Practica 3: Regresión Logística multi-clase y redes neuronales

Nuria Bango Iglesias (nubango@ucm.es)

Álvar Julián de Diego López (alvarded@ucm.es)

1. Regresión logística multi-clase

En esta práctica vamos a aplicar regresión logística multi-clase al reconocimiento de imágenes que representan números escritos a mano.

1.1. Visualización de los datos

Primero cargamos los datos con loadmat y extraemos las matrices X e Y

```
valores = loadMat("ex3data1.mat")

X = valores['X']
Y = valores['y']
```

A continuación vamos a visualizar 10 elementos aleatorios de los ejemplos de entrenamiento

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

1.2. Clasificación de uno frente a todos

Ahora vamos a entrenar un clasificador para cada una de las 10 clases del conjunto de datos.

```
def oneVsAll(X, Y, numEtiquetas, reg):
    clase = 10
    Theta = np.zeros([numEtiquetas, X.shape[1]])

    for i in range(numEtiquetas):
        claseVector = (Y == clase)

        result = opt.fmin_tnc(func = cost, x0 = Theta[i], fprime = gradient, args = (X, claseVector, reg))
        Theta[i] = result[0]

        if clase == 10:
            clase = 1
        else:
            clase += 1
    return Theta
```

Para comprobar que el código es correcto, probamos con una regularización de 0,1 que debería dar un 95% de precisión.

```
l = 0.1
Theta = oneVsAll(X, Y, numEtiquetas, l)

success = logisticSuccessPercentage(X, Y, Theta)
print("Precisión de la regresión logística: " + str(success) + " %")
```

```
Precisión de la regresión logística: 94.17999999999999 %
```

2. Redes neuronales

En esta segunda parte de la práctica vamos a usar los pesos proporcionales ya entrenados para una red neuronal ya entrenada sobre los ejemplos para evaluar su precisión sobre esos mismos ejemplos.

Para ello implementamos la propagación hacia delante para computar el valor de hθ(x(i)) para cada ejemplo de i.

```
def propagacion(X1, Theta1, Theta2):
    X2 = sigmoid(X1.dot(Theta1.T))
    X2 = np.hstack([np.ones([X2.shape[0], 1]), X2])
    return sigmoid(X2.dot(Theta2.T))
```

Para comprobar que el código es correcto vamos a calcular la precisión, que nos debería dar entorno al 97.5%

```python
def neuronalSuccessPercentage(results, Y):
    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
```

```
Precisión de la red neuronal: 97.52 %
```