Practica 3

Antonio Rodriguez Hurtado y Miguel Ferreras Chumillas

Regresion logistica

Importamos libreria, definimos las variables necesarias y cargamos y los datos

```
#   Librerias
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt


#   Extraemos los ejemplos de entrenamiento del archivo que los contiene
data = loadmat("ex3data1.mat")

#   Separamos los atributos de los ejemplos y sus etiquetas (El 0 viene etiquetado como 10)
y = data['y']
X = data['X']

#   Obtenemos las etiquetas unicas
etiquetas = np.unique(y)

#   Obtenemos el numero de ejemplos de entrenamiento, sus numeros de pixeles (m) y las etiquetas
num_ejemplos = np.shape(X)[0]
m = np.shape(X)[1]
num_etiquetas = etiquetas.size

#   Añadimos una columna de unos en X para facilitar los calculos con matrices
OX = np.hstack([np.ones([num_ejemplos, 1]), X])

#   Definimos nuestra theta
theta = np.zeros(m +  1)

#   Definimoos nuestra termino de regularizacion
lamda = 0.1

# Seleccionamos aleatoriamente 10 ejemplos y los pintamos
sample = np.random.choice(X.shape[0], 10)
plt.imshow(X[sample, :].reshape(-1, 20).T)
plt.axis('off')
```

Ejemplo de los datos:

Modificamos las funciones de la practica anterior y definimos las necesarias para entrenar a los clasificadores

```
#   REGRESIÓN LOGÍSTICA
#   Definimos la funcion sigmoide
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

#   Defininmos la funcion de hipotesis
def hip(a, b):
    return sigmoid(np.matmul(a, b))

#   Definimos la funcion de coste
def cost(theta, X, Y):
    Y = np.ravel(Y)
    return (-1 / m) * (np.dot(Y, np.log(hip(X, theta))) + np.dot((1 - Y), np.log(1 - hip(X, theta) + 1e-6))) + (lamda / (2 * m)) * np.

#   Definimos la funcion de gradiente
def gradient(theta, X, Y):
    gradiente =  ((1 / m) * np.matmul((hip(X, theta) - np.ravel(Y)), X))
    result = gradiente[0]
    i = 1
    for e in gradiente[1:]:
```

```
        result = np.append(result, e + ((lamda / m) * theta[i]))
        i += 1
    return result


#   ENTRENAMIENTO DE CLASIFICADORES
#   Definimos la función de entrenamiento para los clasificadores
def oneVsAll(X, Y, etiquetas, reg):
    result = []
    for i in etiquetas:
        aux = (y == i) * 1
        op = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(OX, aux), disp=False)
        result.append(op[0])
    return result
```

Red neuronal

Definimos la funcion de la red

```
#   RED NEURONAL
#   Obtenemos las matrices de pesos del archivo
weights = loadmat('ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']

#   Definimos el comportamiento de nuestra red neuronal
def redNeuronal(x):
    oculta = hip(theta1, x)
    outputIn = np.hstack([[1], oculta])
    outputOut = hip(theta2, outputIn)
    result = np.where(outputOut == np.max(outputOut))[0] + 1
    return result
```

Comparativa

```
#  Calculamos el porcentaje de aciertos de los clasificadores
def porcentajeClasificadores(X , Y):
    theta_opt = oneVsAll(X, Y, etiquetas, lamda)
    i = 0
    ok = 0
    for x in X:
        h = hip(theta_opt, x)
        etiqueta = np.where(h == np.max(h))[0] + 1
        if(Y[i] == etiqueta):
            ok +=1
        i += 1
    return (ok / num_ejemplos) * 100

#   Calculamos el porcentaje de aciertos con la red neuronal
def porcentajeRed(X, Y):
    i = 0
    ok = 0
    for x in X:
        if(Y[i] == redNeuronal(x)):
            ok +=1
        i += 1
    return (ok / num_ejemplos) * 100

print("Porcentaje de aciertos con red neuronal: " + str(porcentajeRed(OX, y)) + "%")
print("Porcentaje de aciertos con clasificadores: " + str(porcentajeClasificadores(OX, y)) + "%")
```

```
Porcentaje de aciertos con red neuronal: 97.52%
Porcentaje de aciertos con clasificadores: 96.46000000000001%
```

Se puede ver como el porcentaje de aciertos con la red es ligeramente superior