Practica 2

Antonio Rodriguez Hurtado y Miguel Ferreras Chumillas

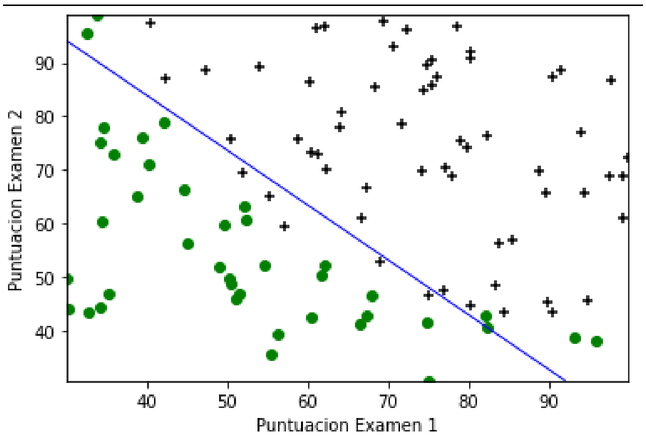
Parte 1

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

#Definicion de variables -------------------------

#   Convertimos el .csv en un array de numpy
valores = read_csv("ex2data1.csv", header=None).to_numpy().astype(float)

#   Separamos las coordenadas en dos arrays, X e Y
X = valores[:, :-1]
Y = valores[:, -1]

#   Obtenemos el numero de ejemplos de entrenamiento (m) y de atributos (n)
m = np.shape(X)[0]
n = np.shape(X)[1]

#   Añadimos una columna de unos en X para facilitar los calculos con matrices
OX = np.hstack([np.ones([m, 1]), X])

theta = [0] * (n + 1)


#Definicion de funciones -------------------------

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

def hip(X, theta):
    return sigmoid(np.matmul(X, theta))

def cost(theta, X, Y):
    return (-1 / m) * (np.dot(Y, np.log(hip(X, theta))) + np.dot((1 - Y), np.log(1 - hip(X, theta))))

def gradient(theta, X, Y):
    return (1 / m) * np.matmul((hip(X, theta) - Y), X)

def porcentaje():
    ok = 0
    i = 0
    for h in hip(OX, theta_opt):
        if h >= 0.5:
            if Y[i] == 1.0:
                ok +=1
        else:
            if Y[i] == 0.0:
                ok +=1
        i +=1
    return (ok / m)

def pinta_frontera_recta(X, Y, theta):
    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 = sigmoid(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')


#Ejecucion -------------------------

#   Obtenemos el coste y los valores de theta optimos
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(OX, Y))
theta_opt = result[0]

#   Definimos los puntos a dibujar
plt.xlabel('Puntuacion Examen 1')
plt.ylabel('Puntuacion Examen 2')
pos = np.where(Y == 1.0)
```

```
neg = np.where(Y == 0.0)

#   Dibujamos la grafica
plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g')
pinta_frontera_recta(X, Y, theta_opt)

#   Calculamos el porcentaje de aciertos del modelo
print("Porcentaje de aciertos: " + str(porcentaje()))
```

Resultado de la ejecucion:

Grafica de la recta frontera:



Parte 2

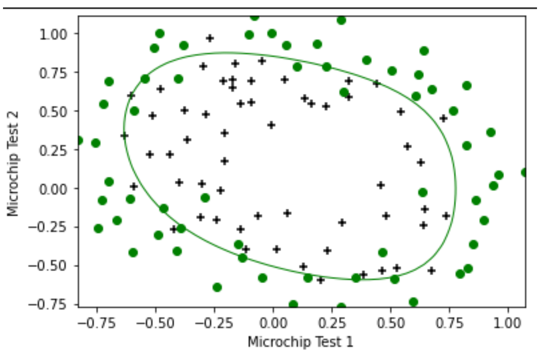
```
#   Librerias
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
from sklearn.preprocessing import PolynomialFeatures


#Definicion de variables -------------------------

#   Convertimos el .csv en un array de numpy
valores = read_csv("ex2data2.csv", header=None).to_numpy().astype(float)

#   Separamos las coordenadas en dos arrays, X e Y
X = valores[:, :-1]
Y = valores[:, -1]

#   Creamos los nuevos atributos de cada ejemplo de entrenamiento
poly = PolynomialFeatures(6).fit_transform(X)

#   Obtenemos el numero de ejemplos de entrenamiento (m) y de atributos (n)
m = np.shape(poly)[0]
n = np.shape(poly)[1]

theta = [0] * (n)
lamda = 0.3


#Definicion de funciones -------------------------


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

def hip(X, theta):
```

```
    return sigmoid(np.matmul(X, theta))

def cost(theta, X, Y):
    return (-1 / m) * (np.dot(Y, np.log(hip(X, theta))) + np.dot((1 - Y), np.log(1 - hip(X, theta)))) + (lamda / (2 * m)) * np.sum(np.

def gradient(theta, X, Y):
    gradiente =  ((1 / m) * np.matmul((hip(X, theta) - Y), X))
    result = gradiente[0]
    i = 1
    for e in gradiente[1:]:
        result = np.append(result, e + ((lamda / m) * theta[i]))
        i += 1
    return result

def plot_decisionboundary(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))
    h = sigmoid(PolynomialFeatures(6).fit_transform(np.c_[xx1.ravel(), xx2.ravel()]).dot(theta))
    h = h.reshape(xx1.shape)

