Práctica 3

Andrés de la Cuesta López
Pablo Rodríguez-Bobada García-Muñoz

➢ Práctica 3.1:

En la práctica 3.1 aprendemos a usar el método de regresión logística con varias clases. Al tener varias clases, utilizamos la matriz de thetas en el que cada fila tiene los pesos para predecir la clase correspondiente. En nuestro caso de imágenes, ThetaMat[1] tendría los pesos para predecir si la imagen que procesamos es 1. Utilizamos el método de “one for all” para entrenar nuestro modelo y obtener los vectores de theta óptimos para cada clase. Este método consiste en aplicar regresión logística regularizada para obtener éstos clasificadores. Una vez obtenida la matriz, comprobamos el porcentaje de éxito obtenido. Para ello comprobamos cuál es la imágen más probable que nuestro modelo predice y si coincide con su valor real. Obtenemos un valor del 96,14%.

```
#%%
import numpy as np
import matplotlib.pyplot as plt
from numpy.core.fromnumeric import size
import scipy.optimize as opt
import math
import sklearn.preprocessing as sp
from pandas.io.parsers import read_csv
from scipy.io import loadmat

#ir caso a caso de X viendo que predice cada theta y quedarnos con el
mayor y luego comprobar si es realmente ese
#por los 5000 casos, probar si es 0,1,2...9, ver cual es mayor y lo
guardamos, luego comprobar si es cierto

# Calcula la funcion sigmoide
def sigmoide(z):
    return 1/(1+math.e**(-z))

#  Calcula el coste (regularizado)
def coste(theta, X, Y ,_lambda):
    H = sigmoide(np.matmul(X,theta))
    m = np.shape(X)[0]
    # usamos la formula con las traspuestas
    coste = (- 1 / (len(X))) * np.sum( Y * np.log(H) + (1 - Y) *
np.log(1 - H + 1e-6) )
```

```
    regularizacion = (_lambda /(2+m) ) * np.sum(np.power(theta,2))
    coste += regularizacion
    return coste

#  Calcula el gradiente (regularizado)
def gradiente(Theta, X, Y, _lambda):
    H = sigmoide(np.matmul(X, Theta))
    m = np.shape(X)[0]
    return (1/m) * np.matmul(np.transpose(X), H - Y) + ((_lambda / m )
* Theta)

# Calcula el porcentaje de éxito de nuestro modelo multi-clase
def evaluacion(X,Y,thetaMat):
    # X (5000x401) thetaMat(t) (401x10)
    hMat = np.matmul(X,np.transpose(thetaMat))  # matriz de 5000x10 con
las predicciones de cada theta
    nCases = np.shape(hMat)[0]
    MaxIndex = np.zeros(nCases)
    for i in range(nCases):
        MaxIndex[i]=np.argmax(hMat[i])
        if(MaxIndex[i]==0):
            MaxIndex[i]=10
    Num= np.sum(np.ravel(Y) == MaxIndex)
    Porcentaje=Num/nCases * 100
    print(Porcentaje)

def oneVsAll(X, y, num_etiquetas, reg):
    """
    oneVsAll entrena varios clasificadores por regresion logistica con
termino de regularizacion
    'reg' y devuelve el resultado en una matriz, donde la fila i-esima
correspone
    al clasificador de la etiqueta i-esima
    """
    porcentaje = np.zeros(10)
    theta_mat = np.zeros((num_etiquetas, np.shape(X)[1] + 1))
    m = np.shape(X)[0]
    X1s = np.hstack([np.ones([m, 1]), X])
    Theta = np.zeros(np.shape(X1s)[1])
    for i in range(num_etiquetas):
        #el numero que queremos buscar, 1s 2s 3s...
        y_i = (y==i) * 1
        if(i == 0):#el caso 0 son 10s, voltearlo
```

```
            y_i = (y==10) * 1
        y_i = np.ravel(y_i)
        print(np.shape(y_i))
        result = opt.fmin_tnc(func=coste, x0 = Theta, fprime =
gradiente, args=(X1s, y_i, reg))
        theta_mat[i] = result[0]


    return theta_mat

def main():
    #1.1 mostrar numeritos
    data = loadmat('ex3data1.mat')
# se pueden consultar las claves con data.keys( )
    y = data [ 'y' ]
    X = data [ 'X' ]
 # almacena los datos leídos en X, y
    sample = np.random.choice(X.shape[0],10)
    plt.imshow(np.transpose(np.reshape(X[sample, :],[-1,20])))
    plt.axis('off')
    #plt.show()

    m = np.shape(X)[0]
    X1s = np.hstack([np.ones([m, 1]), X])

    #1.2 Clasificacion de uno frente a todos
    evaluacion(X1s,y,oneVsAll(X,y, 10, 0.1))


main()
# %%
```

➢ Práctica 3.2:

En la práctica 3.2 utilizamos unos pesos proporcionados sobre una red neuronal ya entrenada con el objetivo de evaluar la precisión de dicha red sobre los mismos ejemplos. Después de hacer la propagación en la red neuronal, obtenemos una precisión del 97.52%. Esta precisión es más alta que en el apartado anterior, lo que quiere decir que los pesos proporcionados por “ex3weights.mat” son mejores que los calculados por nosotros.

```
#%%
import numpy as np
import matplotlib.pyplot as plt
from numpy.core.fromnumeric import size
import scipy.optimize as opt
import math
import sklearn.preprocessing as sp
from pandas.io.parsers import read_csv
from scipy.io import loadmat

# Calcula la funcion sigmoide
def sigmoide(z):
    return 1/(1+math.e**(-z))



# Calcula el porcentaje de éxito de nuestro modelo multi-clase
def evaluacion(y, a_3):
    MaxIndex=np.zeros(np.shape(a_3)[0])

    for i in range(np.shape(a_3)[0]):
        MaxIndex[i]=np.argmax(a_3[i]) + 1#mas uno por como esta
ordenado,el 0 equivale a 1... el 9 a un 10

    Num= np.sum(np.ravel(y) == MaxIndex)
    Porcentaje=Num/np.shape(y)[0] * 100
    print(Porcentaje)

def main():
    data = loadmat('ex3data1.mat')
    y = data [ 'y' ] # 5000, 1
    X = data [ 'X' ] # 5000, 400

    weigths = loadmat('ex3weights.mat')
    theta1 = weigths['Theta1'] # 25 x 401
    theta2 = weigths['Theta2'] # 10 x 26
```

```
    # Propagación
    a_1 = np.hstack([np.ones([np.shape(X)[0], 1]), X]) # 5000 x 401
    z_2 = np.matmul(a_1, np.transpose(theta1)) # 5000 x 25
    a_2=sigmoide(z_2)
    a_2 = np.hstack([np.ones([np.shape(a_2)[0], 1]), a_2]) # 5000 x 26
    z_3 = np.matmul(a_2, np.transpose(theta2)) # 5000 x 10
    a_3=sigmoide(z_3)

    evaluacion(y, a_3)

main()
# %%
```