Práctica 3: Regresión logística multi-clase y redes neuronales

Álvar Domingo Fernández y Pablo Jurado López

Preparación inicial

A continuación se importan todas las librerías externas que serán utilizadas en esta práctica.

[ ]:
```
import operator
import numpy as np
import matplotlib.pyplot as plt
from numpy.lib.npyio import load
import scipy.optimize as opt
from scipy.io import loadmat
```

1 - Regresión logística multi-clase

1.1 - Visualización de los datos

Se ha cargado, con la función loadmat de scipy.io, los ejemplos de entrenamiento de la práctica en forma de diccionario, del cual se han extraído las matrices X e y (para y hemos tenido que utilizar la función .ravel(), que la convierte en una matriz unidimensional).

A continuación, haciendo uso de la librería matplot, se ha generado una imagen formada por 10 ejemplos de entrenamiento aleatorios, que en esta ocasión son números escritos a mano.

[ ]:
```
data = loadmat('ex3data1.mat')
y = data['y'].ravel()
X = data['X']

sample = np.random.choice(X.shape[0], 10)
plt.imshow(X[sample, :].reshape(-1, 20).T)
plt.axis('off')
plt.savefig('numeros.png')
```

1.2 - Clasificación de uno frente a todos

Se ha implementado una función oneVsAll que entrena un clasificador por regresión logística para todas las clases del conjunto de datos. Para ello, se hará uso de algunas funciones desarrolladas en la práctica anterior:

[ ]:
```
def sigmoide(target):
    result = 1 / (1 + np.exp(-target))
    return result


def costeReg(thetas, x, y, lamb):
    sigXT = sigmoide(np.matmul(x, thetas))
    return (-1/np.shape(x)[0]) * (np.matmul(np.transpose(np.log(sigXT)), y) + np.
 ↪matmul(np.transpose(np.log(1-sigXT)), (1-y))) + ((lamb/(2*np.shape(x)[0])) *␣
 ↪sum(thetas ** 2))


def gradienteReg(thetas, x, y, lamb):
    sigXT = sigmoide(np.matmul(x, thetas))
    return ((1/np.shape(x)[0]) * np.matmul(np.transpose(x), (sigXT - y))) +␣
 ↪((lamb/np.shape(x)[0]) * thetas)
```

A continuación, en oneVsAll entrenaremos a los clasificadores para cada una de las 10 clases

del ejemplo (con num_etiquetas = 10) haciendo uso de la función opt.fmin_tnc y las descritas anteriormente:

[ ]:
```
def oneVsAll(X, y, num_etiquetas, reg):
    clasificadores = np.zeros(shape=(10, 400))

    for i in range(1, num_etiquetas + 1):
        filtrados = (y == i) * 1
        thetas = np.zeros(np.shape(X)[1])
        clasificadores[i - 1] = opt.fmin_tnc(
            func=costeReg, x0=thetas, fprime=gradienteReg, args=(X, filtrados,␣
 ↪reg))[0]

    return clasificadores
```

Con el siguiente método se harán unas predicciones en base a los clasificadores que se han generado y se comparará cada una de ellas con el valor de y, obteniendo así el porcentaje de precisión del modelo:

[ ]:
```
def prediccion(X, Y, clasificadores):
    predicciones = {}
    Y_pred = []
    for imagen in range(np.shape(X)[0]):
        for i in range(clasificadores.shape[0]):
            theta_opt = clasificadores[i]
            etiqueta = i + 1
            prediccion = sigmoide(
                np.matmul(np.transpose(theta_opt), X[imagen]))

            predicciones[etiqueta] = prediccion
        Y_pred.append(max(predicciones.items(), key=operator.itemgetter(1))[0])
    return np.sum((Y == np.array(Y_pred)))/np.shape(X)[0] * 100
```

En este caso, la precisión resultante es de un 95,88%

2 - Redes Neuronales

Hemos cargado el fichero ex3weights.mat, a partir del cual se han obtenido las matrices Theta1 y Theta2. A partir de ellas se ha implementado la función que ejecuta la propagación hacia delante de la red y devuelve el valor de a_3, o h_0, para cada uno de los ejemplos:

[ ]:
```
weights = loadmat('ex3weights.mat')
data = loadmat('ex3data1.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']

def propagar(X, theta1, theta2):
    m = np.shape(X)[0]

    a1 = np.hstack([np.ones([m, 1]), X])
```

```
    z2 = np.dot(a1, theta1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoide(z2)])
    z3 = np.dot(a2, theta2.T)
    a3 = sigmoide(z3)
    return a3
```

Para obtener la precisión de los resultados de la red neuronal, también se ha implementado una función que genera predicciones en base al resultado de la propagación:

[ ]:
```
def prediccion_neuronal(X, a3):
    return [(np.argmax(a3[imagen]) + 1) for imagen in range(X.shape[0])]
```

Y finalmente, siguiendo el mismo procedimiento que en el apartado 1, se ha obtenido el porcentaje de aciertos del modelo:

[ ]:
```
h = propagar(X, theta1, theta2)
Y_pred = prediccion_neuronal(X, h)
precision = np.sum((y == np.array(Y_pred))) / np.shape(X)[0]
print("La precisión de la red neuronal es de ", precision * 100)
```

En este caso, la precisión de los resultados ha sido de un 97,52%