Universidad Complutense de Madrid
Facultad de Informática
Aprendizaje Automático y Big Data

Memoria Práctica 2.

Profesor:
- Alberto Díaz Esteban.

Alumnos:
- Marina de la Cruz López.
- Diego Alejandro Rodríguez Pereira.

Introducción

En esta práctica vamos a resolver dos problemas de clasificación binaria usando regresión logística. El primero de ellos usando un *dataset* de exámenes de admisión de una universidad , el cuál va a ser implementado con regresión logística sin regularización. El segundo de ellos es un *dataset* de control de calidad de microchips, este será implementado utilizando regresión logística con regularización. En ambas versiones se usará descenso de gradiente para encontrar la función que determina la frontera entre las dos clases, posteriormente determinaremos lo bueno que han sido nuestros clasificadores contando el porcentaje de los ejemplos que han acertado para cada problema.

Parte 1: Regresión logística.

Para la primera parte de la práctica se ha utilizado como datos las notas obtenidas por una serie de candidatos en dos exámenes de admisión de una universidad, siendo el resultado de estos si fueron o no admitidos. Las notas son valores de entre 0 y 100, y el resultado se representa con un 0 si no ha sido admitido el estudiante y con un 1 si ha sido admitido.

A partir de este conjunto de datos construiremos un modelo por regresión logística sin regularización que calcule la probabilidad de que dado la nota de los dos exámenes se determine si un estudiante es admitido o no.

El primer paso para construir el modelo de regresión es visualizar los datos del conjunto de datos y a su vez determinar si hay que realizar alguna operación sobre los datos antes de utilizarlos.

Para la visualización de los datos se muestra la figura1. En la que pudimos observar donde se ubican cada uno de los puntos del conjunto de ejemplos, y a su vez poder distinguir en donde se encuentran cada una de las clases a clasificar. De esta manera se puede calcular aproximadamente para tener una idea de donde podría estar la frontera de decisión.

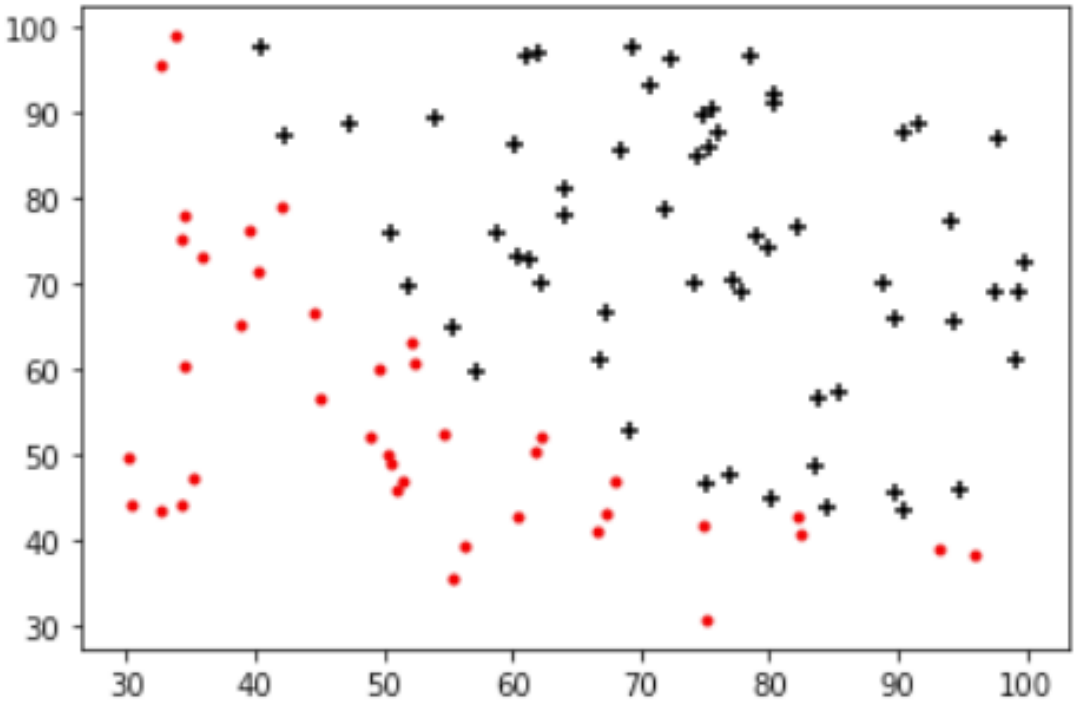


Figure 1

Para las operaciones sobre los datos: en este caso, como los datos se encuentran en la misma escala, dado que ambas variables son notas, no tenemos que normalizar los datos y por lo tanto podemos usarlos directamente en nuestro modelo. Simplemente se han separado los datos en una matriz X que contiene las dos primeras columnas del conjunto de datos (las dos variables) y en un vector Y que contiene la variable resultado. Después de separar los datos hemos añadido a la matriz X una columna en el primer índice inicializando todas sus filas con 1´s, para que de esta manera se puedan realizar los cálculos vectorizados como se ha realizado en las prácticas anteriores.

En segundo lugar, ya con los datos previamente preparados, empezamos a construir nuestro modelo de regresión. Comenzando con la implementación de la función sigmoide (*h(x)*) la cual se usa en la regresión logística para el cálculo del coste y del descenso de gradiente. El parámetro que recibe esta función es la matriz resultado de la multiplicación de las matrices *X*Theta* y luego calcula la función sigmoide para cada elemento de la matriz.

$$g(z) = \frac{1}{1 + e^{-z}}$$

Figure 2

Posteriormente implementamos la función de coste para la clasificación usando la ecuación vista en clase, como se muestra en la figura3. A su vez implementamos la función

de gradiente, que se muestra en la figura4, que es similar a la utilizada en la práctica anterior de regresión lineal pero utilizando como hipótesis *h(X)* la función sigmoide, esta devuelve un vector con los valores del gradiente. Ambas funciones reciben como parámetros un vector *theta*, el conjunto de entrenamiento pasado como matriz X y el vector de resultados Y.

$$J(\theta) = \frac{1}{m}((\ log\,(g(X\theta))^T y + (\ log\,(1 - g(X\theta))^T (1 - y))$$

Figure 3

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} X^T (g(X\theta) - y)$$

Figure 4

Luego de haber implementado la función sigmoide, la función de costes y la función de gradiente hemos pasado a realizar el cálculo del valor óptimo de los parámetros. Para la implementación de este cálculo se ha utilizado la función de *SciPy spicy.optimize.fmin_tnc.* De esta manera se puede obtener el valor de los parámetros del vector *theta* que minimizan la función de coste para la regresión logística implementada anteriormente.

La función *fmin_tnc* recibe como argumentos la función de coste, la función de gradiente, el vector de *Thetas* a minimizar, la matriz X y el vector Y. El vector de *Thetas* luego se utilizará como primer argumento de las funciones, y la X e Y como segundo y tercer argumento de estas funciones. El vector resultante contiene 3 elementos, que son la cantidad de columnas de la matriz X de entrada.

Resultados obtenidos.

En primer lugar se puede observar la gráfica obtenida del problema (figura5), en la que se puede constatar la función de la recta que separa las dos clases. A primera vista se puede ver que la recta obtenida ha sido capaz de separar correctamente ambas clases.

