de cada columna de Xs

    h = sigmoid(poly.fit_transform(np.c_[xx1.ravel(), xx2.ravel()]).dot(theta))# ravel las pone una tras otra

    h = h.reshape(xx1.shape)

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

# g(X*Ot) = h(x)
def sigmoid(Z):
    return 1/(1+np.exp(-Z))

def cost(O, X, Y, l):
    """devuelve un valor de coste"""
    return (-(((np.log(sigmoid(X.dot(O)))).T.dot(Y) + (np.log(1-sigmoid(X.dot(O)))).T.dot(1-Y))/X.shape[0])
    + (l/(2*X.shape[0]))*(O[1:,]**2).sum()) # igual que el coste anterior pero añadiendole esto ultimo para la regularizacion

def gradient(O, X, Y, l):
    """la operacion que hace el gradiente por dentro -> devuelve un vector de valores"""
    AuxO = np.hstack([np.zeros([1]), O[1:,]]) # sustituimos el primer valor de las thetas por 0 para que el termino independiente
    # no se vea afectado por la regularizacion (lambda*0 = 0)
    return (((X.T.dot(sigmoid(X.dot(O))-Y))/X.shape[0])
    + (l/X.shape[0])*AuxO) # igual que el gradiente anterior pero añadiendole esto ultimo para la regularizacion

def successPercentage(X, Y, O):
    """determina el porcentaje de aciertos comparando los resultados estimados con los resultados reales"""
    results = (sigmoid(X.dot(O.T)) >= 0.5)
    results = (results == Y)
    return results.sum()/results.shape[0]

def main():
    valores = load_csv("ex2data2.csv")

    X = valores[:, :-1] # matriz X, con todas las filas y todas las columnas menos la ultima (ys)
    Y = valores[:, -1]  # matriz Y, con todas las filas y la ultima columna

    m = X.shape[0]      # numero de muestras de entrenamiento

    polyGrade = 6       # a mayor grado, mayor ajuste
    poly = preprocessing.PolynomialFeatures(polyGrade)
    Xpoly = poly.fit_transform(X) # añade automaticamente la columna de 1s

    n = Xpoly.shape[1] # numero de variables x que influyen en el resultado y, mas la columna de 1s

    O = np.zeros(n)
    l = 1 # cuanto mas se aproxime a 0, mas se ajustara el polinomio (menor regularizacion)

    # por dentro hace la funcion de gradiente, usando nuestras funciones y variables
    result = opt.fmin_tnc(func = cost, x0 = O, fprime = gradient, args=(Xpoly, Y, l))
    O_opt = result[0] # entre todos los resultados devueltos, este ofrece las thetas optimas

    success = successPercentage(Xpoly, Y, O_opt)
    print("Porcentaje de acierto: " + str(success*100) + "%")

    #GRAFICAS
    graphics(X, Y, O_opt, poly)

main()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00