Práctica 2: Regresión Logística

Álvar Domingo Fernández y Pablo Jurado López

Preparación inicial

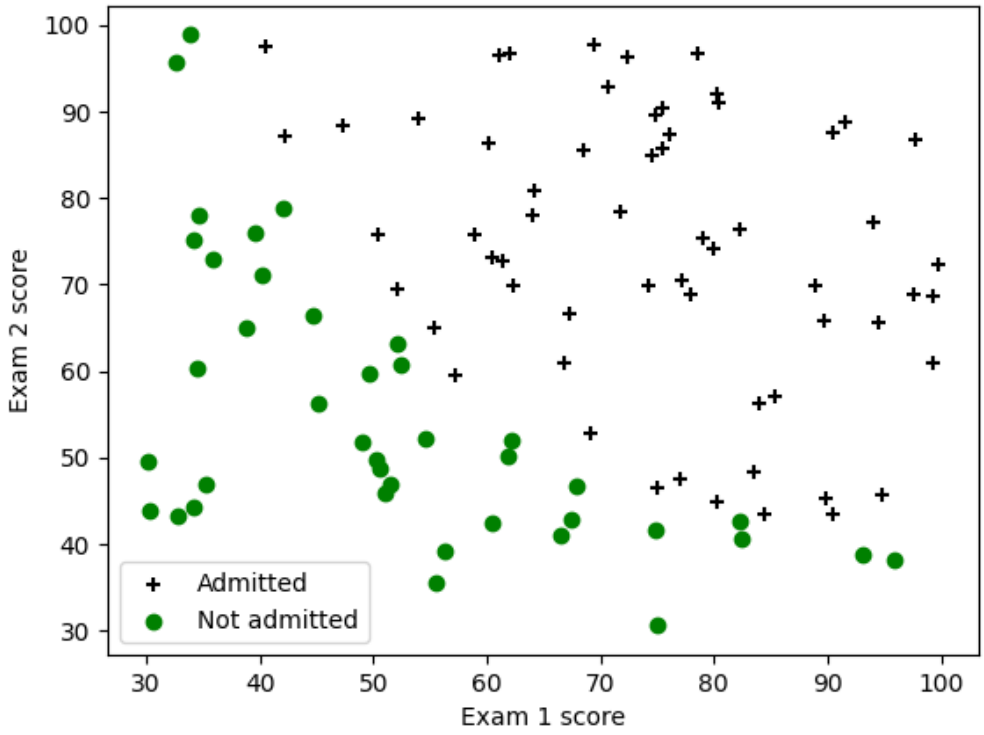
A continuación se importan todas las librerías que serán utilizadas en esta práctica y se indica el método que se utilizará para cargar los datos:

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


def carga_csv(file_name):
    return read_csv(file_name, header=None).to_numpy().astype(float)
```

1.1 - Visualización de los datos

A partir de un fichero con los datos, hemos utilizado la libreria matplot para representarlos en una gráfica, representando de distinta forma los puntos que representan ser admitido o no admitido.

[ ]:
```
data = carga_csv('ex2data1.csv')
X = data[:, :-1]
Y = data[:, -1]
# Obtiene un vector con los índices de los ejemplos positivos
pos = np.where(Y == 1)
neg = np.where(Y == 0)

# Dibuja los ejemplos positivos
plt.figure(0)
plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label='Admitted')
plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g', label='Not admitted')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()
plt.savefig('dataGraph1')
```



1.2 - Función sigmoide

Se ha implementado una función sigmoide, definida por la siguiente fórmula:

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

```
[ ]: def sigmoide(target):
         result = 1 / (1 + np.exp(-target))
         return result
```

1.3 - Cálculo de la función de coste y su gradiente

Se ha implementado la función de coste en regresión logística, que en forma vectorizada viene dada por la siguiente fórmula:

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

```
[ ]: def coste(theta, X, Y):
         m = np.shape(X)[0]
         H = sigmoide(np.matmul(X, theta))
```