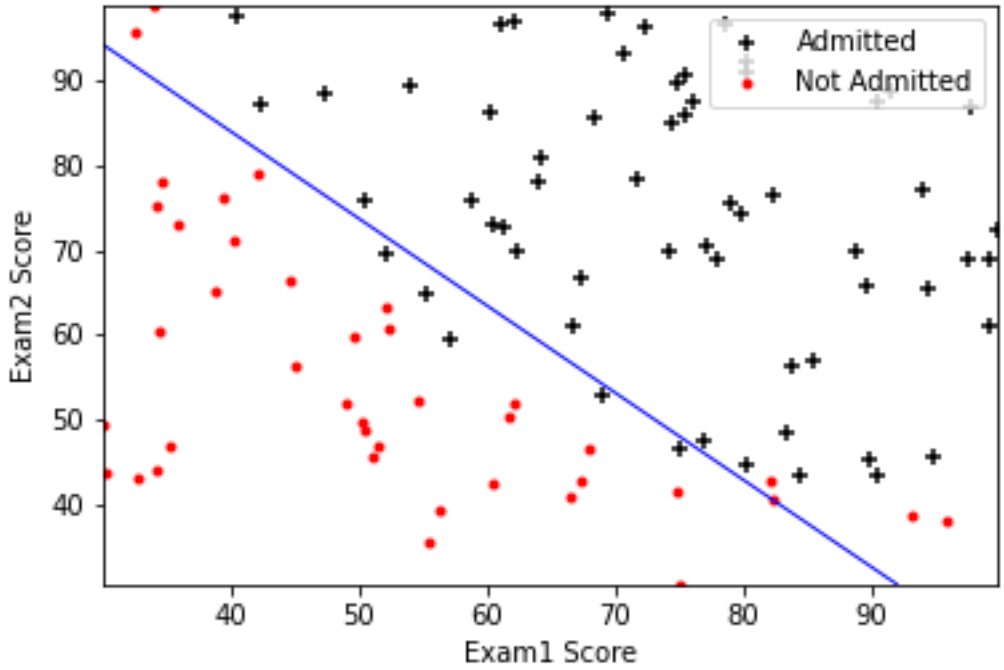


Figure 5

Las ***thetas* optimizadas** obtenidas son: **[-25.16131853 0.20623159 0.20147149].**

El resultado de la **función de coste** para estas *Thetas* es: **0.2034**.
Este resultado coincide con el proporcionado por el profesor.

Una vez obtenida la frontera de decisión y comprobado que la función de coste retorna el valor correcto de la solución podemos hacer una evaluación de nuestro modelo de regresión logística.

Para esto hemos implementado una función que calcula el porcentaje de ejemplos de entrenamiento que clasifican correctamente, la cual utiliza el vector *theta* calculado y mostrado anteriormente. Es decir, en nuestra gráfica, son la cantidad de puntos que han clasificado bien sobre los puntos mal clasificados. Para este cálculo se utiliza la función sigmoide sobre la multiplicación de la matriz de entrada X y los valores optimizados de *Theta*. Si el resultado de la función de sigmoide es mayor que 0.5 clasificamos al ejemplo como '1' o admitido, mientras que si es menor se clasifica como '0' o no admitido. Luego comparamos la clasificación obtenida con el valor real que se encuentra en el vector Y.

Usando este modelo hemos obtenido que se **han clasificado bien el 89%** de los ejemplos de entrenamiento.

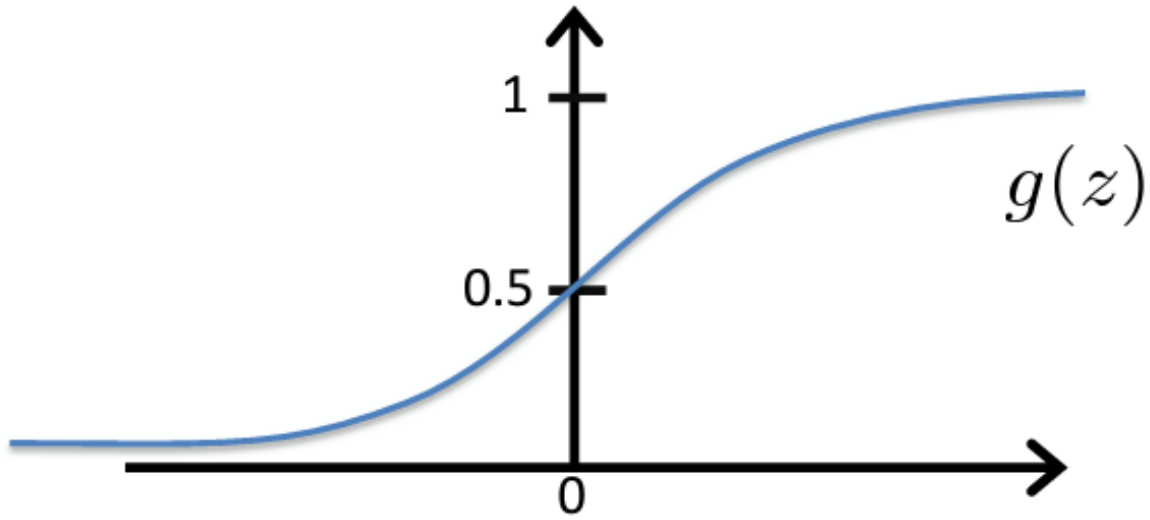


Figure 6

Parte 2: Regresión logística regularizada

En esta segunda parte de la práctica se ha utilizado un conjunto de datos de un control de calidad de unos microchips, en el que cada ejemplo incluye dos variables que son el resultado de dos test y una tercera variable que indica si el ejemplo ha pasado o no el control de calidad. En este apartado se utilizará el modelo de regresión logística regularizada para encontrar una función que prediga si un microchip dado pasará o no el control de calidad.

El primer paso para construir el modelo de regresión es visualizar los datos del conjunto de datos y a su vez determinar si hay que realizar alguna operación sobre los datos antes de utilizarlos.

Para la visualización de los datos se muestra la figura6. En la que pudimos observar donde se ubican cada uno de los puntos del conjunto de ejemplos, y a su vez poder distinguir en donde se encuentran cada una de las clases a clasificar. De esta manera se puede calcular aproximadamente para tener una idea de donde podría estar la frontera de decisión.

Al visualizar la figura6 podemos ver que las clases a clasificar no son linealmente separables por lo que vamos a necesitar combinar los datos de entrada para obtener una curva que pueda separar los datos correctamente.

