Práctica 2: Regresión logística

Rafael Herrera Troca
Rubén Ruperto Díaz

Aprendizaje Automático y Big Data
Doble Grado en Ingeniería Informática y Matemáticas

4 de septiembre de 2020

1. Introducción

En esta práctica hemos implementado una serie de funciones de coste y de gradiente para construir un modelo de regresión logística con el que predecir, en el primer apartado, si un alumno será admitido o no según sus notas en dos exámenes, y, en el segundo, si un microchip está defectuoso o no en función a los resultados de dos test de calidad. El resultado obtenido es una función que, dado un nuevo caso de prueba, devuelve un cierto valor que podemos considerar perteneciente a la clase positiva si es mayor o igual a 0.5 y a la negativa si no.

Hemos programado el código en el entorno *Spyder* con Python 3.7.

2. Procedimiento

Para la realización de la práctica, definimos una serie de funciones de coste y sus gradientes, que luego utilizaremos como parámetros de la función *fmin_tnc* de la librería *scipy.optimize*. Esta última función obtiene los valores de θ que minimizan la función de coste dada usando el gradiente que se le pasa.

Además, definimos la función sigmoide que, dado un cierto valor, lo transforma para que esté entre 0 y 1. Esta función es además la que utilizaremos para que el valor devuelto por el modelo logístico sea interpretable como se explicaba en la introducción: positivo si es mayor o igual que 0.5 y negativo si no lo es. La función sigmoide se define como sigue:

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

2.1. Parte 1: Regresión logística

En este caso, definimos la función de coste según la fórmula que se muestra a continuación:

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

Esta función se presenta en su forma vectorizada, que es como la hemos implementado en el código, haciendo todas las operaciones de forma literal tal y como aparecen en la fórmula.

En cuanto a su gradiente, lo hemos implementado también en su forma vectorizada, y calculamos todos sus elementos a la vez, de forma que tras aplicar la función se obtiene un array con todos los elementos de dicho gradiente. La fórmula que hemos utilizado es la siguiente:

$$\frac{\partial(\theta)}{\partial\theta_j} = \frac{1}{N} X^T (g(X\theta) - y)$$

Una vez tenemos estas dos funciones, para el resto de la primera parte basta con usar *fmin_ tnc* para obtener los valores de θ que hacen mínimo el coste. Estos valores serán los que definen nuestro modelo, de forma que, dado un nuevo dato X, la predicción para éste viene dada por $g(X\theta)$. Después, representamos los datos y la función obtenida en una gráfica y calculamos su porcentaje de aciertos. Los resultados obtenidos se explican en la sección correspondiente.

2.2. Parte 2: Regresión logística regularizada

En cuanto a la segunda parte, comenzamos nuevamente por definir la función de coste de forma vectorizada según la fórmula que sigue:

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

Para calcular su gradiente, también de forma vectorizada, seguimos la fórmula que se muestra a continuación:

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

Al igual que en la primera parte, calculamos todo el gradiente a la vez usando un array, no de forma iterativa. En este caso, hay que tener cuidado porque el término final de la fórmula sólo se suma cuando $j \neq 0$. Para hacer esto, sumamos un array que tiene en cada posición el valor de θ_j correspondiente y, en la primera, un 0.

Además, en esta parte, los datos están repartidos más o menos en dos círculos concéntricos, por lo que una recta no es suficiente para separarlos. Sin embargo, como sólo tenemos dos variables por cada dato, no hay suficientes grados de libertad para construir una función que lo estime con suficiente fiabilidad. Para solucionarlo, usamos la clase *PolynomialFeatures* de la librería *sklearn.preprocessing*, que dados unos datos y un grado, construye todas las posibles potencias de estos datos hasta dicho grado. Así, al usarla con grado 6, nuestras 2 variables se transforman en 28, que nos proporcionan la flexibilidad suficiente como para que la función de decisión se adapte correctamente a la distribución de los datos.

Una vez explicado esto, solo resta usar nuevamente *fmin_ tnc* para calcular el valor de θ que minimiza la función de coste, representar el resultado y calcular el porcentaje de aciertos.

Finalmente, estudiamos más en profundidad cómo varía el modelo construido según el valor de λ. Para ello, hemos puesto este parámetro como uno más de los que recibe la función de coste y el gradiente. Así, usando un bucle hemos ido probando distintos valores con una escala logarítmica de base 10, tomándolos entre 10^{-10} y 10^2. Para cada valor, hemos representado el modelo obtenido y almacenado su porcentaje de acierto. Finalmente, hemos representado todos los porcentajes de acierto en una gráfica. Se pueden encontrar detalles específicos sobre los resultados en la siguiente sección.