```
    return (np.dot(np.transpose(np.log(H)), Y) + np.dot(np.transpose(np.
↪log(1-H)), (1-Y))) / -m
```

También se ha implementado la función que obtiene el gradiente de la función de coste, que viene definida en su forma vectorizada por la siguiente fórmula:

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

[ ]:
```
def gradiente(theta, X, Y):
    m = np.shape(X)[0]
    H = sigmoide(np.matmul(X, theta))
    return (np.matmul(X.T, H - Y)) / m
```

1.4 - Cálculo del valor óptimo de los parámetros

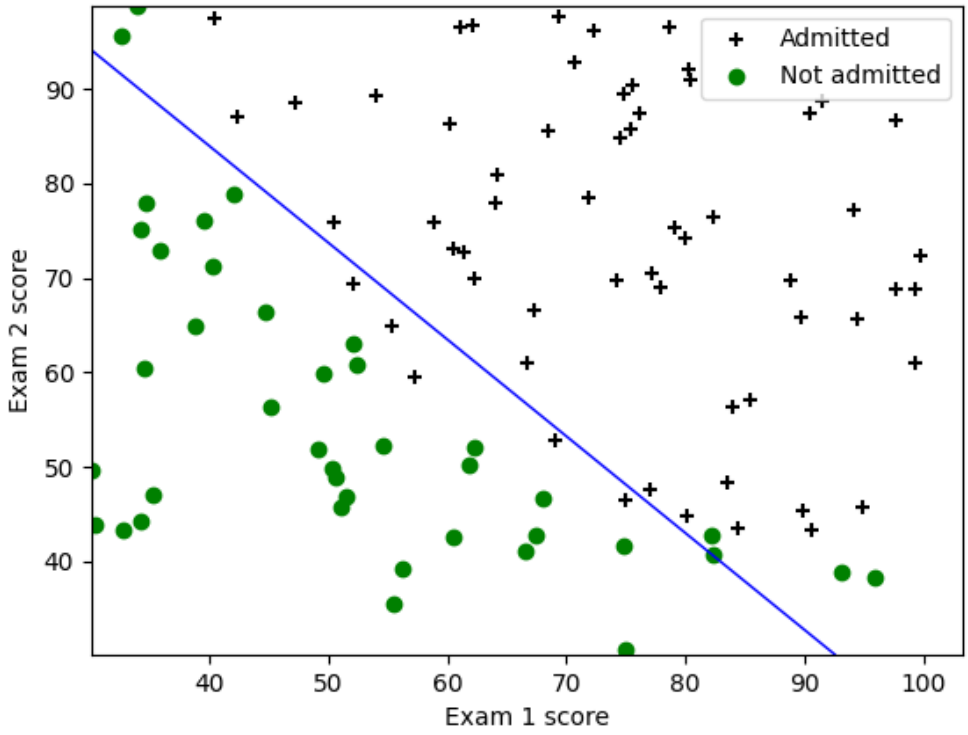
Se ha utilizado la función scipy.optimize.fmin_tnc de SciPy para hallar los parámetros θ que minimizan la función de coste para la regresión del apartado anterior:

[ ]:
```
result = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente, args=(X, Y))
theta_opt = result[0]
print(coste(theta_opt, X, Y))
```

El valor de la función de coste ha sido en este caso de aproximadamente 0.69

A continuación se han representado los resultados en una gráfica que, gracias a este cálculo, también dibuja la frontera de decisión:

[ ]:
```
# Dibuja los ejemplos positivos
plt.figure(0)
x1_min, x1_max = X[:, 1].min(), X[:, 1].max()
x2_min, x2_max = X[:, 1].min(), X[:, 2].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_opt))
h = h.reshape(xx1.shape)

plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
plt.savefig('dataGraph1line')
plt.close()
```



1.5 - Evaluación de la regresión logística

Se ha implementado una función que da como resultado el porcentaje de los casos de entrenamiento que se han clasificado de manera correcta, es decir, aquellos que han quedado en el lado correcto de la frontera de decisión en la imagen anterior:

[ ]:
```
prediccion = sigmoide(np.matmul(X, theta_opt))
correctos = np.mean((prediccion >= 0.5) == Y)
print(correctos)
```

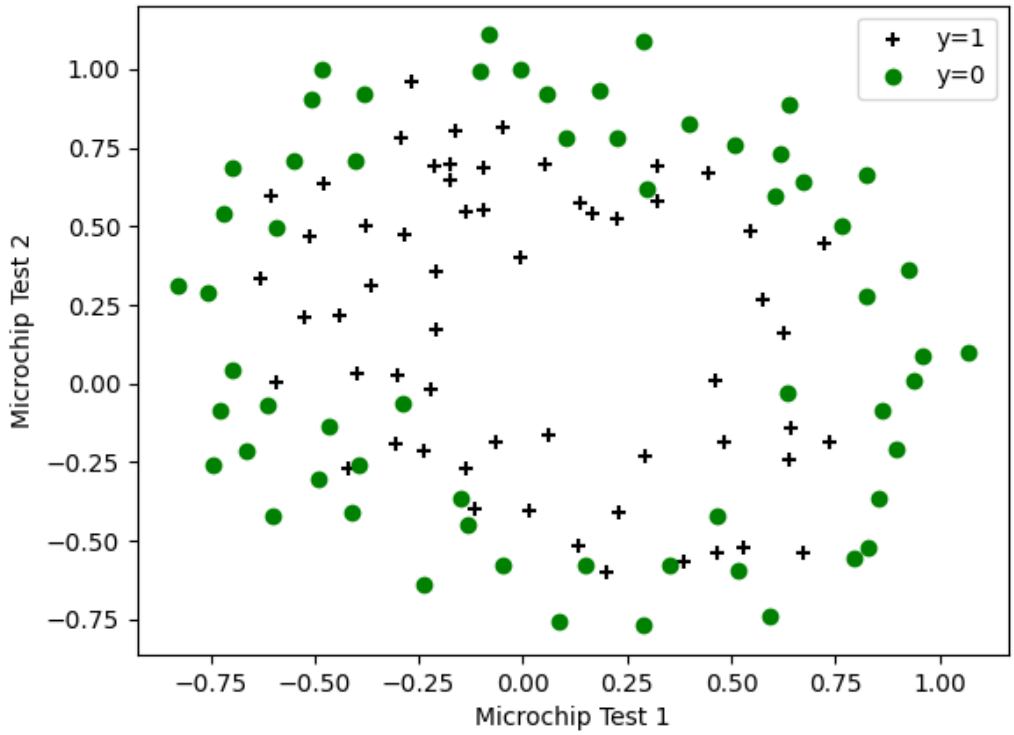
El resultado en este caso ha sido de un 89%

2 - Regresión logística regularizada

Antes de nada, se han visualizado los datos del nuevo conjunto en una gráfica para comprobar que, a diferencia de los datos anteriores, estos no son linealmente separables.

[ ]:
```
data = carga_csv('ex2data2.csv')
X = data[:, :-1]
Y = data[:, -1]
# Obtiene un vector con los índices de los ejemplos positivos
pos = np.where(Y == 1)
neg = np.where(Y == 0)
```