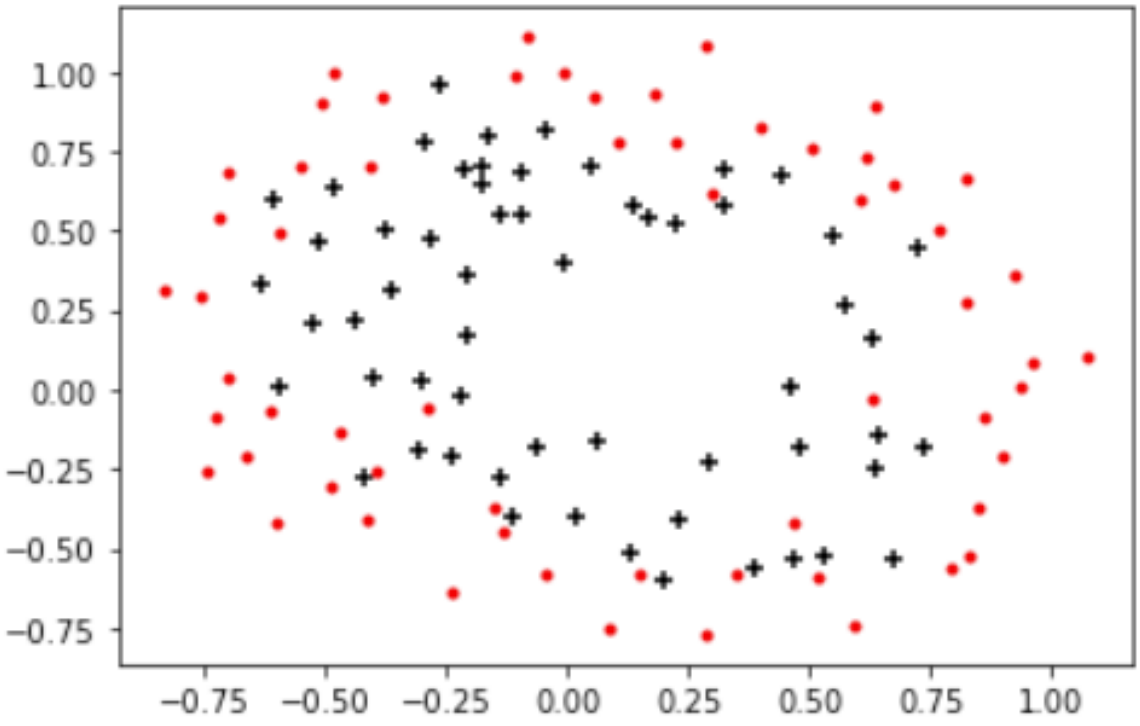


Figure 7

En segundo lugar, como en el apartado anterior, leemos el fichero que contiene el conjunto de datos y se separan los datos en una matriz X que contiene las dos primeras columnas del conjunto de datos (las dos variables) y en un vector Y que contiene la variable resultado (clase).

En este problema, como los datos se encuentran en la misma escala, dado que ambas variables son los valores de los test, no tenemos que normalizar los datos y por lo tanto podemos usarlos directamente para realizar el descenso de gradiente.

Una vez realizado esto, vamos a generar combinaciones con la *x1* y *x2* de la entrada para obtener un mejor ajuste, para ello hacemos las combinaciones de los atributos hasta la sexta potencia usando *PolynomialFeatures(6)*. Obtenemos un matriz de 28 columnas (*poly*), la primera de las cuales son unos (por lo que no tendremos que añadirlos nosotros a mano porque se genera con el *x1^0*x2^0*).

Generar estas combinaciones supone que sea más fácil ajustar la barrera de decisión pero puede resultar en un sobreajuste donde nos pegamos demasiado a los datos y no somos capaces de generalizar la tendencia que sigue la función, para evitar esto añadiremos un término extra de regularización a las *Thetas*.

La función de coste se mantiene igual a la usada en el apartado anterior de regresión logística añadiendo el sumatorio con la regularización de las *Thetas* sobre lambda, determinada por el programador, menos por *Theta0*, la cual no se regulariza.

La función de gradiente también es la misma a excepción de para cada *Theta* hay que añadir el término de regularización de esa *Theta*, excepto para *Theta0*. Para poder realizar los cálculos de forma vectorizada le pasamos a la función que calcula el vector de términos de

regularización el array a partir de *Theta1* y nos devuelve el array con los términos de regularización según la fórmula y con un 0 insertado en la posición 0.

Posteriormente se añade el resultado de la función de gradiente del apartado 1 con este vector y con esto obtenemos el gradiente de la *Thetas* regularizado.

El óptimo se calcula usando la función *opt.fmin_tnc* a la que le volvemos a pasar la función de coste y de descenso de gradiente, el vector de *Thetas* iniciales, y el resto de parámetros que usan nuestras funciones incluyendo la *X_poly*, la Y y la lambda para calcular la regularización.

El resultado será el array de 28 elementos con las *Thetas* optimizadas que minimizan la función de coste.

Resultados obtenidos

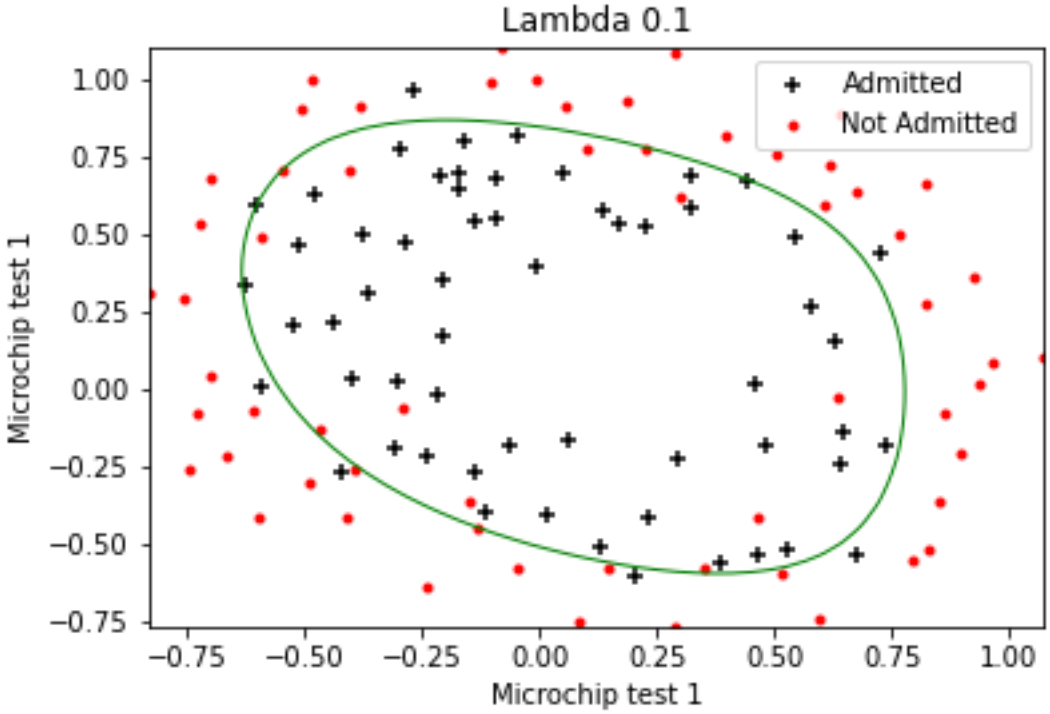
Los resultados obtenidos van a depender del valor de lambda que escojamos para regularizar, por ejemplo, para lambda= 1 el array de *Thetas* resultante es:

```
[ 1.27271029  0.62529965  1.18111687 -2.01987402 -0.9174319  -1.43166932
  0.12393226 -0.36553118 -0.35725405 -0.17516293 -1.4581701  -0.05098418
 -0.61558557 -0.27469165 -1.19271297 -0.2421784  -0.20603302 -0.04466177
 -0.27778948 -0.29539514 -0.45645982 -1.04319155  0.02779373 -0.29244867
  0.01555759 -0.32742405 -0.1438915  -0.92467487]
```

