Practica 2: Regresión Logística

Nuria Bango Iglesias (nubango@ucm.es)

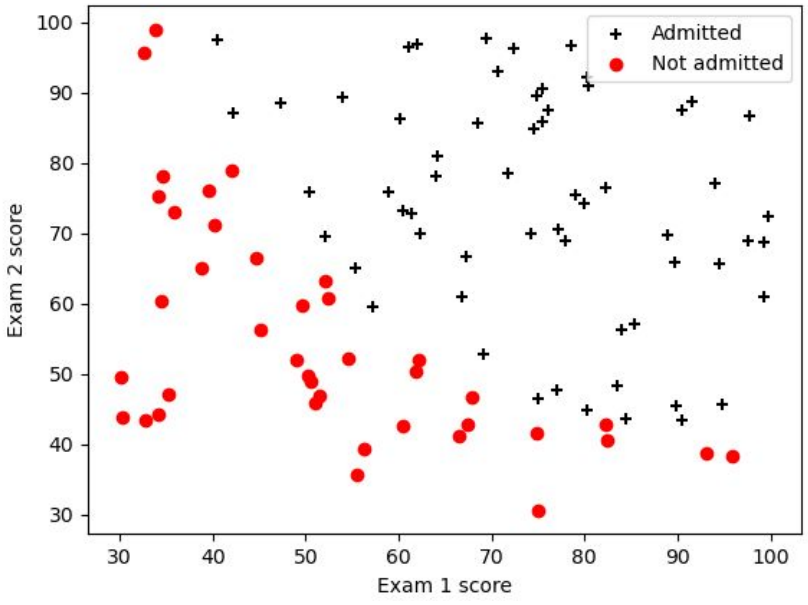
Álvar Julián de Diego López (alvarded@ucm.es)

1.Regresión logística

1.1Visualización de los datos

Lo primero que vamos a hacer es cargar y visualizar el contenido del fichero ex2data1.csv, para ello utilizamos el siguiente código que genera la gráfica expuesta a continuación.

```
data = carga_csv("ex2data1.csv")
X = data[:, :-1]
Y = data[:, -1]

pos = np.where(Y == 1)
neg = np.where(Y == 0)

plt.scatter(X[pos, 0], X[pos, 1], marker='+', c = 'k')
plt.scatter(X[neg, 0], X[neg, 1], marker='o', c = 'r')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend(('Admitted', 'Not admitted'), loc='upper right')
plt.show()
```



1.2 Función sigmoide

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

1.3 Función de coste y gradiente

A continuación vamos a implementar las funciones de coste y de gradiente en su versión vectorizada.

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

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

```
def coste(Theta, X, Y):
    m = np.shape(X)[0]
    Aux = (np.log(g(X @ Theta))).T @ Y
    Aux += (np.log(1 - g(X @ Theta))).T @ (1 - Y)
    return -Aux / m

def gradiente(Theta, X, Y):
    m = np.shape(X)[0]
    Aux = X.T @ (g(X @ Theta) - Y)
    return Aux / m
```

Vamos a inicializar todos los elementos de theta para comprobar que nuestras funciones son correctas, deberíamos obtener un valor para la función de coste de 0,693, aproximadamente, y un gradiente de [-0.1 -12.0092 -11.2628].

```
(base) C:\Users\alvar\Google Drive\Curso2020_21\AprendizajeAutomatico\2>python 2.py
0.6931471805599452
[ -0.1        -12.00921659 -11.26284221]
```