3. Resultados

3.1. Parte 1: Regresión logística

En el primer caso, buscamos un modelo capaz de predecir si un alumno será admitido a una universidad en función de sus resultados en dos exámenes. La función de coste alcanza un valor mínimo de $J(\theta) = 0{,}203$ para $\theta = (-25{,}161,\ 0{,}206,\ 0{,}201)$, por tanto, la recta $X\theta = 0$, que define la frontera de decisión entre los puntos que se predicen como admitidos o como rechazados, viene dada por la ecuación $0{,}206x_1 + 0{,}201x_2 - 25{,}161 = 0$.

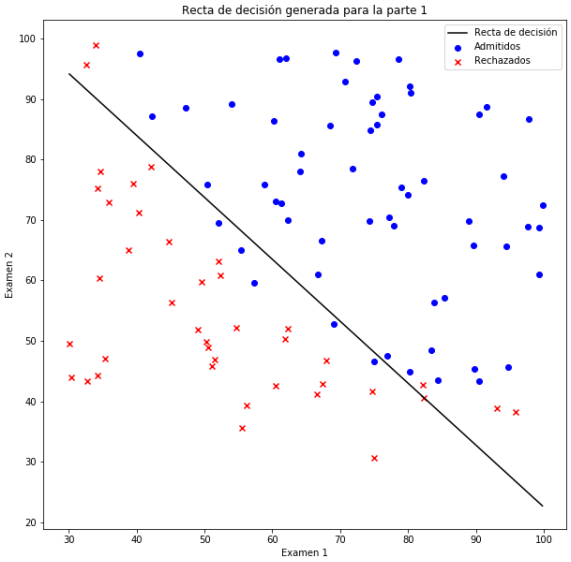

Figura 1: Recta de regresión para la primera parte

Esta recta además coincide con la curva de nivel de altura 0,5 de la función sigmoide $g(X\theta)$, siendo $\theta = (-25{,}161,\ 0{,}206,\ 0{,}201)$ y $X = (1, x_1, x_2)$. Esto se debe a que $g(X\theta) = 0{,}5 \iff X\theta = 0$, donde $X\theta = 0$ es precisamente la ecuación de la frontera de decisión. Por tanto, podemos utilizar la función sigmoide como un clasificador, interpretando que un alumno será admitido si, al introducir sus calificaciones en la función, el resultado es $\geq 0{,}5$ y rechazado en caso contrario. Utilizando este procedimiento, nuestro modelo obtiene un 89 % de éxito al clasificar los datos de entrenamiento.

3.2. Parte 2: Regresión logística regularizada

En este segundo apartado debemos predecir si un microchip está defectuoso o no en función de los resultados de dos test. Sin embargo, como a la vista de los gráficos los datos no son separables mediante una recta, la ecuación que define la frontera de decisión tendrá que tener términos no lineales que combinen las variables x_1 y x_2. De este modo, optimizamos un vector θ más grande, concretamente de 28 componentes, para conseguir una curva de hasta grado 6, y añadimos un término de regularización que depende de λ a la función de coste $J(\theta)$.

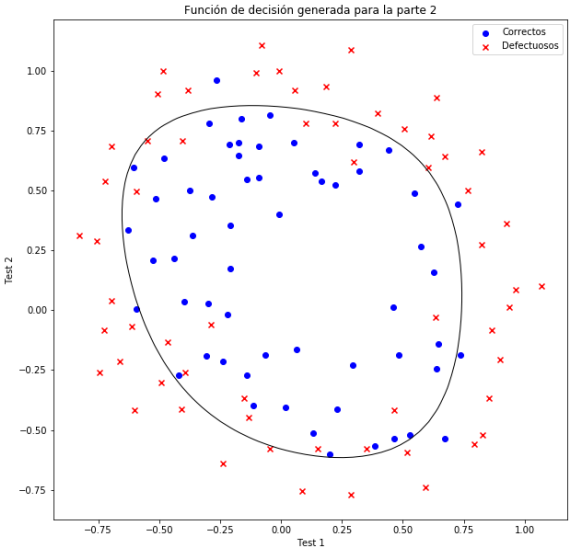

Figura 2: Frontera de decisión para $\lambda = 1$

Tras minimizar la función de coste regularizada con $\lambda = 1$, obtenemos una frontera de decisión que representamos gráficamente mediante la función *contour*, basándonos en que la curva que buscamos es la curva de nivel de altura 0,5 de la función sigmoide, fijado el parámetro θ que minimiza la función de coste regularizada.