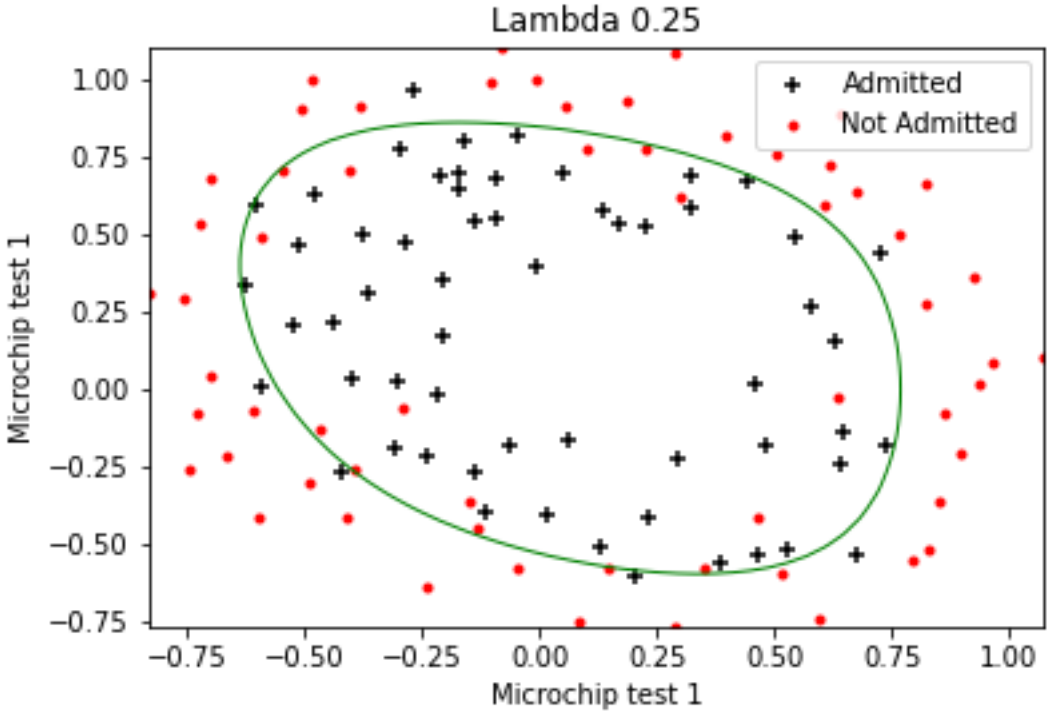
Con estos resultados podemos visualizar la barrera de decisión, así como calcular el porcentaje de aciertos usando la sigmoide de la multiplicación de *X_poli*Thetas*, de la misma manera que para la primera parte.

Hemos probado a hacer esto con distintos valores de lambda para ver como cambia la barrera y el porcentaje de aciertos sobre el conjunto de entrenamiento.

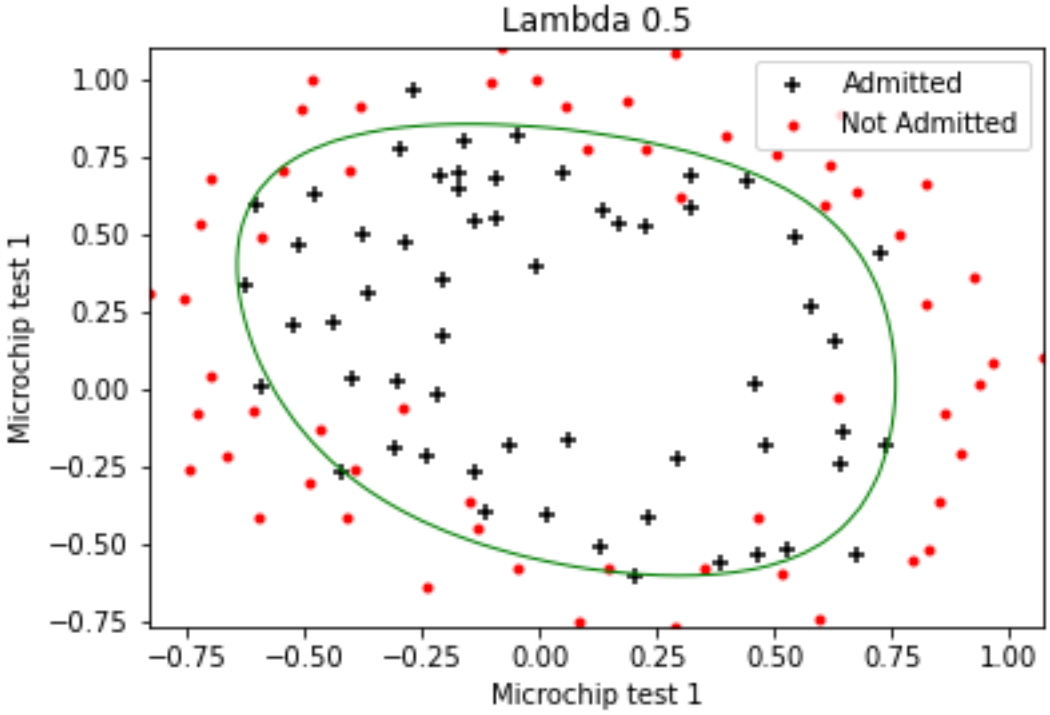
Lambda: 0.1
Porcentaje aciertos: 83.89830508474576



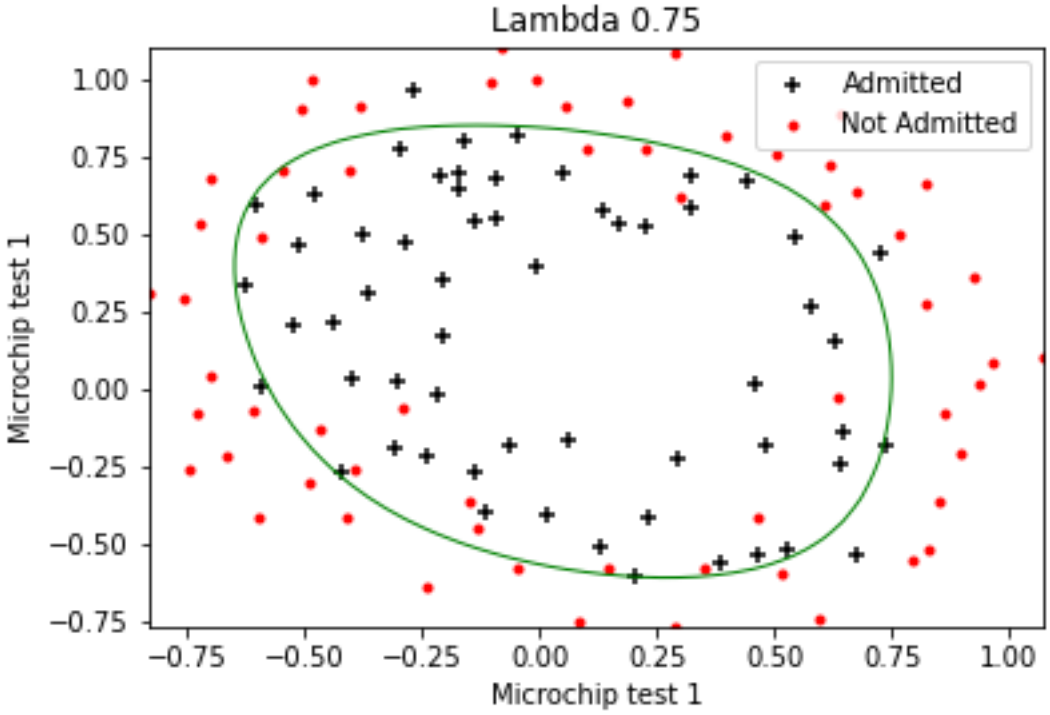
Lambda: 0.25
Porcentaje aciertos: 83.05084745762711



