PRÁCTICA 3 - REGRESIÓN LOGÍSTICA MULTI-CLASE Y REDES NEURONALES

Autores:
- Amaro Blest Polo
- Raúl Blas Ruiz

1.1 Visualización de los datos:

El primer paso de la práctica fue visualizar los datos mediante la extracción de los datos usando la función loadmat de la librería scipy.io, la cual nos devuelve un diccionario donde podemos ver sus keys haciendo uso de data.keys(). Al hacer esto podemos observar que nuestro diccionario contaba con varias keys, entre ellas X e y, que son las que a nosotros nos interesan.

Cada ejemplo de entrenamiento es una imagen de 20x20 píxeles donde cada píxel está representado por un número real que indica la intensidad en escala de grises de ese punto. Cada matriz de 20x20 se ha desplegado para formar un vector de 400 componentes que ocupa una fila de la matriz X. De esta forma, X es una matriz de 5000x400 donde cada fila representa la imagen de un número escrito a mano.

El vector y es un vector de 5000 componentes que representan las etiquetas de los ejemplos de entrenamiento. El “0” se ha etiquetado como “10”, manteniendo las etiquetas naturales del “1” al “9” para el resto de los números.

Para la visualización de estos datos realizamos un random que escoge 10 casos entre los 5000 que hay y los pinta haciendo uso de plt.imshow().

```
# El fichero se carga con la función scipy.io.loadmat que devuelve un diccionario del que podemos extraer las matrices X y
data = loadmat('ex3data1.mat')

# X e Y son matrices. Cada matriz de 20x20 se ha desplegado para formar un vector de 400 componentes que ocupa una
# fila de la matriz X. X es una matriz de 5000x400 donde cada fila representa la imagen de un número escrito a mano.
# almacena los datos leídos en X, y
y = data['y']
X = data['X']

num_labels = 10
regularization = 0.1
oneVsAll(X, y, num_labels, regularization)

# se pueden consultar las claves del diccionario con data.keys()
print(data.keys())
# 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')
#endregion
```

1.2 Clasificación de uno frente a todos:

En este apartado realizamos la implementación del método oneVsAll, el cual utilizamos para entrenar clasificadores utilizando la regresión logística con término de regularización. Para ello lo que hacemos es recorrer el número de etiquetas para comprobar a qué clase pertenece cada una y accedemos a su label sub i. Sin embargo, la etiqueta 0 sabemos que corresponde al número 10 es por ello que al final del bucle accedemos bien a su etiqueta.

A continuación, lo que hacemos es acceder a la theta óptima de cada etiqueta para así aplicar la regresión logística.

