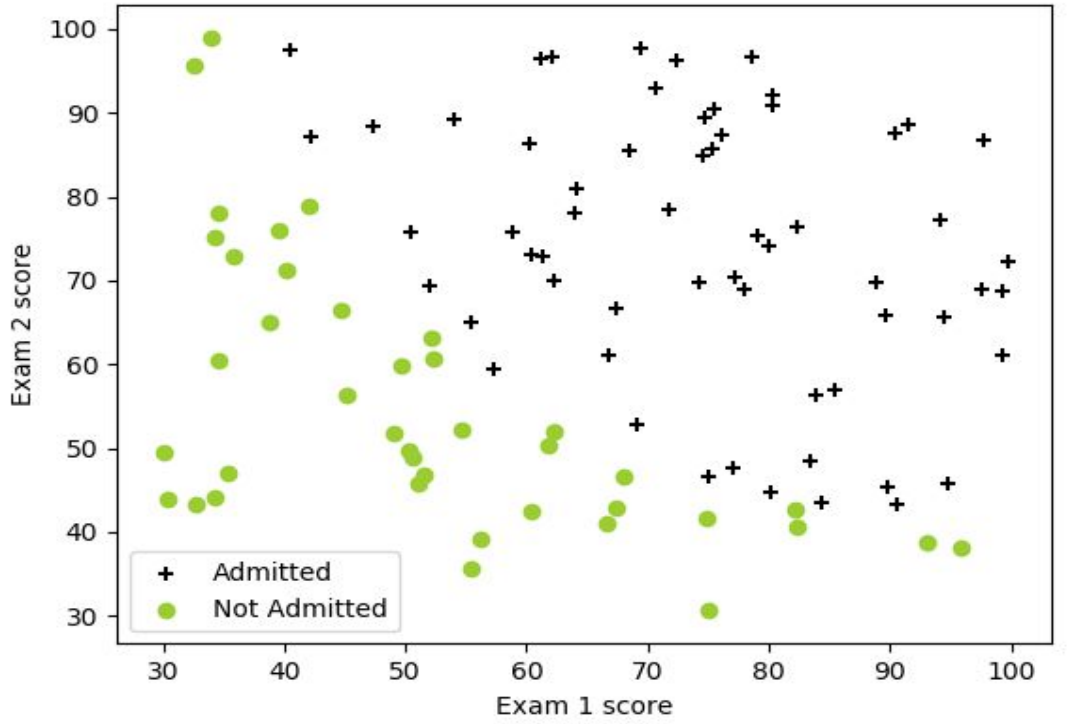
Memoria Técnica. Práctica 2

Sergio Gavilán Fernández sgavil01@ucm.es

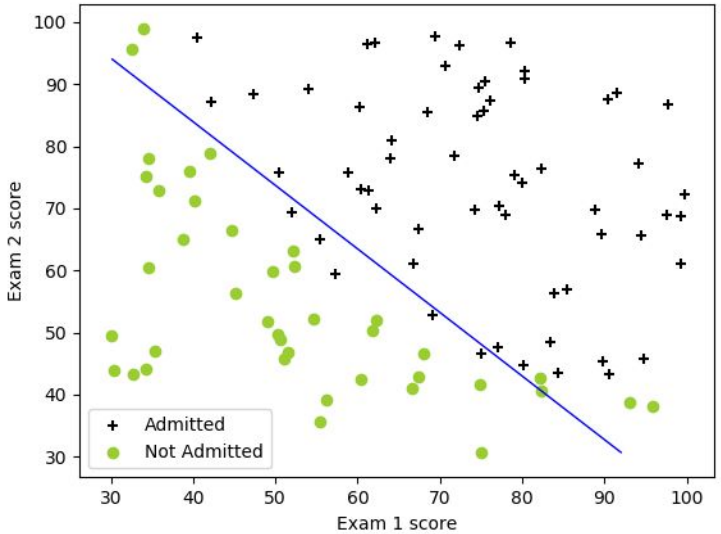
Alejandro Villar Rubio alvill04@ucm.es

PARTE 1: Regresión logística

Partiendo del siguiente conjunto de datos:



Hemos utilizado la función sigmoide para poder implementar la función de coste y su gradiente, de esta forma hemos hecho uso de *scipy.optimize.fmin_tnc* para calcular la θ que minimiza la función de coste. De este modo podemos dibujar la frontera de la regresión logística:



Posteriormente hemos calculado el porcentaje de ejemplos de entrenamiento que se clasifican correctamente, obteniendo un 89% de acierto.

El código de la parte 1 es el siguiente:

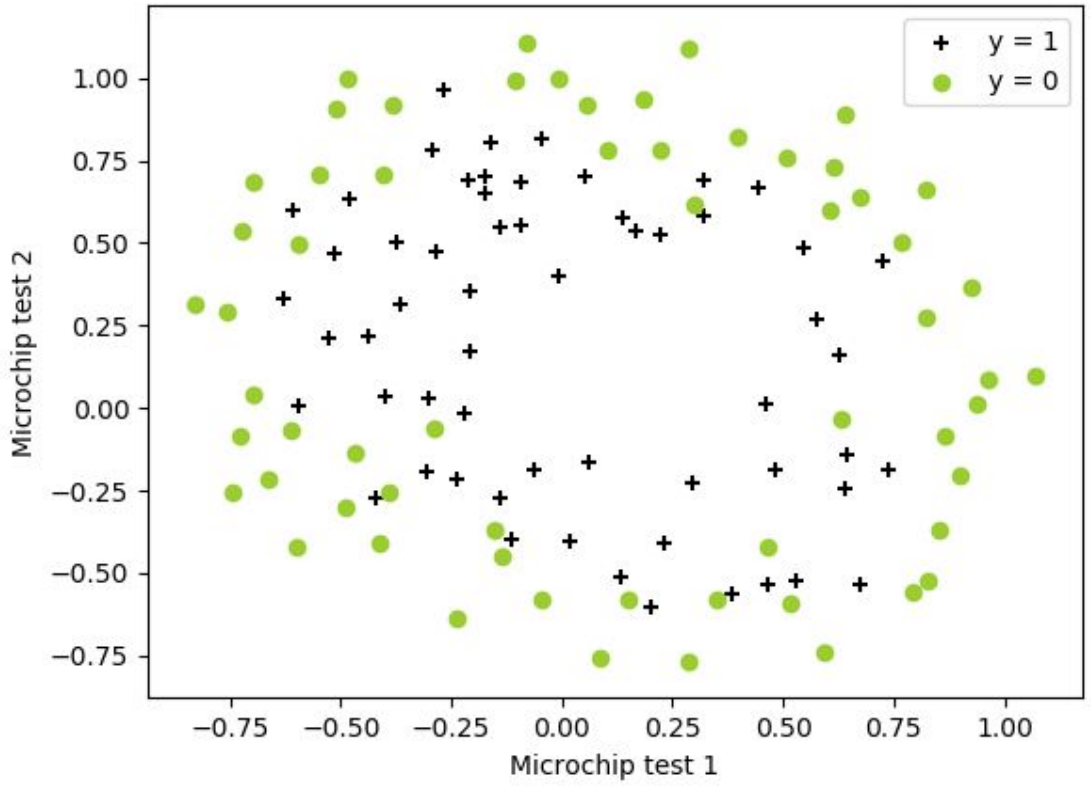
```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt

# Carga el fichero csv especificado y lo devuelve en un array de numpy

def carga_csv(file_name):
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)


def dibuja_grafica(X, Y):
    admitted = np.where(Y == 1)
    notAdmitted = np.where(Y == 0)

    plt.scatter(X[admitted, 0], X[admitted, 1],
                marker='+', c='k', label='Admitted')
    plt.scatter(X[notAdmitted, 0], X[notAdmitted, 1], marker='o',
                c='yellowgreen', label='Not Admitted')

    plt.legend()

    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')

    return plt


def dibuja_h(Theta, X, Y, plt):
    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))

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

    #  el cuarto parámetro es el valor de z cuya frontera se
```

```
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    plt.show()
    plt.savefig('parte1.jpg')
    plt.close()


def funcion_coste(Theta, X, Y):
    m = len(X)
    H = sigmoide(np.matmul(X, Theta))
    return ((-1/m) * (np.dot(np.log(sigmoide(np.dot(X, Theta))).T, Y) +
                      (np.dot(np.log(1 - sigmoide(np.dot(X, Theta))).T, 1 -
Y))))


def funcion_gradiente(Theta, X, Y):
    m = X.shape[0]
    return (1 / m) * (np.dot(X.T, sigmoide(np.dot(X, Theta)) - Y))


def h(X, Theta):
    return (1 / (1 + np.exp(np.dot(X, -Theta.T))))


def sigmoide(z):
    return (1 / (1 + np.exp(-z)))  # z = theta.T * x


def regresion_logistica(Theta, X, Y):
    gradiente = funcion_gradiente(Theta, X, Y)
    coste = funcion_coste(Theta, X, Y)

    print("Función gradiente:", gradiente)
    print("Función coste:", coste)

    result = opt.fmin_tnc(func=funcion_coste, x0=Theta,
                          fprime=funcion_gradiente, args=(X, Y))
    Theta_Opt = result[0]

    return Theta_Opt


def ev_regresion(X, Y, Theta):
    X_ev = sigmoide(np.dot(X, Theta))
    X_ev = (X_ev < 0.5)
    Y_ev = (Y == 0)
    res = (X_ev == Y_ev)
```

```
    porcentaje = ((np.sum(res)) * 100) / res.shape
    return porcentaje


def main():
    datos = carga_csv("ex2data1.csv")
    X = np.delete(datos, np.shape(datos)[1]-1, axis=1)
    Y = datos[:, datos.shape[1]-1]

    plt = dibuja_grafica(X, Y)

    onesColumn = np.ones((X.shape[0], 1))
    X = np.hstack((onesColumn, X))

    Theta = np.zeros(X.shape[1])

    Theta = regresion_logistica(Theta, X, Y)
    dibuja_h(Theta, np.delete(X, 0, axis=1), Y, plt)

    correctos = ev_regresion(X, Y, Theta)
    print("El", correctos[0], "% se clasifican correctamente.")


main()
```

