Práctica 2: Regresión Logística

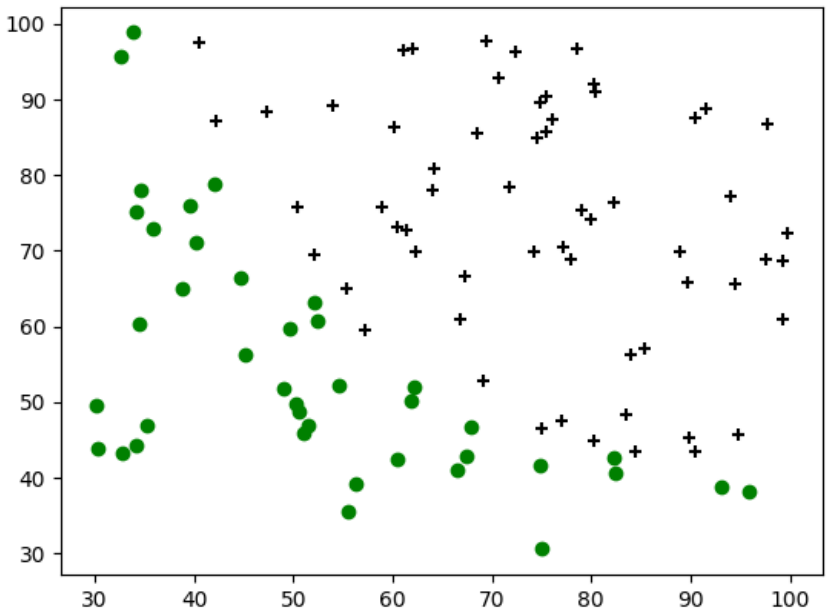
David Godoy Ruiz
Eva Lucas Leiro

Regresión logística:

- Visualización de los datos

Para visualizar los datos obtenemos por un lado las X y por otro las Y y pintamos los puntos de tal forma que los puntos con y= 1 son + y los y=0 son círculos.

```
datos = carga_csv('ex2data1.csv')
X = datos[:, 0:2]
Y = datos[:, 2]
pos = np.where(Y==1)
plt.scatter(X[pos,0], X[pos,1], marker='+', c='k')

pos1 = np.where(Y==0)
plt.scatter(X[pos1,0], X[pos1,1], marker='o', c='green')
plt.savefig("grafica1.png")
```



- Función sigmoide

```
def sigmoid(z):

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

- Cálculo de la función de coste y su gradiente

Para calcular la función de coste calculamos primero el valor de h(x) = g(Xθ) y luego aplicamos la función de coste:

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

Hacemos lo mismo para calcular la gradiente, en este caso con la fórmula:

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

```
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

#---------------------------------------------------------------------

def gradient(theta, XX, Y):
    H = sigmoid( np.matmul(XX,theta))
    grad =(1/len(Y))* np.matmul(XX.T, H - Y)
    return grad
```

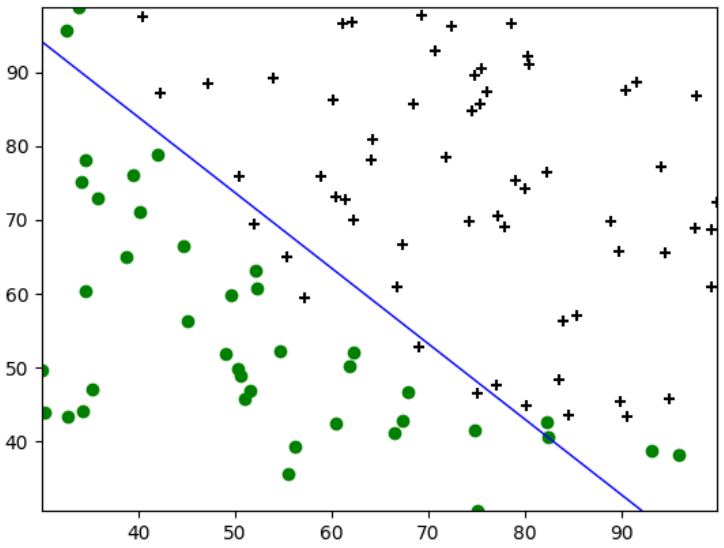
- Cálculo del valor óptimo de los parámetros

Para calcular el valor óptimo de los parámetros llamamos al método fmin_tnc de la librería scipy.optimize

```
 result = opt.fmin_tnc( func=cost , x0=theta , fprime=gradient , args
=(X,Y))
```

Para después pintar la frontera de decisión:

```
    plt.figure()
    pos = np.where(Y==1)
    plt.scatter(X[pos,1], X[pos,2], marker='+', c='k')

    pos1 = np.where(Y==0)
    plt.scatter(X[pos1,1], X[pos1,2], marker='o', c='green')

    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(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.savefig("frontera.png")
```

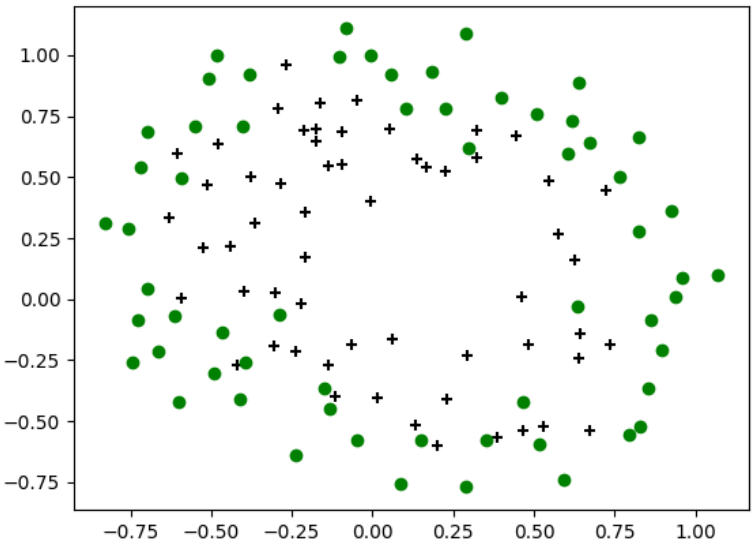


- Evaluación de la regresión logística

Para calcular la evaluación de la regresión logística contamos cada par de valores de X cuya función sigmoide de un resultado que prediga el dato de entrenamiento