```
# Dibuja los ejemplos positivos
plt.figure(0)
plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k', label='y=1')
plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g', label='y=0')
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')
plt.legend()
plt.savefig('dataGraph2')
```



2.1 - Mapeo de los atributos

Con la clase PolynomialFeatures de sklearn se han añadido nuevos atributos a los ejemplos de entrenamiento. Se ha calculado también la matriz de thetas que utilizaremos para el coste y el gradiente a partir de la matriz X extendida.

[ ]:
```
poly = PolynomialFeatures(6)
Xp = poly.fit_transform(X)
thetas = np.zeros(np.shape(Xp)[1])
```

2.2 - Calculo de la función de coste y su gradiente

A continuación se ha calculado la función de coste para la versión regularizada de la regresión logística, que en su forma vectorial viene definida por la siguiente fórmula:

$$J(\theta) = -\frac{1}{m}\left((\log(g(X\theta))^T y + (\log(1 - g(X\theta))^T(1-y)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

[ ]:
```
def gradienteReg(thetas, x, y, lamb):
    sigXT = sigmoide(np.matmul(x, thetas))
    return ((1/np.shape(x)[0]) * np.matmul(np.transpose(x), (sigXT - y))) +␣
 ↪((lamb/np.shape(x)[0]) * thetas)
```

Asimismo se ha implementado también la función de coste, siguiendo la fórmula matemática que permite calcularla de manera vectorizada:

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

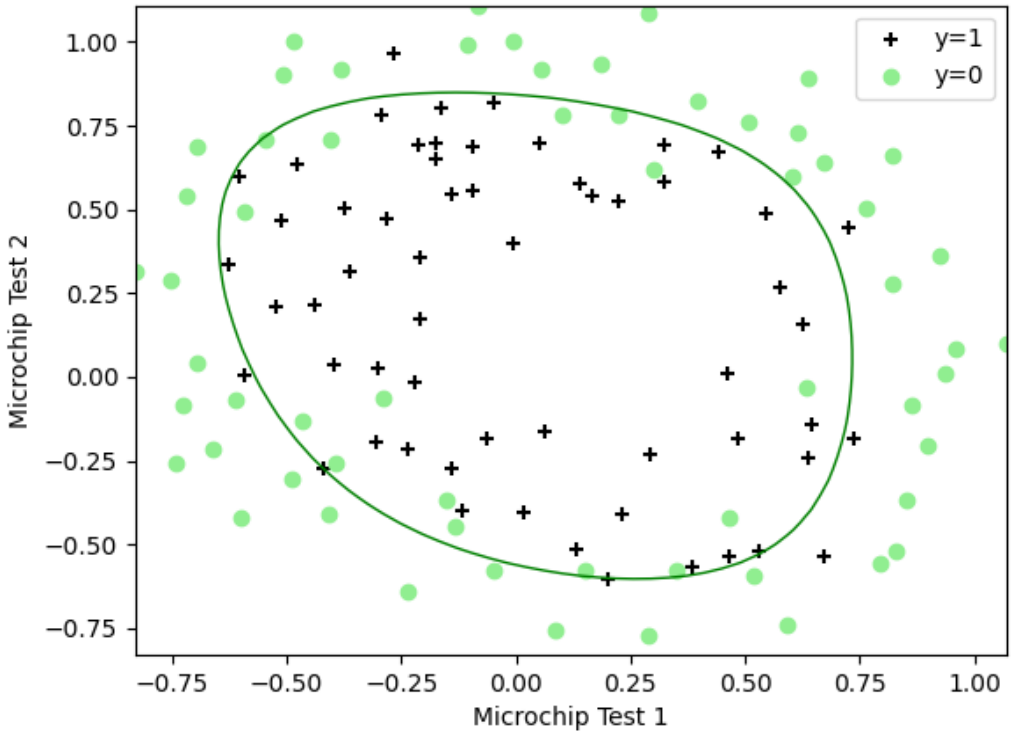
[ ]:
```
def costeReg(thetas, x, y, lamb):
    sigXT = sigmoide(np.matmul(x, thetas))