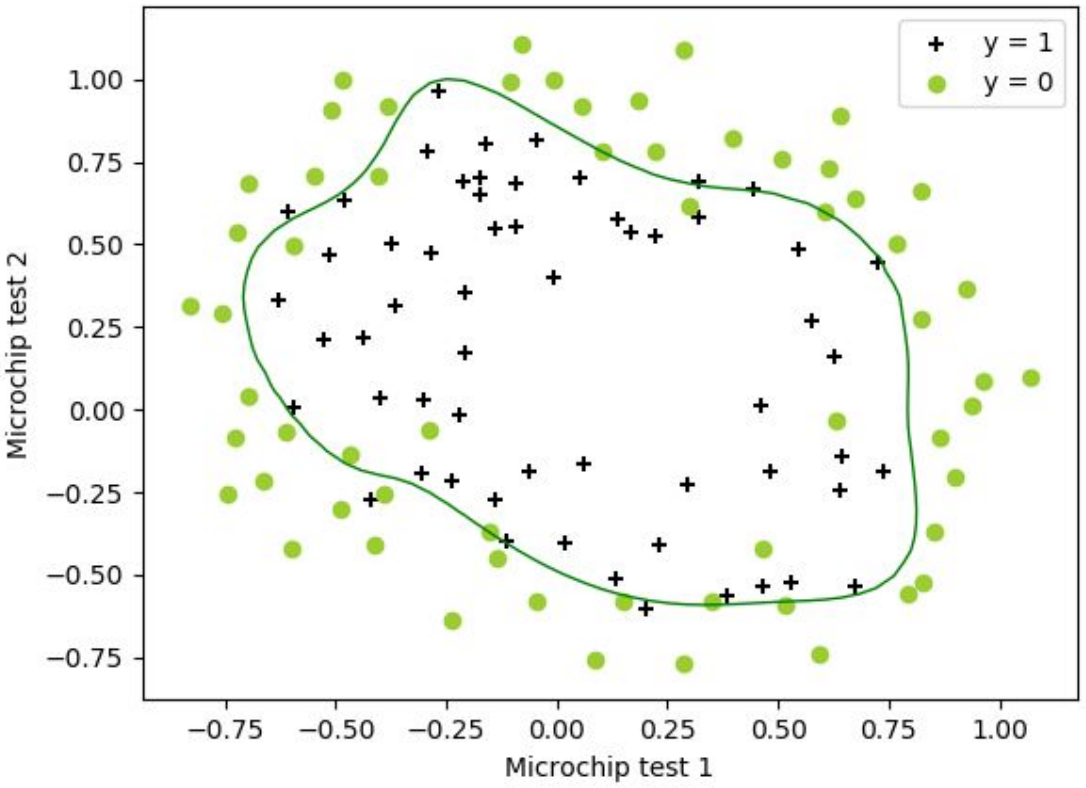
PARTE 2: Regresión logística regularizada