```
def evaluacion(theta, X, Y):
    correctos = np.sum((sigmoid(np.dot(X, theta))>=0.5)==Y)
    return correctos
```

Regresión logística regularizada:



- Mapeo de los atributos

Para el mapeo llamamos al método PolynomialFeatures de sklearn.preprocessing que genera la expresión polinómica hasta el sexto exponente para luego transformar X de tal forma que haya un valor de entrenamiento para cada parte del polinomio (28 en total)

```
def mapfeature(X):
    poly = PolynomialFeatures(6)
    return poly.fit_transform(X)
```

- ## Cálculo de la función de coste y su gradiente

Calcular la función de coste y su gradiente se hace de forma muy similar a lo anterior pero añadiendo un parámetro más para ambas que va en función de λ:

Coste:

$$+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

```
def coste_reg(theta, X, Y, lambd):
    H = sigmoid(np.matmul(X, theta))
    cost = (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
np.log(1 - H))) + lambd/(2*len(X)) * np.sum(theta**2)
    return cost
```

Gradiente:

$$+\frac{\lambda}{m}\theta_j$$

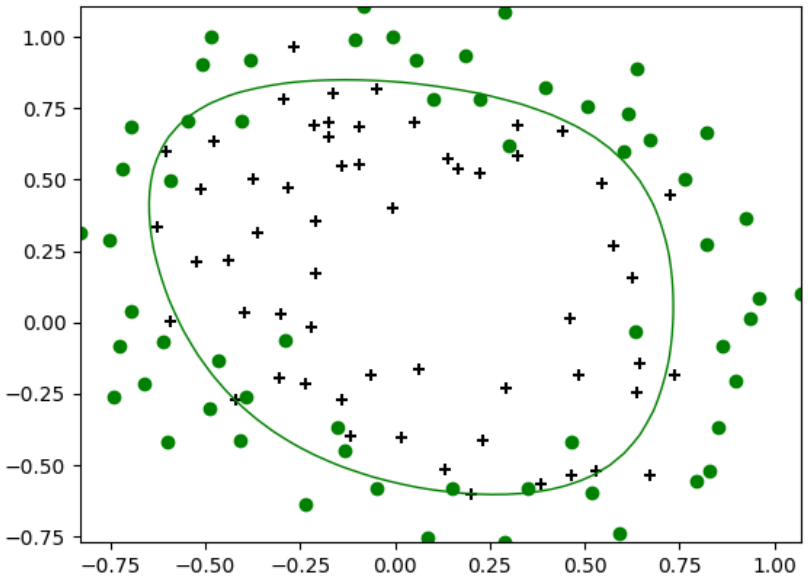
```
def gradiente_reg(theta, XX, Y, lambd):
    H = sigmoid( np.matmul(XX,theta))
    grad =(1/len(Y))* np.matmul(XX.T, H - Y) + lambd/(len(XX)) * theta
    return grad
```

- ## Cálculo del valor óptimo de los parámetros

Para calcular el valor óptimo de los parámetros llamamos al método fmin_tnc de la librería scipy.optimize, como se hizo anteriormente

```
    result = opt.fmin_tnc( func=coste_reg , x0=theta ,
fprime=gradiente_reg , args =(X,Y, 1))
```

Para calcular el valor óptimo de los parámetros llamamos al método fmin_tnc de la librería scipy.optimize, como se hizo anteriormente, que da la theta representada en esta gráfica:

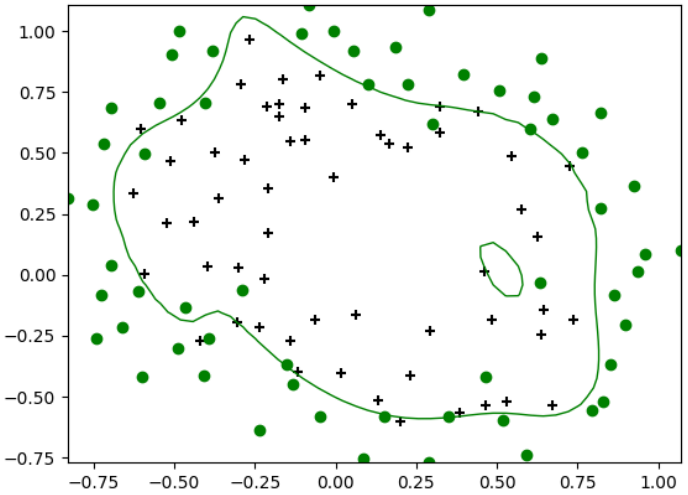


- Efectos de la regularización

Variando lambda podemos ver estos efectos:

- Para lambda = 1 (ver arriba), la frontera de decisión se ajusta bien al ejemplo, aunque deje algunos valores fuera

- Para lambda = 0, la frontera de decisión se ajusta demasiado a los valores, lo que provoca que la predicción sea menos precisa:



- Para lambda = 100 ocurre lo opuesto: la frontera de decisión no se ajusta para nada a los valores de ejemplo, siendo igualmente imprecisa