Práctica 2: Regresión logística

Manuel Hernández Nájera-Alesón
Mario Jiménez Contreras
Grupo 17

Parte 1: Regresión Logística

1.1 Planteamiento:

En este caso se nos presenta una muestra que toma los valores 0 ó 1, por lo tanto la metodología utilizada en regresión lineal no se adapta del todo a este problema y debemos utilizar regresión logística, que es muy similar, pero cambiando la hipótesis por la función sigmoide.

Para abordar la práctica comenzamos escribiendo las funciones básicas para el desarrollo de esta, empezando por la función de coste, que es el valor a minimizar, como hemos dicho, es igual que en regresión lineal, solo que con distinta hipótesis, por lo que, vectorizada y simplificada resulta:

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

Que definimos en código como:

```
def fun_coste(thetas, matrizX, vectorY, muestras):
   H = sigmoide(np.dot(matrizX, thetas))
   oper1 = -(float(1)/muestras)
   oper2 = np.dot((np.log(H)).T, vectorY)
   oper3 = (np.log(1-H)).T
   oper4 = 1-vectorY

   return oper1 * (oper2 + np.dot(oper3, oper4))
```

(Lo hemos separado en distintos operandos por simplificarnos la visión general de la ecuación)

Donde *VectorX* y *VectorY* son variables de uso global ya que todos los métodos las han de usar.
Después, declaramos una función para cada descenso por separado, una para θ_0 y otra para θ_1:

El descenso de gradiente en esta ocasión se calcula mediante la fórmula:

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

Código:

```
def alg_desGrad(thetas, matrizX, vectorY, muestras):
    H = sigmoide(np.dot(matrizX, thetas))
    return np.dot((1.0/muestras), matrizX.T).dot(H-vectorY)
```

Procedimiento:

Importación de librerías

```
import random as rnd
import matplotlib as mp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import scipy as sp
import time
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import scipy.optimize as opt
```

Obtención datos de entrenamiento

```
file = open('ex2data1.csv',encoding="utf8",errors='ignore')
datos = csv.reader(file)

lineas = np.array(list(datos))
muestras = len(lineas)
vectorY = np.array([0 for n in range(muestras)])

numcols = len(lineas[0])
tempa = np.ones(muestras)
X_plain = lineas.T[0:-1].astype(float)
z = (tempa, X_plain)
matrizX = ((np.vstack(z)).T).astype(float)
#np.array([[np.zeros(numcols-1)]for y in range (muestras)]))
vectorY = np.hstack(lineas.T[-1]).astype(int)
thetas = np.zeros(numcols)
```

Cálculo valor óptimo

```
result = opt.fmin_tnc(func = fun_coste, x0 =thetas,
fprime=alg_desGrad, args=(matrizX, vectorY, muestras))
thetas_opt = result[0]
```

Agrupación de datos

```
plt.figure()
pinta_frontera_recta(X_plain.T, vectorY, thetas_opt)
pos = np.where(vectorY==1)
neg = np.where(vectorY==0)
#Grafica
plt.scatter(X_plain.T[pos, 0], X_plain.T[pos, 1], marker='+',
c='k', label = "Admitted")
plt.scatter(X_plain.T[neg, 0], X_plain.T[neg, 1], marker='o',
c='y', label = "Not Admited")
plt.legend()
```

Función encargada de la representación de la frontera

```
def pinta_frontera_recta(X, Y, theta):
   plt.figure()
   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 = sigmoide(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')
```

Los resultados finales se extraen mediante este fragmento de código:

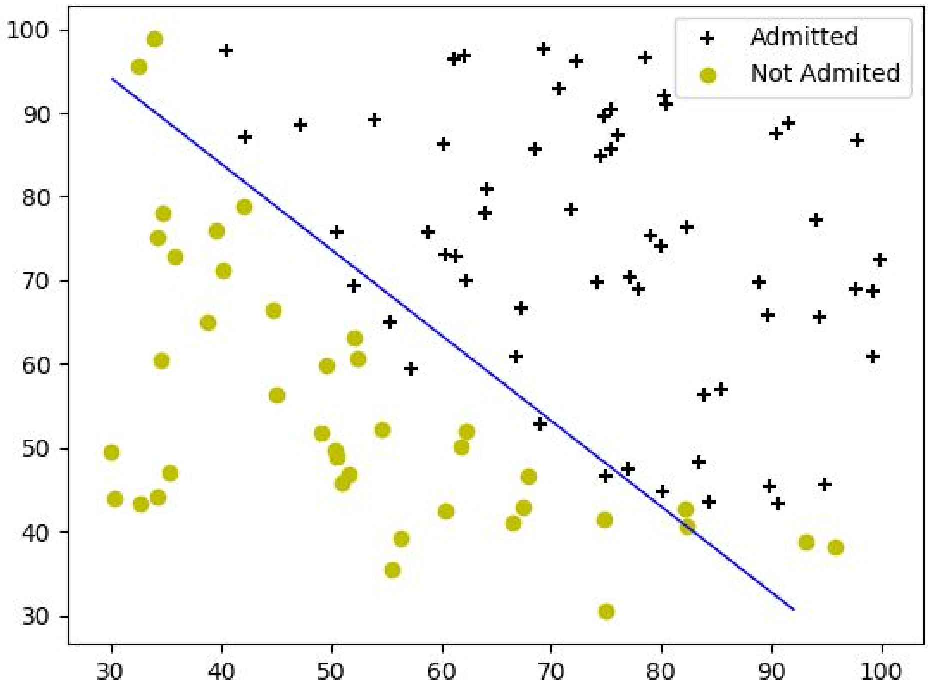
```
yEv = predict_Y(matrizX, thetas_opt)
yEv[ yEv >= 0.5] = 1
yEv[ yEv < 0.5] = 0
print(yEv)

pcent = np.asarray(np.where(yEv == vectorY)).size
pcent = (pcent / len(vectorY))*100
print("El porcentaje de aciertos máquina es : ", pcent, "%")
```

predict_Y:

```
def predict_Y (vectX, thetas):
   return np.matmul(vectX, thetas.T)
```

1.2 Resultados:

El resultado de haber agrupado los puntos positivos en un vector y los negativos en otro distinto nos da la posibilidad de pintarlos con distintos colores y forma, así como la recta generada por el vector de Thetas optimizado, que sirve como frontera entre ambos grupos.



Parte 2: Regresión logística regularizada:

En este caso la frontera no es tan simple y se requieren más parámetros para definir su forma. Para no caer en el problema del over o under-fitting se utiliza regularización.

Las funciones son idénticas a las del apartado anterior, pero se les añade el operando regulador.

Función de coste:

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

Función descenso de gradiente:

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

<u>Código:</u>

Func. coste regularizada

```
def fun_coste(thetas, matrizX, vectorY, muestras, _lambda):
    H = sigmoide(np.dot(matrizX, thetas))
    oper1 = -(float(1)/muestras)
    oper2 = np.dot((np.log(H)).T, vectorY)
    oper3 = (np.log(1-H)).T
    oper4 = 1-vectorY
    oper5 = (_lambda/(2*muestras))*np.sum(thetas**2)
    return (oper1 * (oper2 + np.dot(oper3, oper4)))+ oper5
```

Func. descenso de gradiente

```
def alg_desGrad(thetas, matrizX, vectorY, muestras, _lambda):
    H = sigmoide(np.dot(matrizX, thetas))
    return (np.dot((1.0/muestras),
matrizX.T).dot(H-vectorY))+(_lambda/muestras)*thetas
```

Func. sacar gráfica

```
def print_pantalla(X, Y, theta, poly):
    plt.figure()
    pos = np.where(vectorY==1)
    neg = np.where(vectorY==0)

    plt.scatter(X_plain.T[pos, 0], X_plain.T[pos, 1], marker='+',
c='k', label = "y = 1")
    plt.scatter(X_plain.T[neg, 0], X_plain.T[neg, 1], marker='o',
c='y', label = "y = 0")

    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 = sigmoide(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.legend()
    #plt.show()
    plt.savefig("grafica.png")
```

Evaluar resultados

```
def evaluate(X, Y, thetas):
    yEv = predict_Y(X, thetas)
    yEv[ yEv >= 0.5] = 1
    yEv[ yEv < 0.5] = 0
    pcent = np.asarray(np.where(yEv == Y)).size
    pcent = (pcent / len(Y))*100
    print("El porcentaje de aciertos máquina es : ", pcent, "%")
```

Procedimiento:

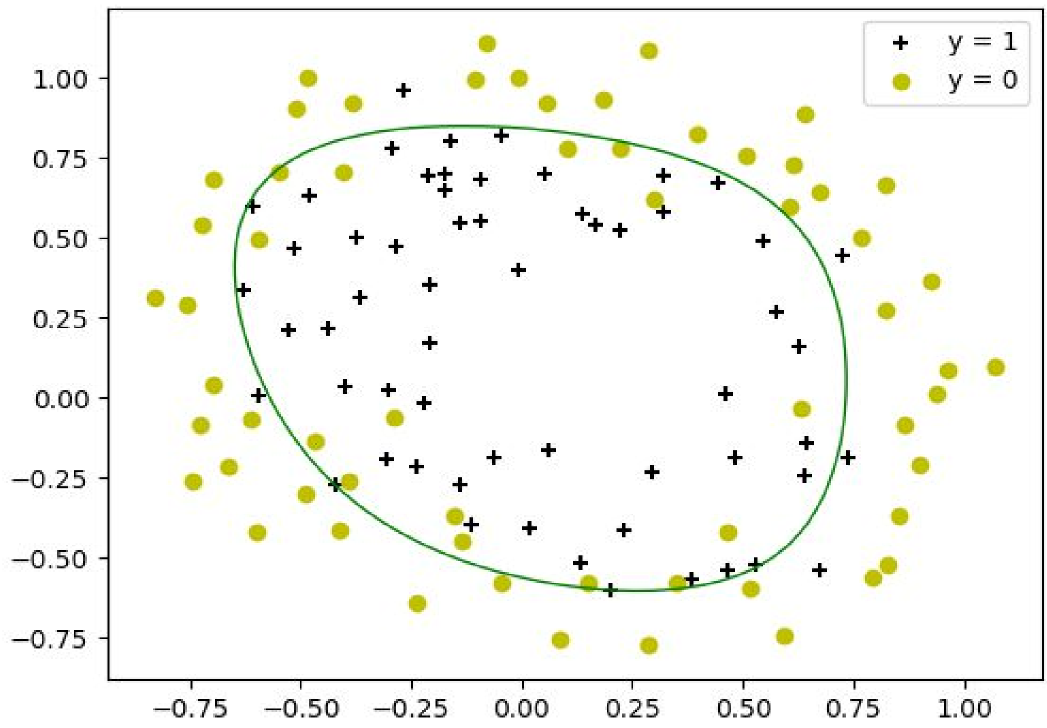
```
# Lectura de datos
file = open('ex2data2.csv',encoding="utf8",errors='ignore')
datos = csv.reader(file)

lineas = np.array(list(datos))
muestras = len(lineas)
vectorY = np.array([0 for n in range(muestras)]) # Este vector se
utiliza en la funcion de coste
```

```
#Numero de XsubJ que tenemos
numcols = len(lineas[0])
tempa = np.ones(muestras)
X_plain = lineas.T[0:-1].astype(float)

vectorY = np.hstack(lineas.T[-1]).astype(int)
thetas = np.zeros(28)
_lambda = 0.0001

poly = skpp.PolynomialFeatures(6)
matrizX = poly.fit_transform(X_plain.T)
thetas = np.zeros(matrizX.shape[1])

print(matrizX.shape)
print(fun_coste(thetas, matrizX, vectorY, muestras, _lambda))

#Optimizacion del vector
thetas = np.zeros(matrizX.shape[1])
result = opt.fmin_tnc(func = fun_coste, x0 =thetas,
fprime=alg_desGrad, args=(matrizX, vectorY, muestras, _lambda))
thetas_opt = result [0]

print(fun_coste(thetas_opt, matrizX, vectorY, muestras, _lambda))

evaluate(matrizX, vectorY, thetas_opt)
print_pantalla(X_plain.T, vectorY, thetas_opt, poly)
```

