Aprendizaje automático y minería de datos

# Práctica 2 – Regresión logística

Ramón Arjona Quiñones
Celia Castaños Bornaechea

# Contenido

# 1. Regresión Logística

## Objetivo

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.

## Método de carga

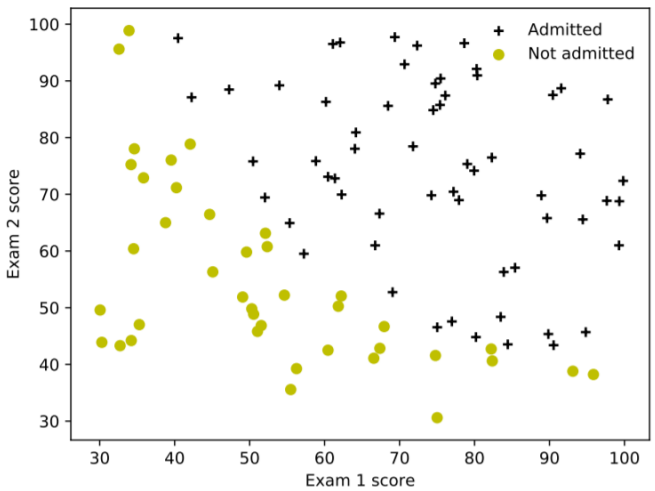
Carga el fichero especificado y devuelve los datos en formato array de numpy. Supondremos que dichos datos serán tipo *float*

```
def carga_csv(file_name):
    """carga el fichero csv especificado y lo devuelve en un array de
      numpy
    """
    valores = read_csv(file_name, header=None).values

    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

## 1.1 Visualización de los datos

Método encargado de mostrar en una gráfica los datos de los casos de entrenamiento.




```
## Pinta los datos para una regresión logística, con '•' para las Y = 1 y
      '+' para las Y = 0
def visualizacion_datos(X, Y, labelX, labelY, title=""):

    # Obtienemos vectores con los índices de los ejemplos cuya Y es 0/1
      (respectivamente)
    pos0 = np.where (Y == 0 )
    pos1 = np.where (Y == 1 )
```

```
    # Los dibujamos con distintos símbolos
    plt.scatter(X[pos0, 0], X[pos0, 1], marker='o' ,c='darkkhaki',
      label="y = 0"  )
    plt.scatter(X[pos1,0], X[pos1, 1], marker='+' ,c='k', label="y = 1" )

    # Pintamos los ejes, el título y la leyenda
    plt.xlabel(labelX)
    plt.ylabel(labelY)
    plt.title(title)
    plt.legend()
    plt.legend(loc='upper right')
```

### 1.2 Función sigmoide

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

Es aplicable tanto para un número, un vector o una matriz. La función sigmoide se utiliza como sustituta a la función salto. Al ser esta última discontinua y, por lo tanto, no derivable se busca una parecida pero continua y esa es la función sigmoide. Todos los valores se encuentran en el intervalo (0, 1).

```
## Calcula el sigmoide del número z
def sigmoid(z): #Ej. 1.2
    s = 1 / (1 + np.exp(-z))
    return s
```

### 1.3 Función coste

Calcula cómo de lejos está la hipótesis calculada de los ejemplos de entrenamiento. Esto nos indica cómo de buena es la función hipótesis. Hay dos formas, mediante regresión logística:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Y de forma vectorizada (forma utilizada en la práctica):

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