Una vez estudiado el caso $\lambda = 1$, hemos probado el comportamiento de la regresión al dar a λ los distintos valores que hay en el conjunto $\{10^i \,|\, i \in [-10, 2], i \in \mathbb{Z}\}$. Hemos realizado una gráfica que muestra el porcentaje de casos de entrenamiento clasificados correctamente en función de los valores dados a λ, siendo importante notar que la escala del eje horizontal, donde se muestran los valores de λ, es logarítmica, por lo que los valores representados son sucesivas potencias de 10.

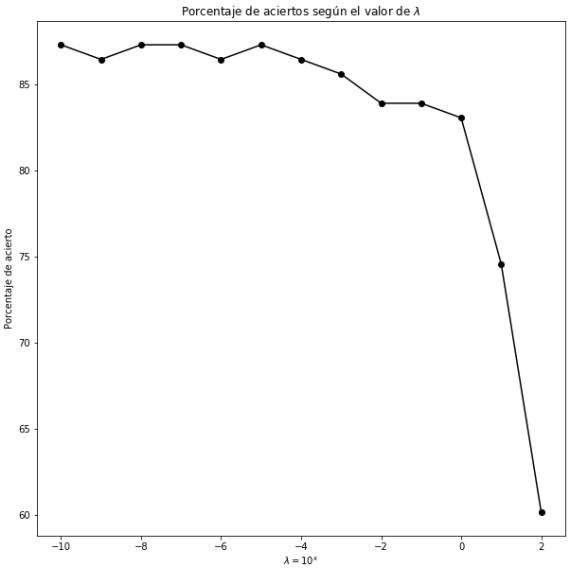

Figura 3: Comparativa de éxito al clasificar los datos de entrenamiento según λ

Vemos que si $\lambda \in \{0{,}01,\ 0{,}1,\ 1\}$ se tiene un porcentaje de aciertos entre 80 y 85 %. Para valores mayores ($\lambda \in \{10,\ 100\}$) la gráfica se desploma, indicando que la curva no se puede ajustar correctamente. Esto se debe a que aumentar λ tiene como consecuencia la disminución del valor de las componentes de θ que acompañan a x_1, x_2 y sus combinaciones, y por tanto se restringe la forma de la curva y se reduce su tamaño.

Por otro lado, valores de $\lambda \leq 0{,}001$ obtienen tasas de acierto de más del 85 % con los datos de entrenamiento, pero esto en realidad no es bueno, ya que se debe a un sobreaprendizaje del modelo. La curva de decisión se ajusta demasiado a los datos de entrenamiento, permitiendo más aciertos entre esos datos, pero disminuyendo el éxito para predicciones con datos nuevos.

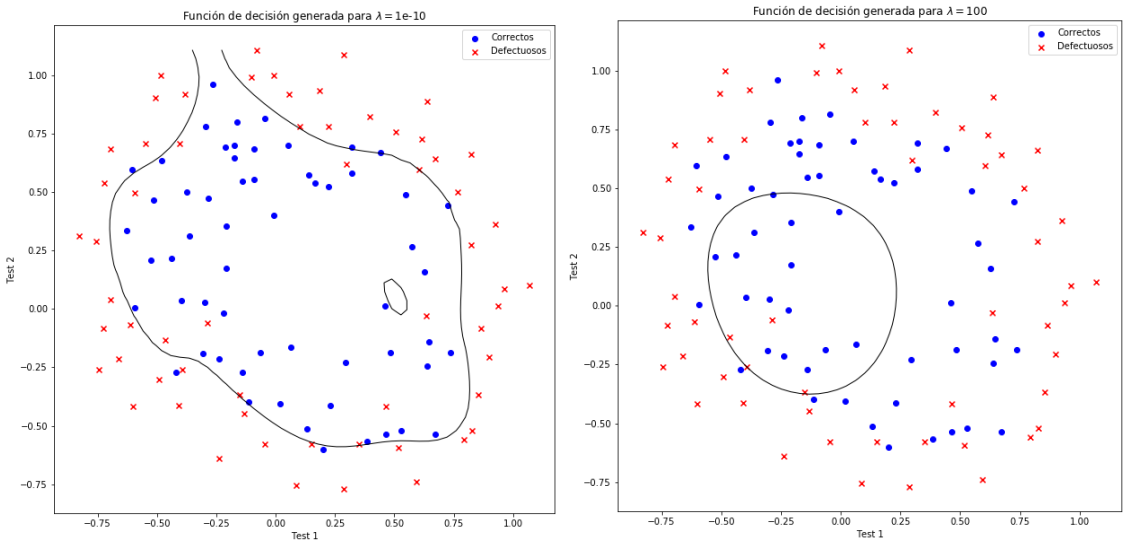

Figura 4: A la izquierda, la curva correspondiente a $\lambda = 10^{-10}$, con una forma excesivamente adaptada a los datos de entrenamiento. A la derecha, la curva para $\lambda = 100$ es demasiado pequeña como para englobar muchos de los puntos correspondientes a chips correctos.