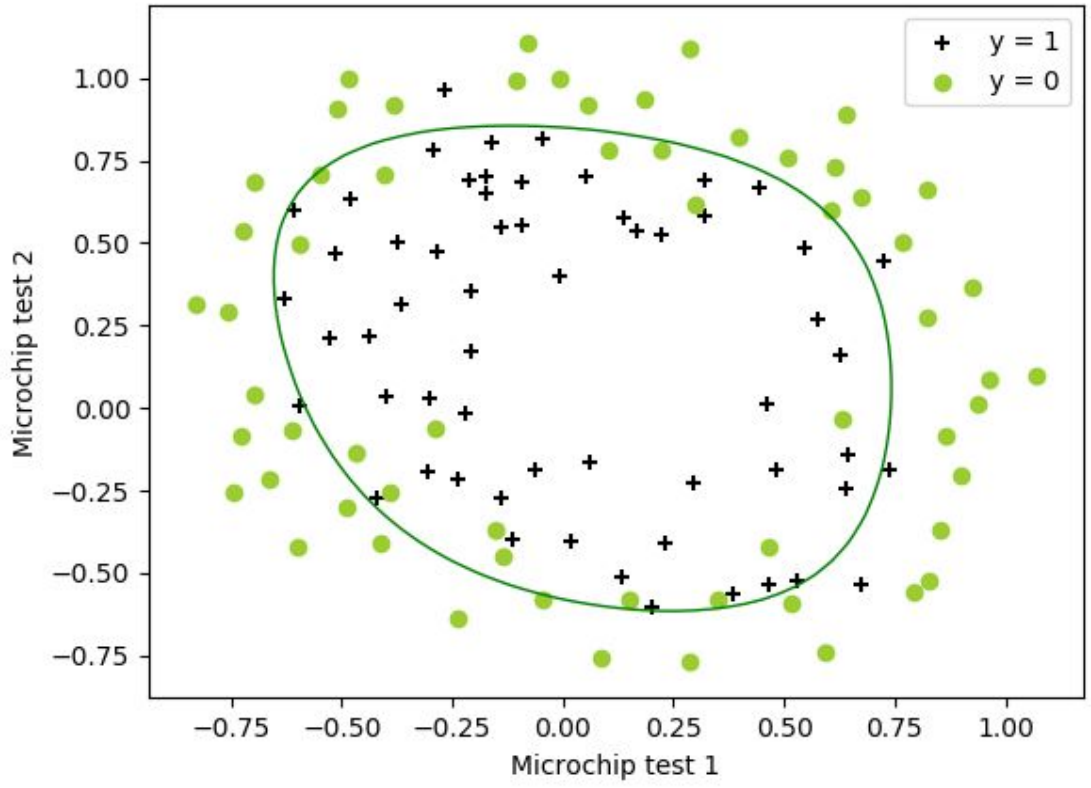
Partiendo de estos datos sobre microchips que superan o no un control de calidad:



En primer lugar hemos mapeado los atributos combinando los atributos originales para obtener unos mejores resultados, después, hemos calculado la función de coste y su gradiente para posteriormente calcular la θ óptima utilizando *scipy.optimize.fmin_tnc.*

Finalmente se ha experimentado con distintos valores de lambda (lambda = 0,lambda = 1)





Podemos observar que a mayor lambda la frontera se ajusta menos a los ejemplos de entrenamiento pero generará predicciones más ajustadas (dentro de un rango)

El código de la parte 2 es el siguiente:

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

# Carga el fichero csv especificado y lo devuelve en un array de numpy


def carga_csv(file_name):
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)


def dibuja_grafica(X, Y):
    admitted = np.where(Y == 1)
    notAdmitted = np.where(Y == 0)

    plt.scatter(X[admitted, 0], X[admitted, 1],
                marker='+', c='k', label='y = 1')
    plt.scatter(X[notAdmitted, 0], X[notAdmitted, 1], marker='o',
                c='yellowgreen', label='y = 0')

    plt.legend()

    plt.xlabel('Microchip test 1')
    plt.ylabel('Microchip test 2')

    return plt


def dibuja_h(Theta, X, Y, plt, 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))

    h = sigmoide(poly.fit_transform(
        np.c_[xx1.ravel(), xx2.ravel()]).dot(Theta))
    h = h.reshape(xx1.shape)
```

```
    #  el cuarto parámetro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='g')
    plt.show()
    plt.close()


def f_gradiente(Theta, X, Y, lam):
    m = len(X)
    tempTheta = np.r_[[0], Theta[1:]]
    return (((1 / m) * np.dot(X.T, sigmoide(np.dot(X, Theta)) - Y))
            + ((lam / m) * tempTheta))


def f_coste(Theta, X, Y, lam):
    m = len(X)
    return (((-1 / m) * (np.dot(np.log(sigmoide(np.dot(X, Theta))).T, Y)
                         + np.dot(np.log(1 - sigmoide(np.dot(X, Theta))).T,
(1 - Y))))
            + ((lam / (2 * m)) * np.sum(Theta**2, initial=1)))


def sigmoide(z):
    return 1 / (1 + np.exp(-z))


def regresion_logistica_regularizada(X, Y, Theta, lam):
    poly = PolynomialFeatures(6)
    X_poly = poly.fit_transform(X)

    grad = f_gradiente(Theta, X_poly, Y, lam)
    coste = f_coste(Theta, X_poly, Y, lam)

    result = opt.fmin_tnc(func=f_coste, x0=Theta,
                          fprime=f_gradiente, args=(X_poly, Y, lam))
    Theta_Opt = result[0]
    return poly, Theta_Opt


def main():
    datos = carga_csv("ex2data2.csv")
    X = np.delete(datos, np.shape(datos)[1]-1, axis=1)
    Y = datos[:, datos.shape[1]-1]

    Theta = np.zeros(28)
    lam = 1
```

```
    plt = dibuja_grafica(X, Y)
    poly, Theta_Opt = regresion_logistica_regularizada(X, Y, Theta, lam)

    dibuja_h(Theta_Opt, X, Y, plt, poly)


main()
```