Aprendizaje automático y minería de datos: Práctica 3

Jorge Rodríguez García y Gonzalo Sanz Lastra

Código de la práctica:

```python
import numpy as np
import scipy.optimize as opt            # para la funcion de gradiente
from matplotlib import pyplot as plt    # para dibujar las graficas
from scipy.io import loadmat

def load_mat(file_name):
    """carga el fichero mat especificado y lo devuelve en una matriz data"""
    return loadmat(file_name)

def graphics(X):
    """Selecciona aleatoriamente 10 ejemplos y los pinta"""
    sample = np.random.choice(X.shape[0], 10)
    plt.imshow(X[sample, :].reshape(-1, 20).T)
    plt.axis('off')
    plt.show()

# g(X*Ot) = h(x)
def sigmoid(Z):
    return 1/(1+np.exp(-Z))

def cost(O, X, Y, l):
    """devuelve un valor de coste"""
    return (-(((np.log(sigmoid(X.dot(O)))).T.dot(Y) + (np.log(1-sigmoid(X.dot(O)))).T.dot(1-Y))/X.shape[0])
    + (l/(2*X.shape[0]))*(O[1:,]**2).sum()) # igual que el coste anterior pero añadiendole esto ultimo para la regularizacion

def gradient(O, X, Y, l):
    """la operacion que hace el gradiente por dentro -> devuelve un vector de valores"""
    AuxO = np.hstack([np.zeros([1]), O[1:,]]) # sustituimos el primer valor de las thetas por 0 para que el termino independiente
    # no se vea afectado por la regularizacion (lambda*0 = 0)
    return (((X.T.dot(sigmoid(X.dot(O))-np.ravel(Y)))/X.shape[0])
    + (l/X.shape[0])*AuxO) # igual que el gradiente anterior pero añadiendole esto ultimo para la regularizacion

def oneVsAll(X, Y, num_etiquetas, reg):
    """para cada ejemplo (fila de Xs), haya los pesos theta por cada posible tipo de número que pueda ser"""
    clase = 10                          # la primera clase a comprobar sera el numero 0
    O = np.zeros([num_etiquetas, X.shape[1]]) # un vector de thetas aprendidas (fila) por cada clase de numero a comprobar (columna)

    for i in range(num_etiquetas):
        claseVector = (Y == clase) # vector con 1s si es de la clase a comprobar y 0s el resto

        result = opt.fmin_tnc(func = cost, x0 = O[i], fprime = gradient, args=(X, claseVector, reg))
        O[i] = result[0] # entre todos los resultados devueltos, este ofrece las thetas optimas

        if clase == 10: clase = 1
        else: clase += 1
    return O

def logisticSuccessPercentage(X, Y, O):
    """determina el porcentaje de aciertos de la regresión logística multicapa comparando los resultados estimados con los resultados reales"""
```

```
    numAciertos = 0

    for i in range(X.shape[0]):
        results = sigmoid(X[i].dot(O.T))
        maxResult = np.argmax(results)
        if maxResult == 0: maxResult = 10
        if maxResult == Y[i]: numAciertos += 1

    return (numAciertos/(X.shape[0]))*100

def neuronalSuccessPercentage(results, Y):
    """determina el porcentaje de aciertos de la red neuronal comparando los resultados estimados con los resultados reales"""
    numAciertos = 0

    for i in range(results.shape[0]):
        result = np.argmax(results[i]) + 1
        if result == Y[i]: numAciertos += 1
    return (numAciertos/(results.shape[0]))*100

def propagacion(X1, O1, O2):
    """propaga la red neuronal a traves de sus dos capas"""
    X2 = sigmoid(X1.dot(O1.T))
    X2 = np.hstack([np.ones([X2.shape[0], 1]), X2])
    return sigmoid(X2.dot(O2.T))

def main():
    # REGRESION LOGISTICA MULTICAPA
    valores = load_mat("ex3data1.mat")

    X = valores['X'] # matriz X, con todas las filas y todas las columnas menos la ultima (ys)
    Y = valores['y'] # matriz Y, con todas las filas y la ultima columna

    m = X.shape[0]      # numero de muestras de entrenamiento
    n = X.shape[1] # numero de variables x que influyen en el resultado y, mas la columna de 1s
    num_etiquetas = 10

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

    l = 0.1 # cuanto mas se aproxime a 0, mas se ajustara el polinomio (menor regularizacion)
    O = oneVsAll(X, Y, num_etiquetas, l)

    success = logisticSuccessPercentage(X, Y, O)
    print("Logistic regression success: " + str(success) + " %")

    # REDES NEURONALES
    weights = load_mat('ex3weights.mat')
    O1, O2 = weights['Theta1'], weights['Theta2']

    success = neuronalSuccessPercentage(propagacion(X, O1, O2), Y)
```

```
def main():
    # REGRESION LOGISTICA MULTICAPA
    valores = load_mat("ex3data1.mat")

    X = valores['X'] # matriz X, con todas las filas y todas las columnas menos la ultima (ys)
    Y = valores['y'] # matriz Y, con todas las filas y la ultima columna

    m = X.shape[0]      # numero de muestras de entrenamiento
    n = X.shape[1] # numero de variables x que influyen en el resultado y, mas la columna de 1s
    num_etiquetas = 10

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

    l = 0.1 # cuanto mas se aproxime a 0, mas se ajustara el polinomio (menor regularizacion)
    O = oneVsAll(X, Y, num_etiquetas, l)

    success = logisticSuccessPercentage(X, Y, O)
    print("Logistic regression success: " + str(success) + " %")

    # REDES NEURONALES
    weights = load_mat('ex3weights.mat')
    O1, O2 = weights['Theta1'], weights['Theta2']

    success = neuronalSuccessPercentage(propagacion(X, O1, O2), Y)
    print("Neuronal network success: " + str(success) + " %")

    #GRAFICAS
    graphics(X[:, 1:])

main()
```