4. Conclusiones

Esta práctica nos ha permitido comprender con gran nivel de detalle el funcionamiento de la regresión logística como método para construir modelos de predicción. Gracias a los gráficos realizados hemos podido visualizar el modelo obtenido y ver cómo se ajusta más o menos a los datos, sobre todo en la segunda parte al ir variando λ.

Además, hemos aprendido a utilizar la función *fmin_ tnc* para minimizar el coste de $J(\theta)$ una vez se tiene la función de coste y su gradiente, lo cual es mucho más cómodo que implementar el proceso como se hacía en la práctica anterior, y a utilizar la clase *PolynomialFeatures* para aumentar el número variables que tiene cada dato.

5. Anexo: Código

A continuación se ofrece el código escrito para resolver los ejercicios propuestos en la práctica.

```
'''
Práctica 2

Rubén Ruperto Díaz y Rafael Herrera Troca
'''

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import fmin_tnc
from sklearn.preprocessing import PolynomialFeatures


os.chdir("./resources")


# Función sigmoide
def sigmoide(z):
    return 1 / (1 + np.exp(-z))

def costeP1(theta, X, Y):
    gXTheta = sigmoide(np.dot(X, theta))
    factor = np.dot(np.log(gXTheta).T, Y) + np.dot(np.log(1 - gXTheta).T, 1-Y)
    return -1 / len(Y) * factor

def gradienteP1(theta, X, Y):
    gXTheta = sigmoide(np.dot(X, theta))
    return 1 / len(Y) * np.dot(X.T, gXTheta-Y)

def acierto(X, Y, theta):
    gXTheta = sigmoide(np.dot(X, theta))
    resultados = [((gXTheta >= 0.5) & (Y == 1)) | ((gXTheta < 0.5) & (Y == 0))]
    return np.count_nonzero(resultados) / len(Y)

def costeP2(theta, X, Y, lmb=1):
    gXTheta = sigmoide(np.dot(X, theta))
    factor = np.dot(np.log(gXTheta).T, Y) + np.dot(np.log(1 - gXTheta).T, 1-Y)
    return -1 / len(Y) * factor + lmb / (2 * len(Y)) * np.sum(theta**2)

def gradienteP2(theta, X, Y, lmb=1):
    gXTheta = sigmoide(np.dot(X, theta))
    thetaJ = np.concatenate(([0], theta[1:]))
```

```
    return 1 / len(Y) * np.dot(X.T, gXTheta-Y) + lmb / len(Y) *
    thetaJ


## Parte 1: Regresión con una variable


# Leemos los datos
data = pd.read_csv("ex2data1.csv", names=['exam1', 'exam2', '
    resultado'])
X = np.array([data['exam1'], data['exam2']]).T
Y = np.array(data['resultado'])

# Representamos los datos en una gráfica
adm = np.where(Y == 1)
rech = np.where(Y == 0)
plt.figure(figsize=(10,10))
plt.scatter(X[adm, 0], X[adm, 1], marker='o', c='blue', label="
    Admitidos")
plt.scatter(X[rech, 0], X[rech, 1], marker='x', c='red', label="
    Rechazados")
plt.title("Datos proporcionados para la parte 1")
plt.xlabel("Examen 1")
plt.ylabel("Examen 2")
plt.legend(loc="upper right")
plt.savefig("p1Datos.png")
plt.show()

# Calculamos el valor óptimo de theta
X2 = np.hstack((np.array([np.ones(len(Y))]).T,X))
theta0 = np.zeros(np.shape(X2)[1])
result = fmin_tnc(func=costeP1, x0=theta0, fprime=gradienteP1, args
    =(X2, Y))[0]

# Representamos la recta en una gráfica junto a los datos
part = np.linspace(min(X[:,0]), max(X[:,0]), 1000)
plt.figure(figsize=(10,10))
plt.scatter(X[adm, 0], X[adm, 1], marker='o', c='blue', label="
    Admitidos")
plt.scatter(X[rech, 0], X[rech, 1], marker='x', c='red', label="
    Rechazados")
plt.plot(part, -(result[0]+result[1]*part)/result[2], 'k', label="
    Recta de decisión")
plt.title("Recta de decisión generada para la parte 1")
plt.xlabel("Examen 1")
plt.ylabel("Examen 2")
plt.legend(loc="upper right")
plt.savefig("p1Recta.png")
plt.show()

# Comprobamos el porcentaje de aciertos de la recta obtenida
porc_ac = acierto(X2, Y, result)
print("Se tiene un " + str(porc_ac*100) + "% de aciertos")