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


#Ejecucion -------------------------


#   Obtenemos el coste y los valores de theta optimos
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(poly, Y))
theta_opt = result[0]

#   Definimos los puntos a dibujar
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')
pos = np.where(Y == 1.0)
neg = np.where(Y == 0.0)

#   Dibujamos la grafica
plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g')
plot_decisionboundary(X, Y, theta_opt, poly)
```

Tras el desarrollo de la segunda parte de la practica probamos cambiando los valores de lamda para ver como afectaba al resultado del aprendizaje.

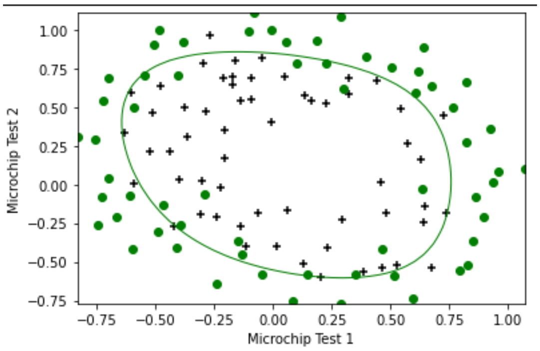
Primero empezamos con unos valores razonables [0.1, 0.5, 1, 5]

```
lamda = 0.1
print("Porcentaje de aciertos: " + str(porcentaje()))
```



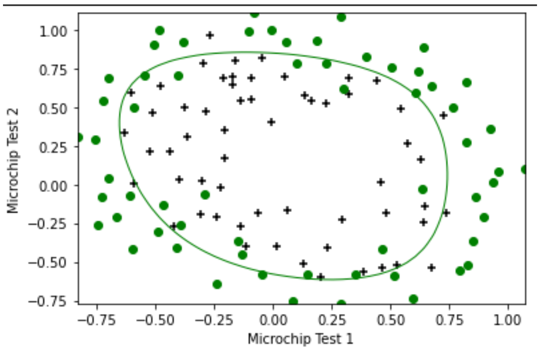

```
Porcentaje de aciertos: 0.8389830508474576
```

lamda = 0.5



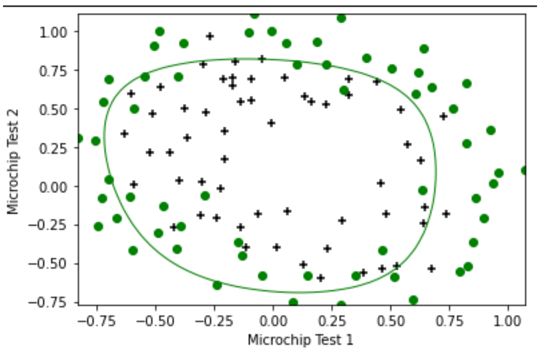

```
Porcentaje de aciertos: 0.8220338983050848
```

lamda = 1




```
Porcentaje de aciertos: 0.8305084745762712
```

lamda = 5

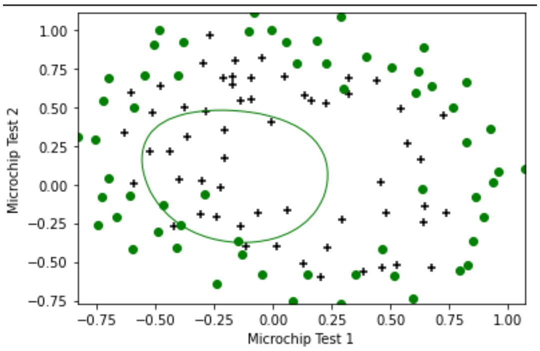



```
Porcentaje de aciertos: 0.8135593220338984
```

Con todos estos valores la frontera forma una elipse que recoge a la mayoria de los datos positivos de entrada sin ajustarse demasiado a ellos. Podemos ver tambien que todas tienen un porcentaje superior al 80%

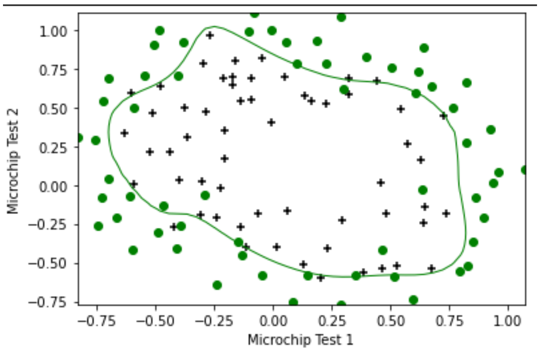
A continuacion decidimos probar valores mas extremos de lamda para ver como reaccionaba la frontera

lamda = 100




```
Porcentaje de aciertos: 0.6016949152542372
```

lamda = 0




```
Porcentaje de aciertos: 0.8728813559322034
```

Al aumentar lamda a 100 se ve como la frontera se reduce drasticamente y se aleja mucho de los datos de entrada produciendo una prediccion mucho menos fiable con tan solo un 60% de aciertos.

Al reducir lamda hasta 0 por el contrario se puede ver como el porcentaje de aciertos sube, pero al fijarnos en la grafica se ve claramente como la frontera se ha ajustado demasiado a los datos de entrenamiento. Creando un modelo que muy probablemente no funcione bien con datos nuevos.