Aprendizaje Automático y minería de datos: Práctica 2

Autores: *Tatiana Duarte Balvís, Miguel Mur Cortés*

1. Objetivo:

El objetivo de la práctica es aplicar los algoritmos de Regresión Logística sobre diversos datos para establecer relaciones entre sus valores y estimar resultados.

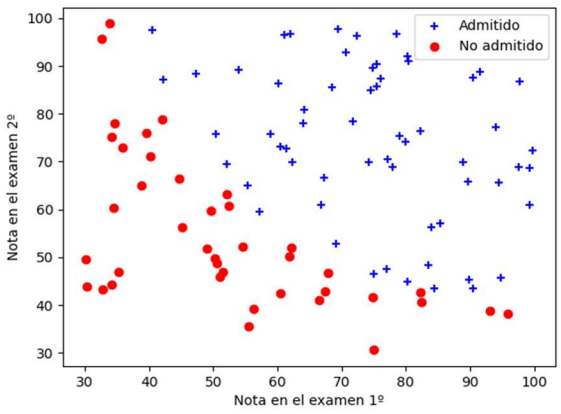
2. Regresión Logística:

Los datos representan las notas obtenidas por una serie de candidatos en los dos exámenes de admisión de una universidad junto con la información sobre si fueron (1) o no (0) admitidos.

Con los datos proporcionados, podemos generar esta figura:

```
datos = carga_csv("ex2data1.csv")
    x = datos[:, :-1]
    y = datos[:, -1]
    pos = np.where(y == 1)
    neg = np.where(y == 0)

    plt.figure()
    plt.scatter(x[pos, 0], x[pos, 1], marker="+", c="blue")
    plt.scatter(x[neg, 0], x[neg, 1], marker="o", c="red")
    plt.legend(["Admitido","No admitido"])
    plt.xlabel("Nota en el examen 1º")
    plt.ylabel("Nota en el examen 2º")
    plt.savefig("admision.png")
```



Para procesar los datos, debemos definir una función sigmoide que se pueda aplicar indistintamente sobre vectores, números o matrices.

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

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

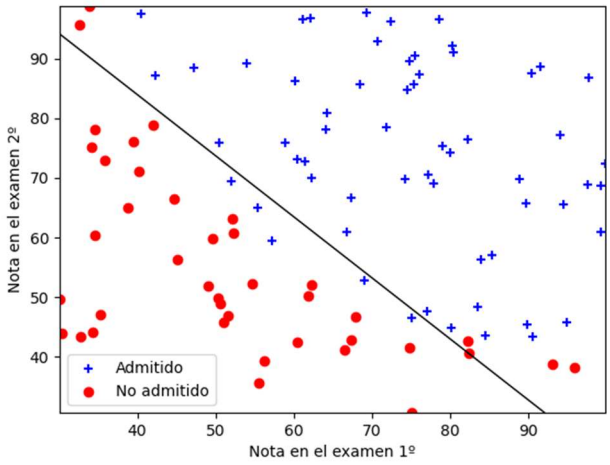
Además, se debe implementar una función de coste y otra gradiente que devuelva un vector con los valores del gradiente de la misma función para un vector de parámetros theta, un conjunto de ejemplos de entrenamiento dados en la matriz X y etiquetados con los valores del vector y

```
def coste(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    op1 = np.dot(np.log(H), y)
    op2 = np.dot(np.log(1 - H), (1 - y))
    return -(op1 + op2) / len(x)

def gradiente(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    return np.dot((H - y), x) / len(y)
```

Con las funciones definidas previamente, podemos calcular el valor θ óptimo para definir la frontera de decisión de esta forma:

```
res = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente, args=(x_aux, y),
messages=0)
theta_opt = res[0]
costeOptimo = coste(theta_opt, x_aux, y)
```



Una vez calculada la recta, hay que definir el porcentaje de ejemplos de entrenamiento que se clasifican correctamente utilizando el vector θ óptimo que se ha obtenido anteriormente.

```
def evaluacion_regresion(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    admitidos = np.mean((H >= 0.5) == y)
    return admitidos
```

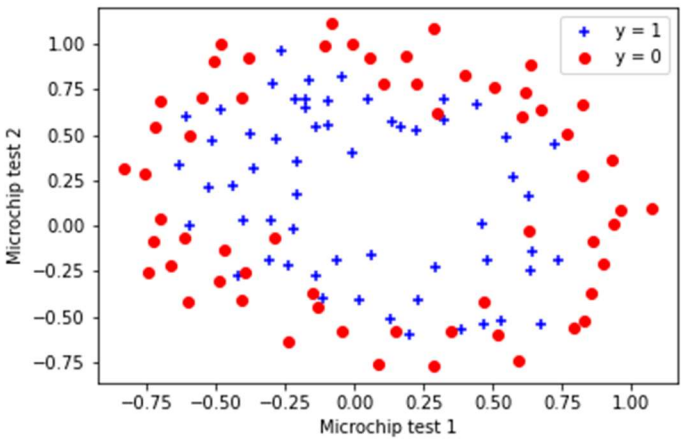
3. Regresión logística regularizada:

El objetivo de esta parte es utilizar la regresión logística regularizada para encontrar una función que pueda predecir si un microchip pasará o no el control de calidad, a partir del resultado de dos tests a los que se somete a los microchips.

Primero, se debe visualizar los datos sobre una gráfica:

```
datos = carga_csv("ex2data2.csv")
x = datos[:, :-1]
y = datos[:, -1]
pos = np.where(y == 1)
neg = np.where(y == 0)

plt.figure()
plt.scatter(x[pos, 0], x[pos, 1], marker="+", c="blue")
plt.scatter(x[neg, 0], x[neg, 1], marker="o", c="red")
plt.legend(["y = 1", "y = 0"])
plt.xlabel("Microchip test 1")
plt.ylabel("Microchip test 2")
plt.savefig("microchips.png")
```



Para obtener un mejor ajuste a los ejemplos de entrenamiento, utilizamos la clase *sklearn.preprocessing.PolynomialFeatures* para exteneder cada ejemplo de entrenamiento hasta la sexta potencia, completando hasta 28 atributos por ejemplo.

```
poly = PolynomialFeatures(6)
mapFeature = poly.fit_transform(x)
```

A continuación, de nuevo implementamos una función de coste y otra que devuelva un vector con los valores del gradiente de la misma función para la versión regularizada de la regresión logística.

```
def coste_reg(theta, x, y, lamda):
    op1 = coste(theta, x, y)
    op2 = lamda * np.sum(theta**2) / (2*len(x))
    return op1 + op2


def gradiente_reg(theta, x, y, lamda):
    op1 = gradiente(theta, x, y)
    aux = op1[0]
    op2 = lamda * theta / len(y)
    op2[0] = aux
    return op1 + op2
```

Del mismo modo que en la regresión logística sin regularizar, en la versión regularizada debemos calcular un valor óptimo θ para definir la frontera que divida a los ejemplos de entrenamiento.

Además, para apreciar mejor los ejemplos de la regularización, buscamos el valor θ óptimo para distintos valores λ.

```
lamdas = np.linspace(0.1, 10, 9)
colors = ["salmon", "darkturquoise", "purple", "gold", "orangered",
              "blueviolet", "hotpink", "limegreen", "dodgerblue"]
for lamda, color in zip(lamdas, colors): #10 ejemplos
      res = opt.fmin_tnc(func=coste_reg, x0=theta, fprime=gradi
           ente_reg, args=(mapFeature, y, lamda), messages=0)
      theta_opt = res[0]
      costeOptimo = coste_reg(theta_opt, mapFeature, y, lamda)
```