Práctica 3 – Regresión logística multi-clase y redes neuronales

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

En esta práctica aplicamos la regresión logística multi-clase y usamos redes neuronales.

Tenemos dos archivos. El primero: "ex3data.1.mat" son 5.000 ejemplos de entrenamiento que son imágenes de 20x20 que han sido desplegadas en un vector de 400 componentes. Estas imágenes son números escritos a mano que tendremos que clasificar entrenando una red neuronal.

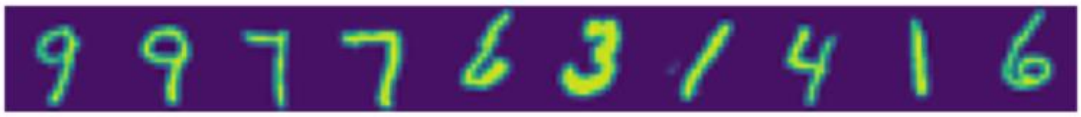
El segundo archivo "ex3weights.mat" contiene dos matrices de pesos que son resultados de haber entrenado una red neuronal. Con este archivo implementaremos la red neuronal y calcularemos su precisión, que debería estar alrededor de 97.5%.

- Código de la práctica
  Primera parte:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt
import sklearn.preprocessing


def sigmoid(X):
    # x > 50 returns 1
    return 1 / (1 + np.exp(-X))


def costRegression(T, XX, Y, l):
    m = Y.size
    H = sigmoid(XX.dot(T))
    l1 = np.transpose(np.log(H))
    l2 = np.transpose(np.log(1 - H + 1e-6))

    ret = (-1 / m) * ((np.matmul(l1, Y)) + (np.matmul(l2, (1 - Y))))
    return ret + l / (2 * m) * np.sum(H * H)


def gradiente(theta, X, Y, l):
    m = np.shape(X)[0]
    H = sigmoid(X.dot(theta))
    ret = (1 / m) * np.matmul(np.transpose(X), H - Y)
    return ret + (l / m) * theta
```

```
    def oneVsAll(X, Y, num_etiquetas, l):
        result = []
        mapFeatureX = X
        T = np.zeros(mapFeatureX.shape[1])

        for i in range(0, num_etiquetas + 1):
            YY = (Y == i) * 1
            result.append(opt.fmin_tnc(func = costRegression, x0 = T, fprime
    = gradiente,
                args = (mapFeatureX, YY, l))[0])
        return result, mapFeatureX


    def calcularAciertos(X, Y, T):
        aciertos = 0
        j = 0
        tags = len(T)
        for i in range(len(X)):
            maxi = sys.float_info.min
            for tag in range(0, tags):
                pred = sigmoid(X[i].dot(T[tag]))
                if(pred > maxi):
                    p = tag
                    maxi = pred

            if Y[i] == p:
                aciertos += 1
            j += 1
        return aciertos / len(Y) * 100


    def main():
        data = loadmat('./ex3data1.mat')
        X = data['X']
        Y = data['y']
        Y = np.ravel(Y)

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

        X = np.hstack([np.ones([np.shape(X)[0], 1]), X])
        l = .1
        tags = 10
        result, X = oneVsAll(X, Y, tags, l)
        print(calcularAciertos(X, Y, result))


main()
```

Primero cargamos correctamente una imagen de ejemplos cogidos aleatoriamente:



Tras ejecutar el resto del código (entrenando para cada número del 0 al 10) obtenemos la siguiente precisión:

```
96.48

D:\Users\Juan\Desktop\Practicas-Aprendizaje-Automatica-y-Big-Data-main\Practica 3>
```

Segunda parte:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt
import sklearn.preprocessing


def sigmoid(X):
    return 1 / (1 + np.exp(-X))


def hypothesis(X, T):
    return sigmoid(np.matmul(T.T, X))


def nn(X, theta1, theta2):
    a1 = X
    a2 = np.matmul(theta1, a1.T)
    a2 = sigmoid(a2).T
    a2 = np.hstack([np.ones([np.shape(a2)[0], 1]), a2])
    a3 = np.matmul(theta2, a2.T)
    a3 = sigmoid(a3).T

    return a3


def calcularAciertos(X, Y, H):
    aciertos = 0
    for row in range(len(H)):
        guess = np.argmax(H[row]) + 1
        if guess == Y[row]:
            aciertos += 1

    return aciertos / len(H) * 100


def main():
    data = loadmat('./ex3data1.mat')
    X = data['X']
    Y = data['y']
    Y = np.ravel(Y)
```

```
        weights = loadmat('./ex3weights.mat')
        theta1, theta2 = weights['Theta1'], weights['Theta2']
        X = np.hstack([np.ones([np.shape(X)[0], 1]), X])

        result = nn(X, theta1, theta2)
        print(calcularAciertos(X, Y, result))


main()
```

En esta segunda parte cargamos los pesos de la red neuronal y calculamos el porcentaje de aciertos según los ejemplos proporcionados:

```
D:\Users\Juan\Desktop\Practicas-Aprendizaje-Automatica-y-Big-Data-main\Practica 3>Practica3.2.py
97.52
```

- Conclusiones

En la primera parte de la práctica aprendemos como es bastante sencillo crear el clasificador de dígitos y como además obtenemos una precisión bastante buena teniendo en cuenta lo simple que es la función de la hipótesis.

En la segunda parte vemos cómo se puede almacenar y cargar los datos de una red neuronal ya entrenada y como esta puede tener un muy buen porcentaje de aciertos.