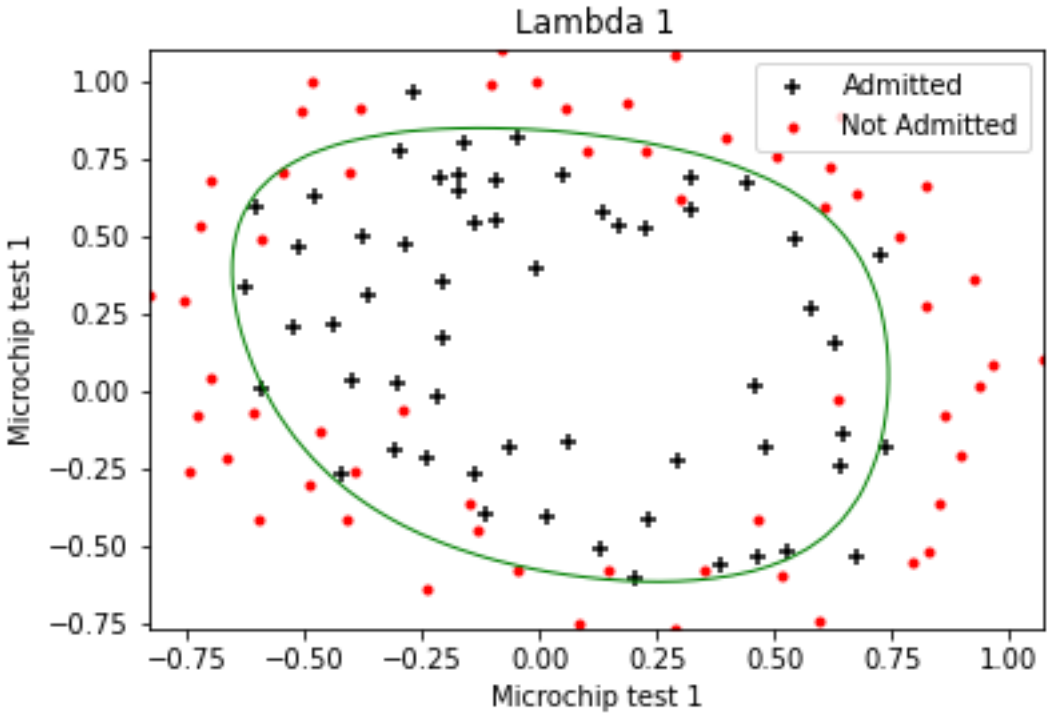
Lambda: 0.5
Porcentaje aciertos: 82.20338983050848



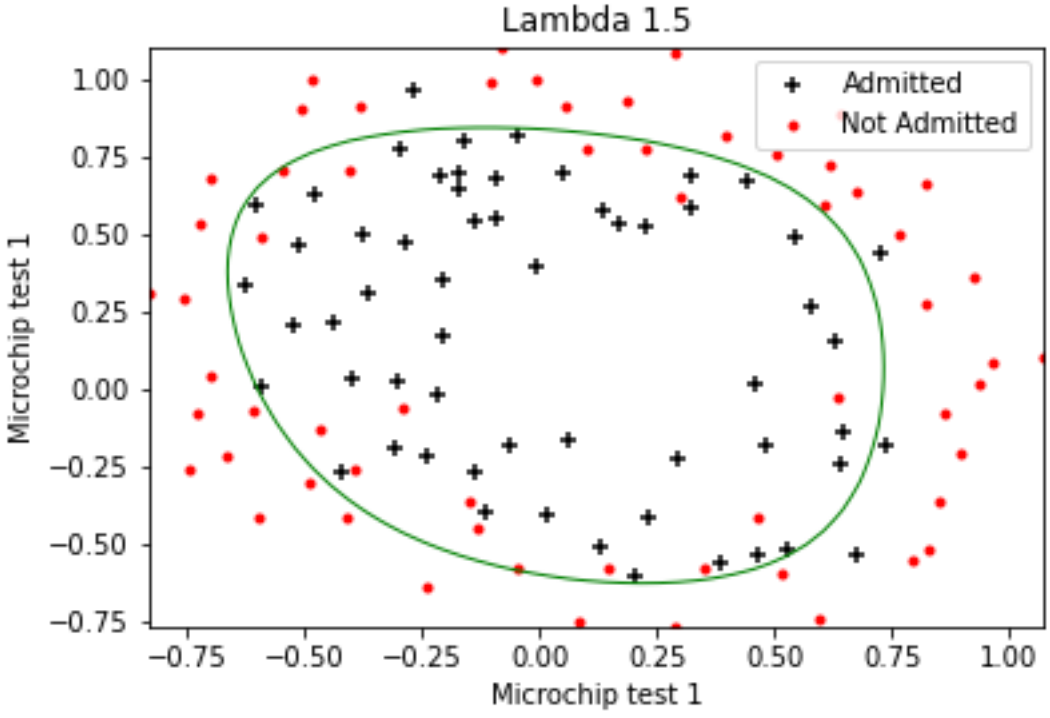
Lambda: 0.75
Porcentaje aciertos: 83.05084745762711



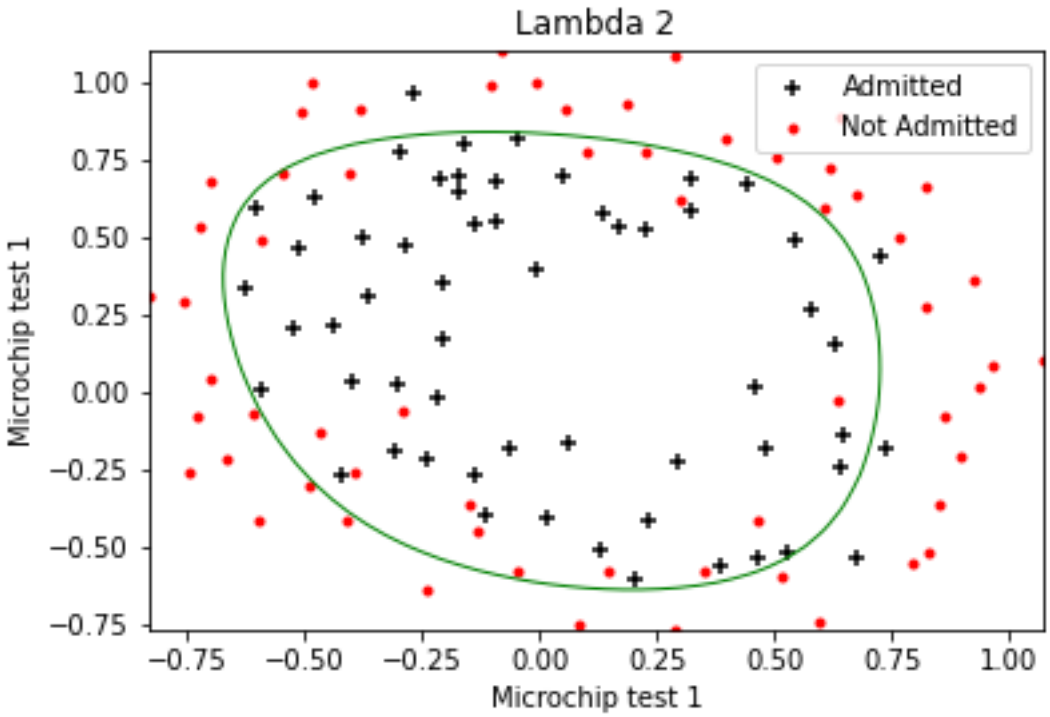
Lambda: 1
Porcentaje aciertos: 83.05084745762711