```
## Calcula el coste sobre los ejemplos de entrenamiento para un cierto
      valor de theta
def cost(theta, X, Y):
    H = sigmoid(np.matmul(X, theta))
    cost = (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
      np.log(1 - H)))
    return cost
```

### 1.3 Función gradiente

Gracias a ella encontramos los parámetros de nuestro modelo que mejor definan el conjunto de entrenamiento. Nos vamos acercando iterativamente a un valor que minimiza la función de coste.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Al igual que la función coste tiene una forma vectorizada.

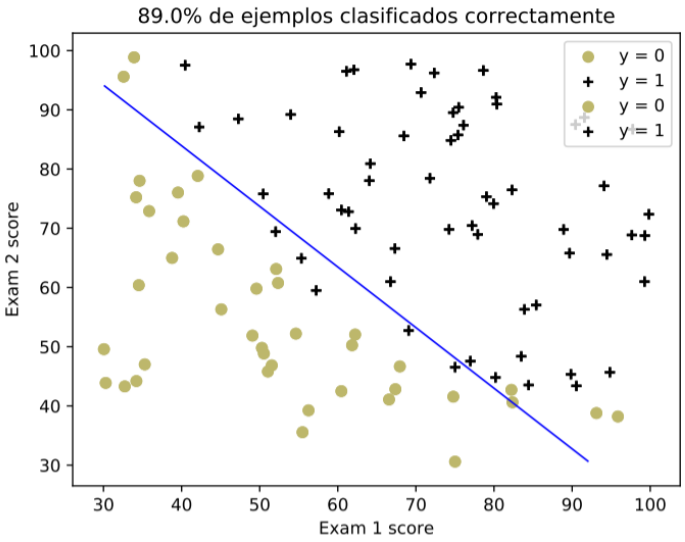
$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m}X^T(g(X\theta) - y)$$

El vector resultante tiene la misma longitud que *theta.*

```
## Calcula el gradiente sobre los ejemplos de entrenamiento para un
      cierto valor de theta
def gradiente(theta, X, Y):
    H = sigmoid(np.matmul(X, theta))
    grad = (1/ len(Y)) * np.matmul(X.T, H-Y)
    return grad
```

### 1.4 Frontera regresión logística mediante una recta

Con el coste y el valor de *theta* óptimos se define una recta (frontera) que separa los distintos casos de entrenamiento según su valor de salida (0 o 1).




```
## Pinta la frontera de decisión en la regresión lineal (como una recta)
def pinta_frontera_recta(X, Y, theta):
    # Mínimo y máximo valor para cada componente de la X
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()
```

```
    # Hacemos un meshgrid
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
    np.linspace(x2_min, x2_max))

    # Hallamos el sigmoide y cambiamos su tamaño
    h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)),
    xx1.ravel(),
    xx2.ravel()].dot(theta))
    h = h.reshape(xx1.shape)

    # Pintamos la frontera para z = 0.5
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
```

### 1.5 Cálculo del porcentaje

Calcula el porcentaje de ejemplos de entrenamiento que se clasifican correctamente utilizando el vector *theta.*

Se calcula el valor de la función sigmoide sobre cada ejemplo de entrenamiento, e interpreta que si el resultado es ≥ 0,5 el alumno será admitido (1) y si es menor no lo será (0).

```
## Calcula el porcentaje de ejemplos de entrenamiento que han sido
clasificados correctamente
def calcula_porcentaje(X, Y, theta):

    #Calculamos el sigmoide
    n = Y.shape[0]
    H = sigmoid(np.matmul(X, theta))

    # Ponemos los elementos de H a 1 (si hi > 0.5) o a 0 (si hi < 0.5)
    H = np.where (H > 0.5, H, 0)
    H = np.where (H < 0.5, H, 1)

    # Vemos cuantos de ellos coinciden con Y y devolvemos el porcentaje
      sobre el total de ejemplos
    coinciden = np.where ( Y == H )
    aciertos = len(coinciden[0])

    return (aciertos / n) * 100
```

### Código completo

```
## Ejercicio 1: regresión logística (frontera divisible por una recta)
def Ejercicio1():
    #Leemos los valores de la matriz de datos
    valores = carga_csv("ex2data1.csv")
    X = valores[:, :-1]
    Y = valores[:, -1]

    #Inicializamos theta y ponemos la columna de 1's a las X
    theta = np.zeros((X.shape[1] + 1))
    m = X.shape[0]
    unos = np.ones((m, 1))
    unosX = np.hstack((unos, X))

    # Calculamos el vector de pesos óptimo gracias a la función fmin_tnc
      (que recibe el valor inicial, las funciones de coste y gradiente y
      los parámetros extra necesarios (X, Y y lambda))
    result = opt.fmin_tnc(func = cost, x0=theta, fprime=gradiente,
      args=(unosX,Y))
    theta_opt = result[0]

    #Porcentaje
    porc = calcula_porcentaje(unosX, Y, theta_opt)

    # Pintamos los datos y la frontera de decisión
    title = str(porc) + "% de ejemplos clasificados correctamente"
    visualizacion_datos(X, Y, "Exam 1 score", "Exam 2 score", title)
    pinta_frontera_recta(X, Y, theta_opt)