```

```
## Parte 2


# Leemos los datos
data = pd.read_csv("ex2data2.csv", names=['test1', 'test2', '
    resultado'])
X = np.array([data['test1'], data['test2']]).T
Y = np.array(data['resultado'])

# Representamos los datos en una gráfica
corr = np.where(Y == 1)
defe = np.where(Y == 0)
plt.figure(figsize=(10,10))
plt.scatter(X[corr, 0], X[corr, 1], marker='o', c='blue', label="
    Correctos")
plt.scatter(X[defe, 0], X[defe, 1], marker='x', c='red', label="
    Defectuosos")
plt.title("Datos proporcionados para la parte 2")
plt.xlabel("Test 1")
plt.ylabel("Test 2")
plt.legend(loc="upper right")
plt.savefig("p2Datos.png")
plt.show()

# Aplicamos PolynomialFeatures para mapear los atributos y obtener
# un mejor ajuste a los ejemplos de entrenamiento
X2 = PolynomialFeatures(6).fit_transform(X)

# Calculamos el valor óptimo de theta
theta0 = np.zeros(np.shape(X2)[1])
result = fmin_tnc(func=costeP2, x0=theta0, fprime=gradienteP2, args
    =(X2, Y))[0]

# Representamos el resultado obtenido
xx1, xx2 = np.meshgrid(np.linspace(min(X[:,0]),max(X[:,0])), np.
    linspace(min(X[:,1]),max(X[:,1])))
mesh = PolynomialFeatures(6).fit_transform(np.c_[xx1.ravel(),xx2.
    ravel()])
h = sigmoide(np.dot(mesh,result))
h = h.reshape(xx1.shape)
plt.figure(figsize=(10,10))
plt.scatter(X[corr, 0], X[corr, 1], marker='o', c='blue', label="
    Correctos")
plt.scatter(X[defe, 0], X[defe, 1], marker='x', c='red', label="
    Defectuosos")
plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='k')
plt.title("Función de decisión generada para la parte 2")
plt.xlabel("Test 1")
plt.ylabel("Test 2")
plt.legend(loc="upper right")
plt.savefig("p2FuncionL1.png")
plt.show()

# Comprobamos el porcentaje de aciertos de la recta obtenida
porc_ac = acierto(X2, Y, result)
```

```
138 print("Se tiene un " + str(porc_ac*100) + "% de aciertos")
139
140 # Repetimos el proceso anterior con distintos valores de lambda
141 porc_ac = []
142 for lmb in range(-10, 3):
143     theta0 = np.zeros(np.shape(X2)[1])
144     result = fmin_tnc(func=costeP2, x0=theta0, fprime=gradienteP2,
    args=(X2, Y, 10**lmb))[0]
145
146     xx1, xx2 = np.meshgrid(np.linspace(min(X[:,0]),max(X[:,0])), np
    .linspace(min(X[:,1]),max(X[:,1])))
147     mesh = PolynomialFeatures(6).fit_transform(np.c_[xx1.ravel(),
    xx2.ravel()])
148     h = sigmoide(np.dot(mesh,result))
149     h = h.reshape(xx1.shape)
150     plt.figure(figsize=(10,10))
151     plt.scatter(X[corr, 0], X[corr, 1], marker='o', c='blue', label
    ="Correctos")
152     plt.scatter(X[defe, 0], X[defe, 1], marker='x', c='red', label=
    "Defectuosos")
153     plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='k')
154     plt.title(r"Función de decisión generada para $\lambda = $" +
    str(10**lmb))
155     plt.xlabel("Test 1")
156     plt.ylabel("Test 2")
157     plt.legend(loc="upper right")
158     plt.savefig("p2FuncionVL" + str(lmb) + ".png")
159     plt.show()
160
161     porc_ac.append(acierto(X2, Y, result)*100)
162     print("Se tiene un " + str(porc_ac[-1]) + "% de aciertos para
    lambda " + str(10**lmb))
163
164 # Representamos el porcentaje de aciertos según el valor de lambda
165 plt.figure(figsize=(10,10))
166 plt.plot(range(-10,3), porc_ac, 'ko-')
167 plt.title(r"Porcentaje de aciertos según el valor de $\lambda$")
168 plt.xlabel(r"$\lambda = 10^{x}$")
169 plt.ylabel("Porcentaje de acierto")
170 plt.savefig("AcXLambda.png")
171 plt.show()
```