    return (-1/np.shape(x)[0]) * (np.matmul(np.transpose(np.log(sigXT)), y) + np.
 ↪matmul(np.transpose(np.log(1-sigXT)), (1-y))) + ((lamb/(2*np.shape(x)[0])) *␣
 ↪sum(thetas ** 2))
```

2.3 - Cálculo del valor óptimo de los parámetros

Se ha vuelto a hacer uso de la función scipy.optimize.fmin_tnc para hallar el valor óptimo de los parámetros, esta vez para la versión regularizada de la regresión, mediante el siguiente fragmento de código:

[ ]:
```
result = opt.fmin_tnc(func=costeReg, x0=thetas, fprime=gradienteReg, args=(Xp,␣
 ↪Y, 1))
theta_opt = result[0]
print(theta_opt)
```

También se ha representado gráficamente la frontera de decisión:

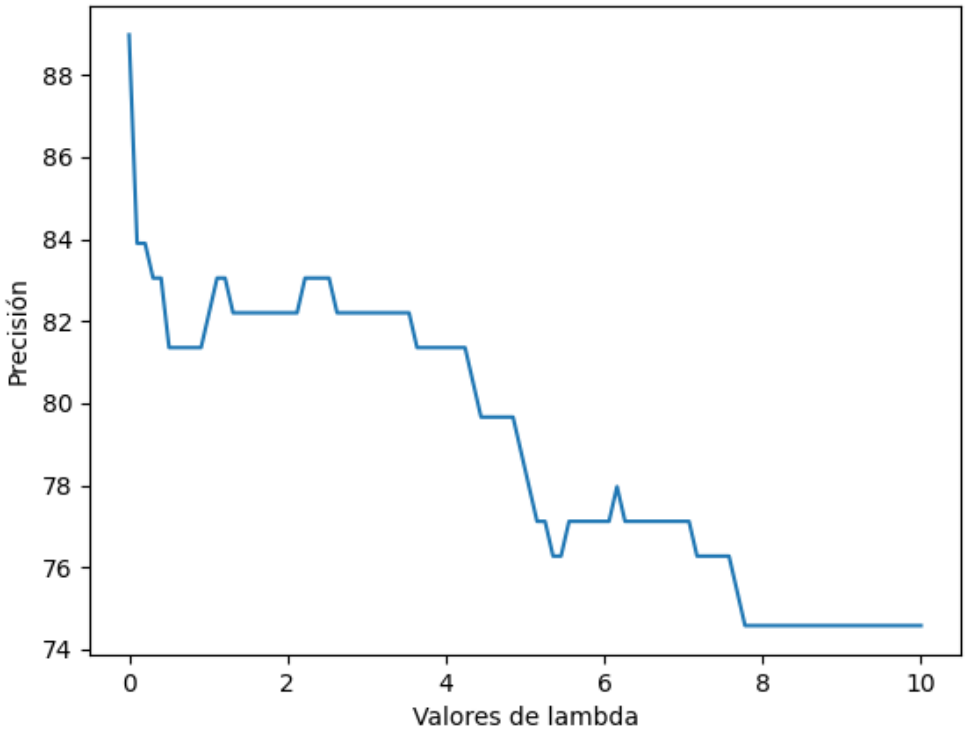


2.4 - Efectos de la regularización

Finalmente, se ha experimentado con varios valores de λ para ver cómo afectaría la variación a la precisión del resultado del aprendizaje. Se ha utilizado el siguiente fragmento de código:

[ ]:
```
def efectosReg(Xp, Y, thetas):
    accuracy = []
    lambdas = np.linspace(0, 10, 100)
    for i in range(np.shape(lambdas)[0]):
        theta_opt = opt.fmin_tnc(
            func=costeReg, x0=thetas, fprime=gradienteReg, args=(Xp, Y,␣
↪lambdas[i]))[0]
        accuracy.append(
            np.mean((sigmoide(np.matmul(Xp, theta_opt)) >= 0.5) == Y) * 100)

    plt.figure()
    plt.xlabel("Valores de lambda")
    plt.ylabel("Precisión")
    plt.plot(lambdas, accuracy)
    plt.savefig("Regularization")
```

Finalmente se ha representado la evolución de dicha precisión en la siguiente gráfica:


Precisión
88
86
84
82
80
78
76
74
0
2
4
6
8
10
Valores de lambda