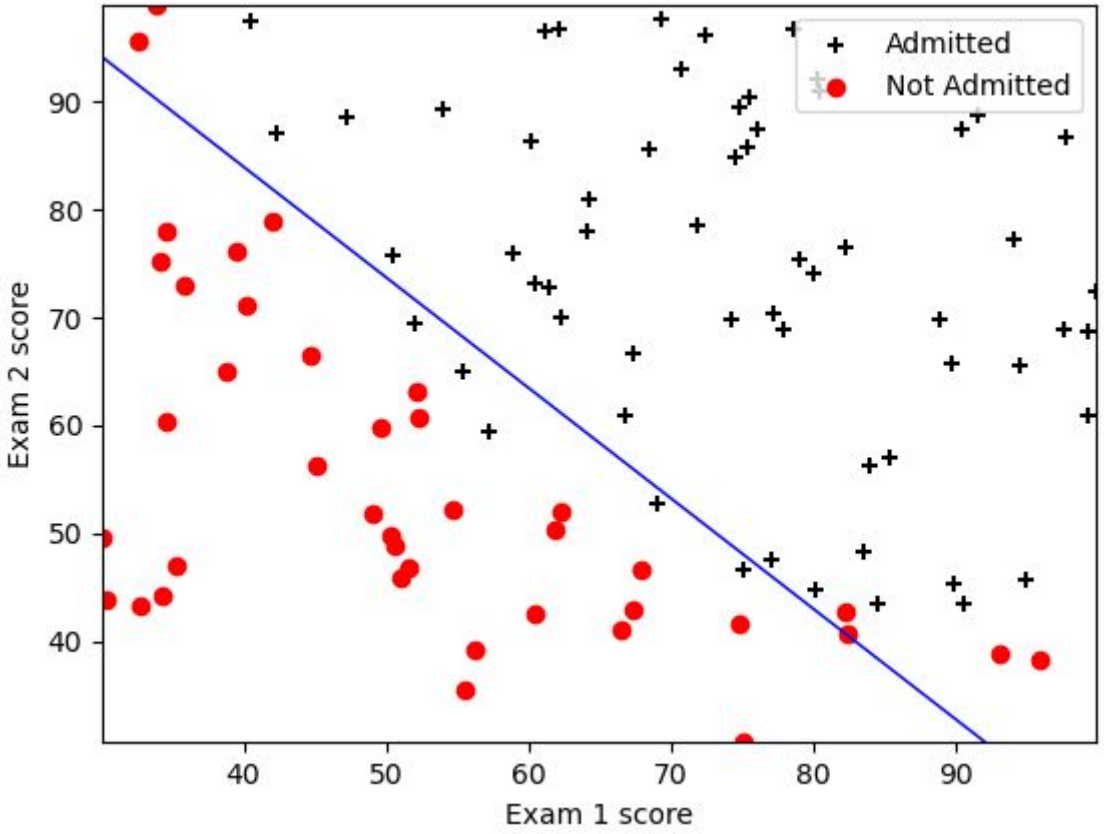
Lo ejecutamos y efectivamente son correctas.

1.4 Cálculo del valor óptimo de los parámetros

Vamos a utilizar la función `scipy.optimize.fmin_tnc` de SciPy para obtener los valores de theta que minimizan la función de coste para la regresión logística implementada.

```
result = opt.fmin_tnc(func=coste , x0=Theta , fprime=gradiente , args=(Xvec, Y))
theta_opt = result [0]
```

Con estos nuevos valores de theta la frontera de decisión es la siguiente:



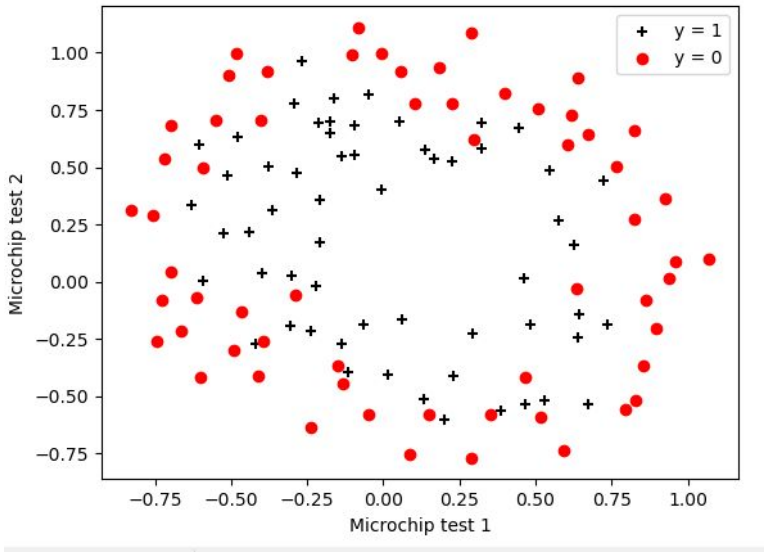
1.5 Evaluación de la regresión logística

En esta parte implementamos una función que calcula el porcentaje de ejemplos de entrenamiento que se clasifican correctamente.

```
98.33333333333333% de positivos clasificado correctamente.
97.5% de positivos clasificado correctamente.
```

2.Regresión logística regularizada

En este apartado vamos a utilizar regresión logística regularizada para encontrar una función que pueda predecir si un microchip pasará o no el control de calidad a través de dos test. Lo primero que hacemos es visualizar los datos de la misma manera que en la misma parte:



2.1 Mapeo de los atributos

A continuación vamos a mapear los atributos y extender cada ejemplo de entrenamiento con los términos polinómicos de x1 y x2 hasta la sexta potencia,completando así un total de 28 atributos para cada ejemplo

```
m = np.shape(X)[0]
poly = PolynomialFeatures(degree=6)
X_reg = poly.fit_transform(X)
n = np.shape(X_reg)[1]
```