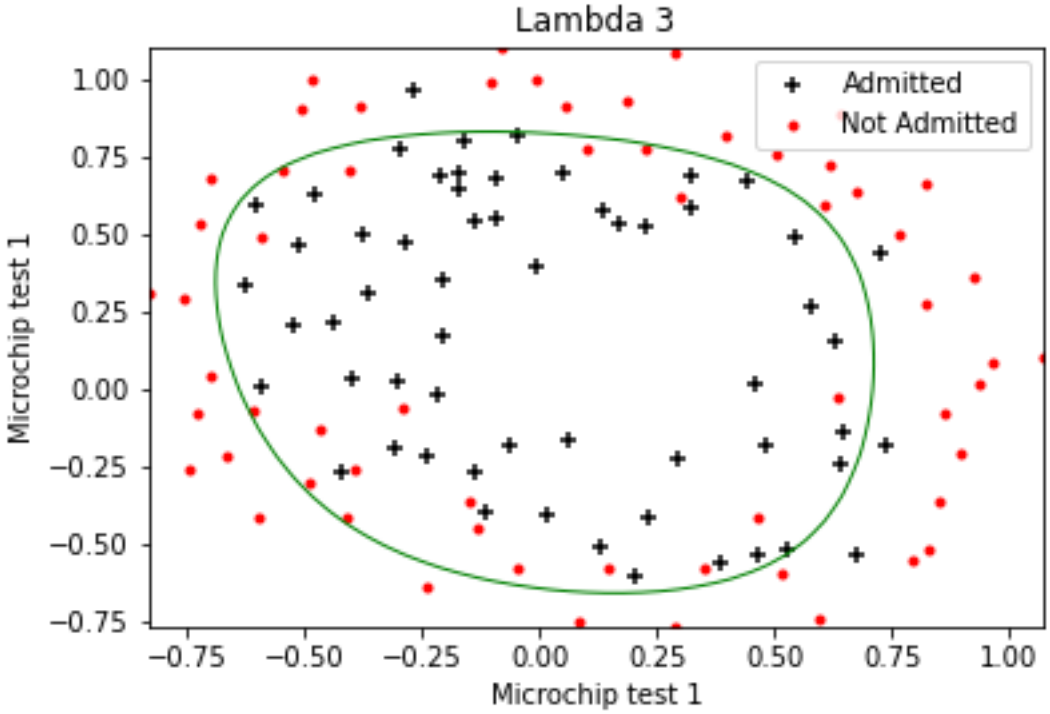
Lambda: 1.5
Porcentaje aciertos: 83.05084745762711



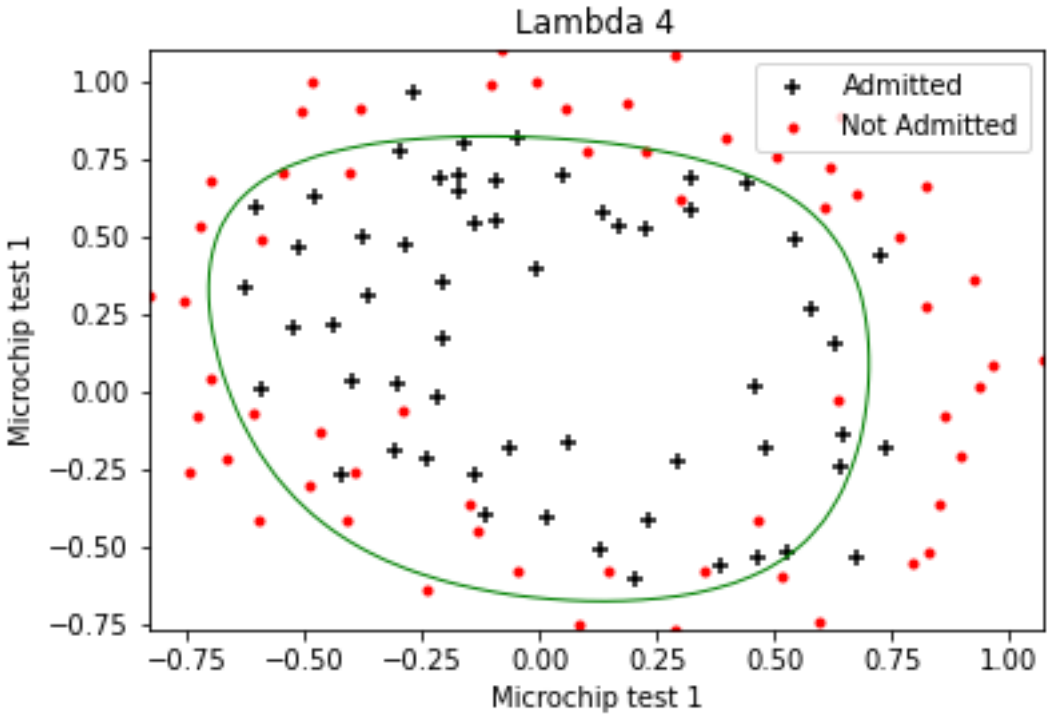
Lambda: 2
Porcentaje aciertos: 83.05084745762711



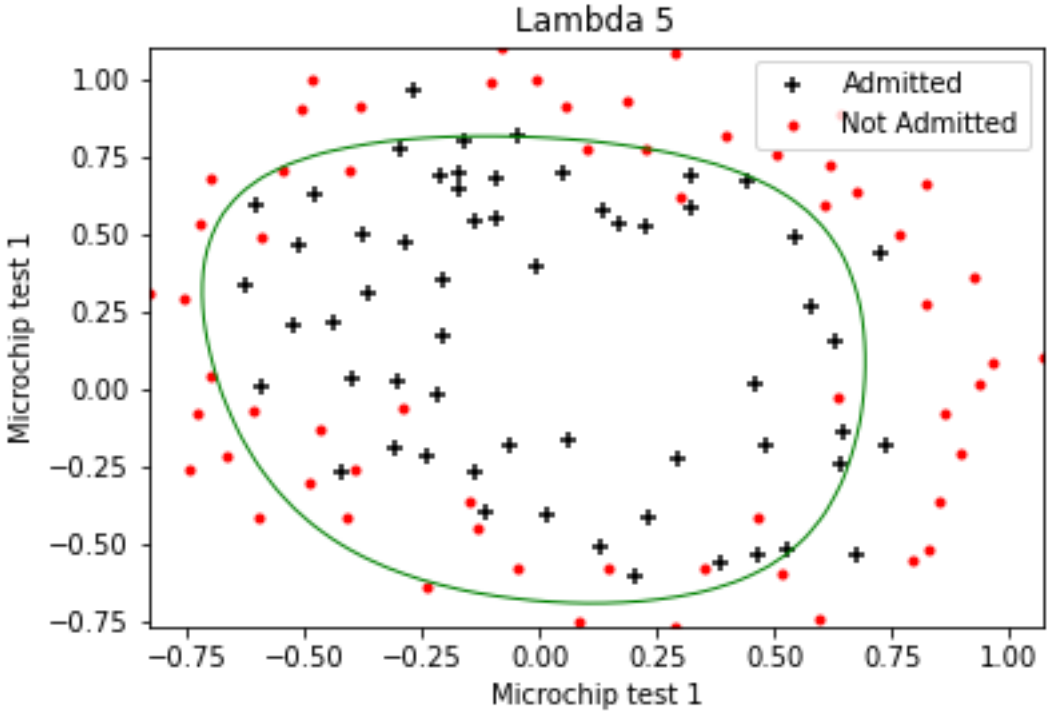
Lambda: 3
Porcentaje aciertos: 80.50847457627118