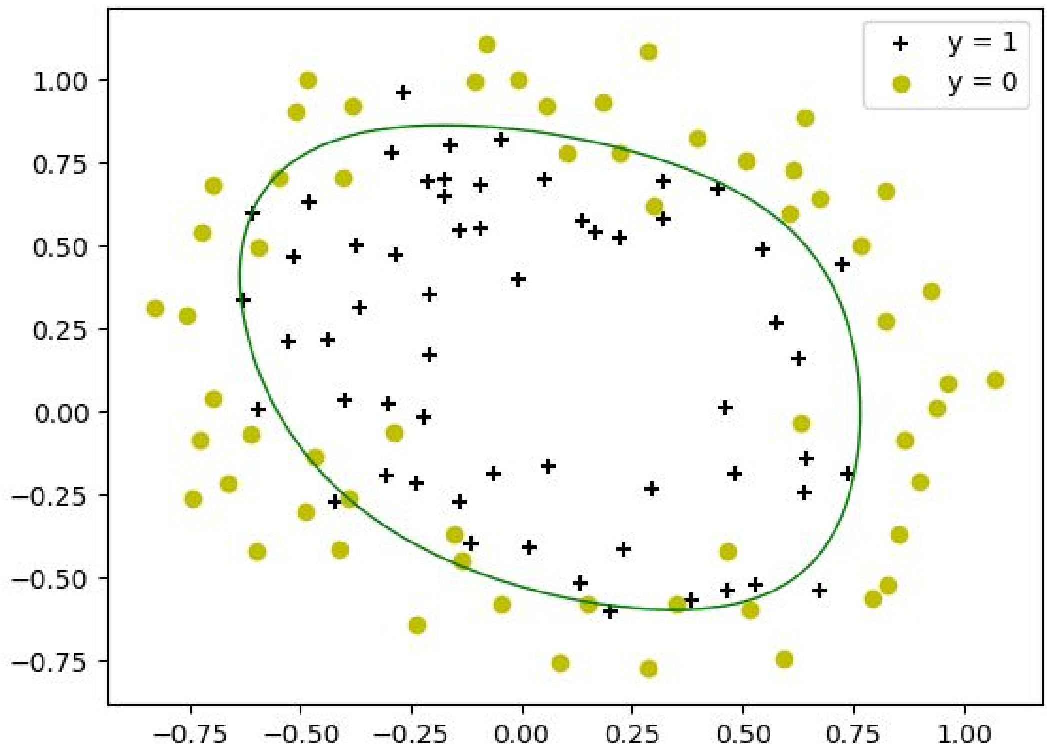
2.2 Resultados:

Probando distintos valores de lambda:

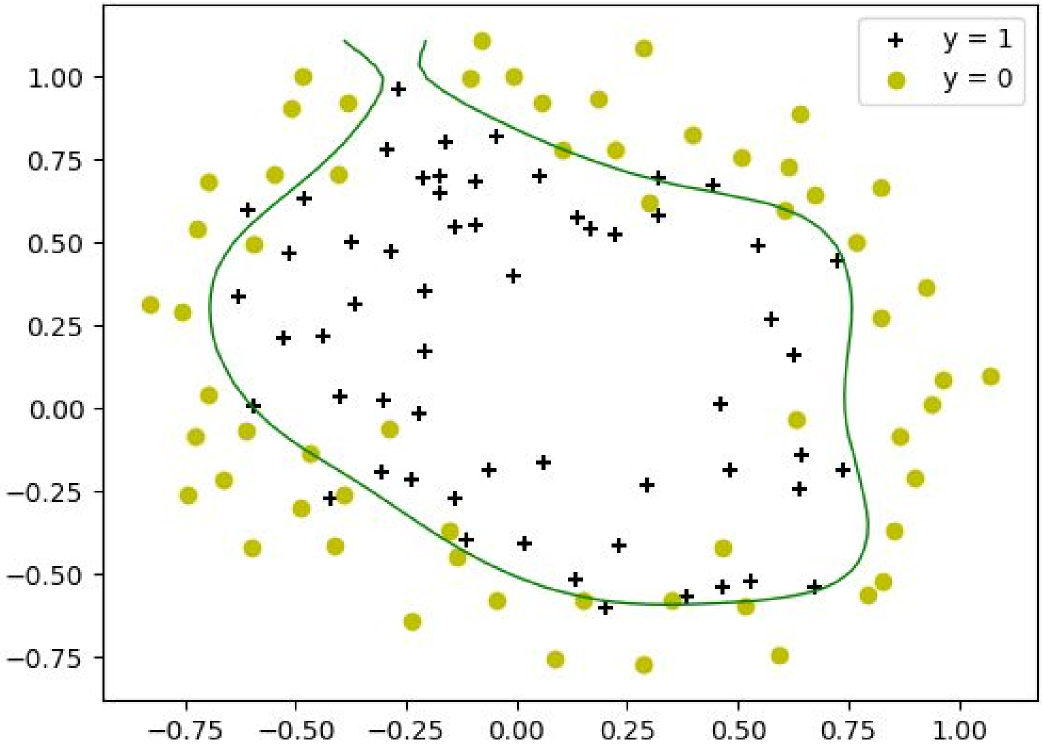
- Lambda = 1, porcentaje de aciertos de 76.27%
- Lambda = 0.5, porcentaje de aciertos de 80.5%
- Lambda = 0.25, porcentaje de aciertos de 81.35%
- Lambda = 0.025, porcentaje de aciertos de 83.89%
- Lambda = 0.0001, porcentaje de aciertos de 87.28%



Lambda = 1



Lambda = 0.25



Lambda = 0.0001

Con estos datos, podemos observar que cuanto más apuramos lambda, mejores resultados se obtienen