    # Guardamos la gráfica y cerramos
    plt.savefig("RegresionLogistica.pdf")
    plt.close()
```

## 2. Regresión Logística Regularizada

### Objetivo

Utilizando la regresión logística regularizada hay que encontrar una función que pueda predecir si un microchip pasará o no el control de calidad. Esto se llevará a cabo a partir del resultado de dos tests a los que se les ha sometido. Para ello cada pareja de valores de la matriz X se sustituirá por 28 atributos obtenidos de multiplicar los valores de la pareja entre ellos hasta obtener grado 6.

## 2.2 Función coste

La función coste para la regresión logística regularizada es una implementación vectorizada de su expresión matemática.

$$J(\theta) = \left[\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Expresión matemática.

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

Expresión matemática vectorizada.

Esta segunda es la fórmula utilizada.

```
def regularizedCost(theta, lamda, X, Y):
    regCost = cost(theta, X, Y)

   #Añadimos el sumatorio de thetas al cuadrado al valor del coste normal
    regCost = regCost + (lamda / 2*(len(X))) * np.sum(theta**2)
    return regCost
```

## 2.2 Función gradiente

Vector de la misma longitud que *theta*.

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{para } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{para } j \geq 1$$

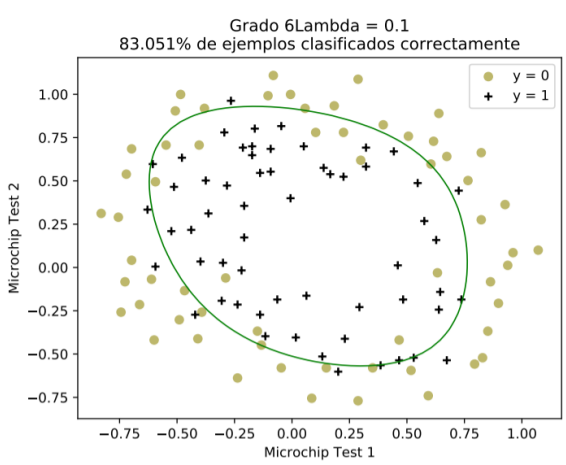
Y a continuación su forma vectorizada.

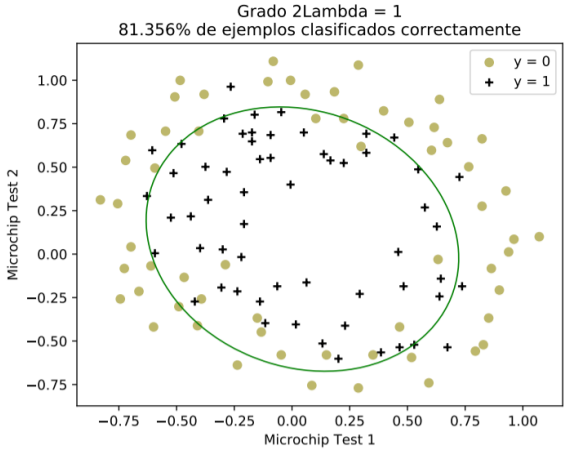
$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m}X^T(g(X\theta) - y) + \frac{\lambda}{m}\theta_j$$

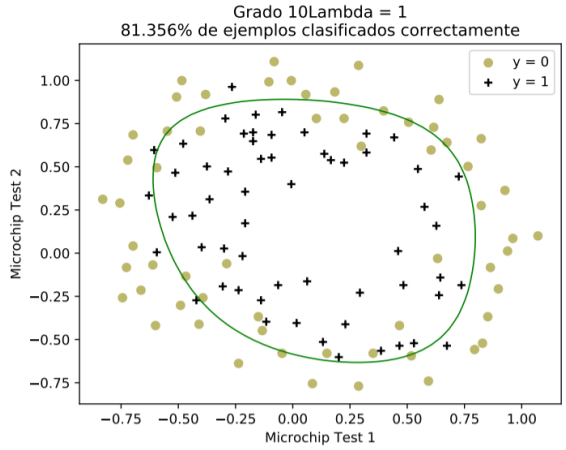
```
## Calcula el gradiente de forma regularizada para un cierto lambda
def regularizedGradient(theta, lamda, X, Y):
    regGrad = gradiente(theta, X, Y)
    aux = np.copy(theta)
    aux[0] = 0
    #Añadimos el sumatorio de thetas al valor del gradiente normal
    regGrad = regGrad + (lamda / len(Y) * aux)
    return regGrad
```

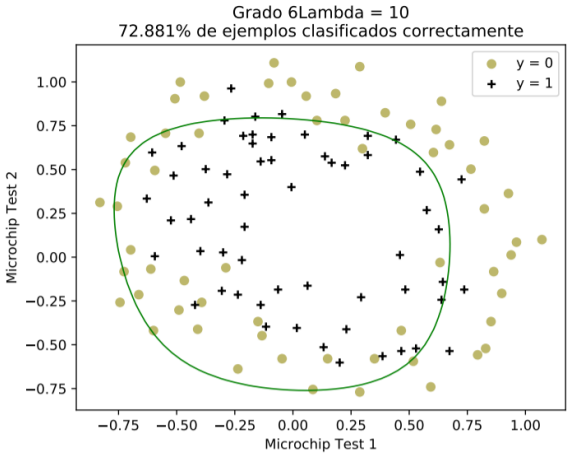
## 2.4 Frontera regresión logística mediante *contour*

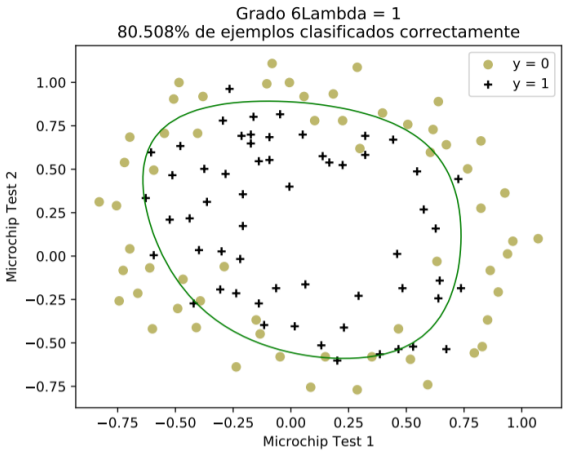
Dibuja una frontera que separa las distintas salidas de los casos de entrenamiento recibiendo las características del polinomio a partir de las cuales se creó *X*. En este apartado se prueba el resultado de las fronteras dándole a *lambda* distintos valores.