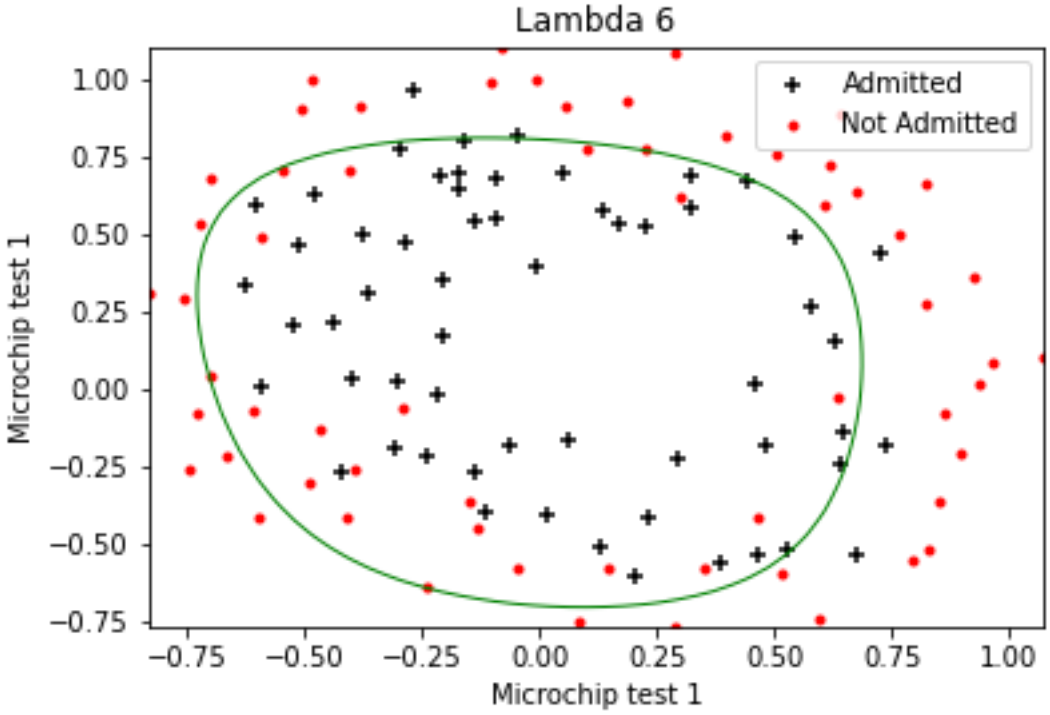
Lambda: 4
Porcentaje aciertos: 80.50847457627118



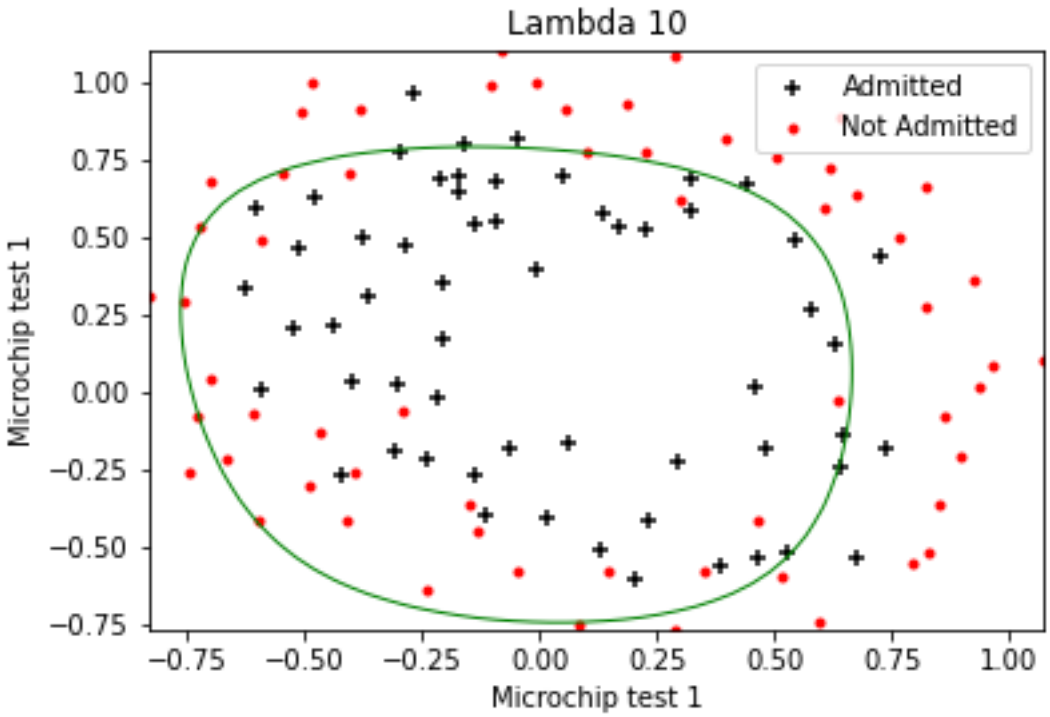
Lambda: 5
Porcentaje aciertos: 81.35593220338984



Lambda: 6
Porcentaje aciertos: 78.8135593220339



Lambda: 10
Porcentaje aciertos: 74.57627118644068



De estas pruebas con distintas lambdas podemos observar que cuanto más alta, más regulariza y menos se pega a los datos de entrenamiento por lo que la tendencia es que de peores resultados en el accuracy con los datos de entrenamiento.

En general, para poder elegir correctamente el valor de lambda necesitaremos buscar un valor que generalice lo suficiente pero tratando de obtener el mejor valor de accuracy posible. Para ello, la mejor opción es realizar pruebas sobre conjuntos de test (disjuntos al de entrenamiento), sacar el accuracy para cada uno de ellos y en base a los resultados elegir el valor que mejor clasifique.

Para este problema consideramos que visualmente, sobre los ejemplos de entrenamiento, las mejores lambdas son las que se encuentran entre 0.5 y 1.5.