2.2 Cálculo de la función de coste y su gradiente

Ahora vamos a cambiar las funciones de coste y gradiente añadiendoles la regularización

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

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

```python
def coste_reg(Theta, X, Y, Lambda):
    m = np.shape(X)[0]
    Aux = (np.log(g(X @ Theta))).T @ Y
    Aux += (np.log(1 - g(X @ Theta))).T @ (1 - Y)
    Cost = -Aux / m
    Regcost = (Lambda / (2 * m)) * sum(Theta ** 2)
    return Cost + Regcost

def gradiente_reg(Theta, X, Y, Lambda):
    m = np.shape(X)[0]
    Aux = X.T @ (g(X @ Theta) - Y)
    Grad = Aux / m
    theta_aux = Theta
    theta_aux[0] = 0.0
    Grad = Grad + (Lambda / m) * theta_aux
    return Grad
```

Vamos a probar estas funciones inicializando theta con ceros y lambda a 1, el coste debería ser 0,693 aproximadamente.

```
(base) C:\Users\alvar\Google Drive\Curso2020_21\AprendizajeAutomatico\2>python 2.py
0.6931471805599453
```

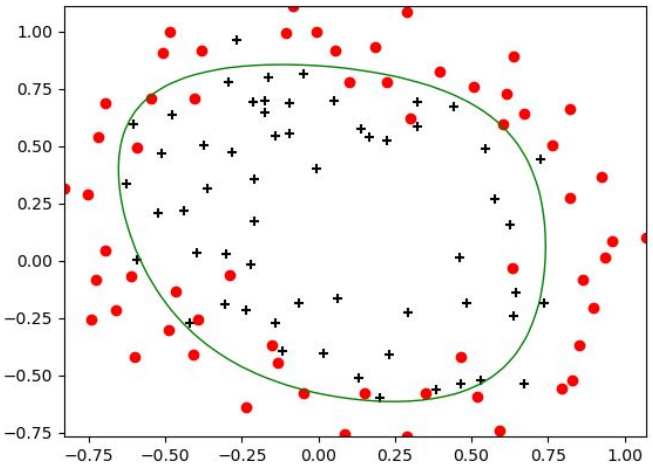
2.3 Cálculo del valor óptimo de los parámetros

Calculamos el valor óptimo para Theta usando scipy.optimize.fmin_tnc

```python
result = opt.fmin_tnc(func=coste_reg, x0=Theta,
                      fprime=gradiente_reg, args=(X_reg, Y, Lambda))
theta_opt = result[0]
```

Y dibujamos la gráfica con:

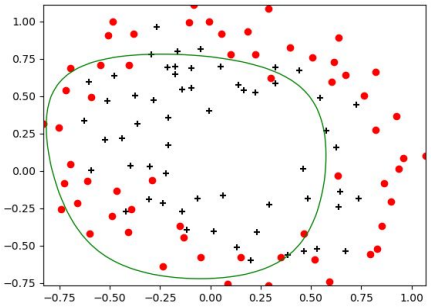
```python
def plot_decisionboundary(Theta, X, Y, poly):
    plt.figure()

    pos = np.where(Y == 1)
    neg = np.where(Y == 0)
    pts_pos = plt.scatter(X[pos, 0], X[pos, 1], c='k', marker='+')
    pts_neg = plt.scatter(X[neg, 0], X[neg, 1], c='r', marker='o')

    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 = g(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.show()
```



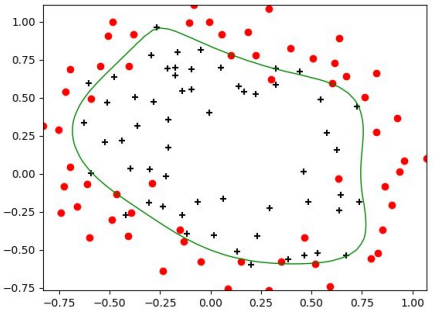
2.4 Efectos de la regularización

Vamos a cambiar el valor de Lambda para ver cómo afecta al resultado:

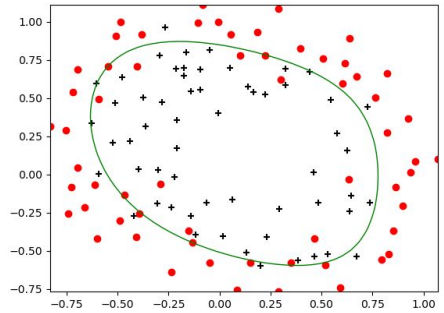
Lambda = 20



Lambda = 0.005



Lambda = 0.1



Podemos apreciar que cuanto más pequeño sea lambda, más se ajusta la frontera a los valores.