```
## Pinta la frontera de decisión en la regresión logística (más precisa
      que la recta)
def plot_decisionboundary(X, Y, theta, poly):
    #Igual que pinta_frontera_recta pero con el poly_fit_transform
    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(poly.fit_transform(np.c_[xx1.ravel(), xx2.ravel()]).dot(t
heta))
    h = h.reshape(xx1.shape)

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

## Código completo

```
## Ejercicio 2: regresión logística y regularización (recibe el valor de
lambda)
def Ejercicio2(lamda, grado):
    #Leemos los valores de la matriz de datos
    valores = carga_csv("ex2data2.csv")
    X = valores[:, :-1]
    Y = valores[:, -1]

    # Creamos el polinomio X de grado 6 a partir de combinaciones de x1 y
      x2
    poly = PolynomialFeatures(grado)
    polyX = poly.fit_transform(X)
    theta = np.zeros((polyX.shape[1]))

    # Calculamos el vector de pesos óptimo igual que en el ejercicio 1
      (ahora pasamos también lambda para la regularización)
    result = opt.fmin_tnc(func = regularizedCost, x0=theta,
      fprime=regularizedGradient, args=(lamda, polyX, Y))
    theta_opt = result[0]

    # Calculamos el porcentaje de evaluados correctamente
    porc = round(calcula_porcentaje(polyX, Y, theta_opt), 3)

    # Pintamos los datos y la frontera de decisión
    title = "Grado " + str(grado) + "Lambda = " + str(lamda) + "\n" +
      str(porc) + "% de ejemplos clasificados correctamente"
    visualizacion_datos(X, Y, "Microchip Test 1", "Microchip Test 2",
      title)
    plot_decisionboundary(X, Y, theta_opt, poly)

    # Guardamos la gráfica y cerramos
    plt.savefig("LogisticaRegularizada_grado" + str(grado) + "_lambda" +
      str(lamda) + ".pdf")
    plt.close()
```