Código

```
#Imports
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from matplotlib.axes import Axes
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import math
import scipy.optimize as opt
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
#Funcion que carga los datos
def load_csv(filename):
    valores = read_csv (filename, header=None).to_numpy()
    return valores.astype(float)
```

```
# 1. Regresión Logística

Los datos del fichero ex2data1.csv representan las notas obtenidas por una seria de
candidatos en los dos exámenes de admisión de una universidad junto con la
información sobre si fueron (1) o no (0) admitidos.
El objetivo es construir un modelo por regresión logística que estime la
probabilidad de que un estudiante sea admitido en esa universidad en base a las
notas de sus exámenes.
```

```
data = load_csv('ex2data1.csv')

#Obtencion de todas las columnas de la tabla menos la ultima columna
X = data[:, :-1]
np.shape(X)
#Obtencion de la ultima columna de la tabla
Y = data[:, -1]
np.shape(Y)
```

```
def mostrar_data(Y, X):
    # Obtiene un vector con los indices de los ejemplos positivos
    pos = np.where(Y == 1)
    # Obtiene un vector con los indices de los ejemplos positivos
    pos2 = np.where(Y == 0)

    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos2, 0], X[pos2, 1], marker='.', c='r')
```

```
mostrar_data(Y, X)
```

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

```
def cost_funct(Theta, X, Y):
    #g(X*Theta)
    g = sigmoid_funct(np.matmul(X, Theta))
    #g = sigmoid_funct(X * Theta)
    m = np.shape(X)[0]

    J = (np.matmul(np.transpose(np.log(g)), Y)) + (np.matmul(np.transpose(np.log(1-
g)), (1 - Y)))
```

```
    return np.sum(-J)/m
```

```
def gradient(Theta, X, Y):
    m = np.shape(X)[0]
    g = sigmoid_funct(np.matmul(X,Theta))
    J = np.dot(np.transpose(X), (g - Y))
    return J/m
```

```
#Obtenido del código ejemplo del profesor
def pinta_frontera_recta(X, Y, Theta):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    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 = sigmoid_funct(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
    aux(fig, ax)
    #plt.savefig("fronteir.png")
    #plt.close()
```

```
def porcentaje_aciertos(Theta, X, Y):
    # Calculamos los valores estimados segun la theta que hemos obtenido
    sigmoid = sigmoid_funct(np.matmul(X, Theta))

    # Se compara la estimacion nuestra con el resultado real (Y).
    # Se devuelve el numeros de ejemplos que se han estimado
    # correctamente, es decir, para aquellos que tengan
    # el resultado del sigmoide mayor o igual a 0.5
    evaluation_correct = np.sum((sigmoid >= 0.5) == Y)

    # Devolvemos el porcentaje
    return evaluation_correct/len(sigmoid) * 100
```

```
def aux(fig, ax):

    data = load_csv('ex2data1.csv')
    #Obtencion de todas las columnas de la tabla menos la ultima columna
    X = data[:, :-1]
    np.shape(X)
```

```
    #Obtencion de la ultima columna de la tabla
    Y = data[:, -1]
    np.shape(Y)
    # Obtiene un vector con los indices de los ejemplos positivos
    pos = np.where(Y == 1)
    # Obtiene un vector con los indices de los ejemplos positivos
    pos2 = np.where(Y == 0)

    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label = "Admitted")
    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos2, 0], X[pos2, 1], marker='.', c='r', label = "Not Admitted")

    # Anadimos leyenda a la grafica y la posicionamos bien
    plt.legend(loc = 'upper right')

    # Anadimos el texto a los ejes (como en la grafica que aparece de ejemplo)
    ax.set_xlabel('Exam1 Score', fontsize=10)
    ax.set_ylabel('Exam2 Score', fontsize=10)
```

```
def logistic_regresion():
    data = load_csv('ex2data1.csv')
    #Obtencion de todas las columnas de la tabla menos la ultima columna
    X = data[:, :-1]
    m = np.shape(X)[0]
    # Aregamos una columna de 1s
    X = np.hstack([np.ones([m, 1]), X])
    n = np.shape(X)[1]
    #Obtencion de la ultima columna de la tabla
    Y = data[:, -1]

    Theta = np.zeros(n)

    result = opt.fmin_tnc(func=cost_funct, x0=Theta, fprime=gradient, args=(X,Y),
messages=0)
    theta_opt = result[0]

    print("theta optimizada:", theta_opt)
    print("Coste final:" , cost_funct(theta_opt, X,Y))
    pinta_frontera_recta(X, Y, theta_opt)
    #aux()
    porcentaje =  porcentaje_aciertos(theta_opt, X, Y)
    print(porcentaje)
```

```
logistic_regresion()
```

```
data = load_csv('ex2data2.csv')

#Obtencion de todas las columnas de la tabla menos la ultima columna
X = data[:, :-1]
```

```
#Obtencion de la ultima columna de la tabla
Y = data[:, -1]
mostrar_data(Y, X)
```

```
def func_coste_reg(Thetas, X, Y, lmb, m):
    return cost_funct(Thetas, X, Y) + regularizacion(Thetas[1:], lmb, m)
```

```
def regularizacion(Thetas, lmb, m):
    return (lmb/(2*m))*np.sum(Thetas**2)
```

```
def func_grad_reg(Thetas, X, Y, lmb, m):
    return np.add(gradient(Thetas, X, Y) , reg_grad(Thetas[1:], lmb, m))
```

```
def reg_grad(Thetas, lmb, m):
        return np.insert(lmb/m*Thetas, 0, values=[0])
```

```
def plot_decisionboundary(X, Y, theta, poly, i):
    fig= plt.figure()

    X = data[:, :-1]
    np.shape(X)
    #Obtencion de la ultima columna de la tabla
    Y = data[:, -1]
    np.shape(Y)
    # Obtiene un vector con los indices de los ejemplos positivos
    pos = np.where(Y == 1)
    # Obtiene un vector con los indices de los ejemplos positivos
    pos2 = np.where(Y == 0)

    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label = "Admitted")
    # Dibujo de los ejemplos positivos
    plt.scatter(X[pos2, 0], X[pos2, 1], marker='.', c='r', label = "Not Admitted")

    # Anadimos leyenda a la grafica y la posicionamos bien
    plt.legend(loc = 'upper right')

    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_funct(poly.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')

    plt.xlabel("Microchip test 1")
    plt.ylabel("Microchip test 1")
```

```
    plt.title("Lambda "+ str(i))

    #plt.savefig("boundary"+str(i)+".png")
    plt.show()
    plt.close()
```

```
data = load_csv('ex2data2.csv')

#Obtencion de todas las columnas de la tabla menos la ultima columna
X = data[:, :-1]

m = np.shape(X)[1]

#Obtencion de la ultima columna de la tabla
Y = data[:, -1]

poly = PolynomialFeatures(6)
datapoly = poly.fit_transform(X)

m = np.shape(datapoly)[0]
n = np.shape(datapoly)[1]
print(m)

#Inicializamos las Thetas a 0
Thetas = np.zeros(n)

lmb = [0.1,0.25, 0.5,0.75,1,1.5, 2,3,4,5,6,7,10]
#lmb = [0.01,0.025, 0.05, 0.075]


#Para cada lambda calculamos las Thetas óptimas, pintamos la función y sacamos el
porcentaje de aciertos
for i in lmb:
    result = opt.fmin_tnc(func=func_coste_reg, x0=Thetas, fprime=func_grad_reg,
args=(datapoly,Y,i,m), messages=0)
    print("Lambda: " , i)
    print(result[0])
    porcentaje =  porcentaje_aciertos(result[0], datapoly, Y)
    print("Porcentaje aciertos: " + str(porcentaje))
    plot_decisionboundary(X,Y, result[0], poly, i)
```