Por último hacemos una evaluación de los resultados obtenidos haciendo uso del método evaluate, el cual nos devuelve el porcentaje de la evaluación usando la función sigmoide y comparando el resultado obtenido.

```python
def oneVsAll(X, y, n_labels, reg):
    """
    oneVsAll entrena varios clasificadores por regresión logística con término
    de regularización 'reg' y devuelve el resultado en una matriz, donde
    la fila i-ésima corresponde al clasificador de la etiqueta i-ésima
    """
    m = X.shape[1]
    theta = np.zeros((n_labels, m))
    yLabels = np.zeros((y.shape[0], n_labels))

    # recorremos el numero de etiquetas y vemos si yj ∈ 0, 1 indica si el ejemplo de entrenamiento j-ésimo
    # pertenece a la clase k (yj = 1) o a otra clase (yj = 0)
    # Al final cambiamos la etiqueta del 0 que corresponde a 10
    for i in range(n_labels):
        yLabels[:,i] = getLabel(y, i)
    yLabels[:,0] = getLabel(y, 10)

    # ahora lo que hacemos es coger la theta optima para los valores anteriores pero con el valor de regularizacion
    # tal y como haciamos en la regresion logistica
    for i in range(n_labels):
        theta_opt = opt.fmin_tnc(func=cost, x0=theta[i,:], fprime=gradient, args=(X, yLabels[:,i], reg))
        theta[i, :] = theta_opt[0]

    # a continuacion evaluamos los resultados
    evaluation = np.zeros(n_labels)
    for i in range(n_labels):
        evaluation[i] = evaluate(theta[i,:], X, yLabels[:,i])

    print("Evaluacion: ", evaluation)
    print("Evaluacion media: ", evaluation.mean())
    return 0
```

```
Evaluacion:  [0.9806 0.9778 0.9368 0.9332 0.9398 0.8984 0.9674 0.9664 0.9124 0.9006]
Evaluacion media:  0.9413400000000001
```

```python
#region ######################## 1.2 Clasificación de uno frente a todos ########################
def func_sigmoid(X):
    return 1.0/(1.0 + np.exp(-X))

def cost(theta, X, Y, reg):
    h = func_sigmoid(np.matmul(X, theta))
    m = X.shape[0]
    coste = (- 1 / m) * (np.dot(Y, np.log(h)) +
                         np.dot((1 - Y), np.log(1 - h))) + ((reg/2*m) * np.sum(np.power(theta[1:],2)))
    return coste

def gradient(theta, X, Y, reg):
    m = X.shape[0]
    h = func_sigmoid(np.matmul(X, theta))
    Taux = theta
    Taux[0] = 0
    aux = (1/m) * np.matmul(X.T, (np.ravel(h) - Y))
    return aux + ((reg/m) * Taux)

def getLabel(Y, etiqueta):
    y_etiqueta = np.ravel(Y)== etiqueta
    y_etiqueta = y_etiqueta *1
    return y_etiqueta
```

```python
def evaluate(theta, X, Y):
    xs = func_sigmoid(np.matmul(X, theta))
    xspositivas = np.where(xs >= 0.5)
    xsnegativas = np.where(xs < 0.5)
    xspositivasexample = np.where (Y == 1)
    xsnegativasexample = np.where (Y == 0)

    porcentajePositivas = np.intersect1d(xspositivas, xspositivasexample).shape[0]/xs.shape[0]
    porcentajeNegativas = np.intersect1d(xsnegativas, xsnegativasexample).shape[0]/xs.shape[0]

    return porcentajeNegativas + porcentajePositivas
```

2. Redes neuronales

En este último apartado de la práctica utilizamos los pesos proporcionados, en el archivo ex3weights.mat, para una red neuronal ya entrenada sobre los ejemplos para evaluar su precisión sobre esos mismos ejemplos.

La red neuronal está formada por tres capas, con 400 unidades en la primera capa (además de la primera fijada siempre a +1), 25 en la capa oculta y 10 en la capa de salida. La red neuronal está entrenada de forma que la primera neurona de salida se activa cuando reconoce un 1, la segunda cuando reconoce un 2, y así sucesivamente hasta la décima que se activa cuando reconoce un 0.

Sabiendo esto realizamos un estudio para hallar la precisión de la red neuronal. Para ello utilizamos la función de argmax de numpy para acceder al valor máximo devuelto por nuestra función h y comprobando si este + 1 corresponde al valor de y[i], si es así es que la red neuronal ha acertado.

Finalmente, escribimos el porcentaje de aciertos en la terminal, 97.52%.

```
#region ######################### 2. Redes neuronales #########################
thetas = loadmat("ex3weights.mat")
print(thetas.keys())
thetas1 = thetas["Theta1"] # Theta1 es de dimensión 25 x 401
thetas2 = thetas["Theta2"] # Theta2 es de dimensión 10 x 26

aux = np.zeros(num_labels)

for i in range(num_labels):
    aux[i] = np.argmax(h(X[sample[i],:],thetas1,thetas2))

print("My guess are: ", (aux)+1)
numAciertos = 0

for i in range(X.shape[0]):
    aux2 = np.argmax(h(X[i,:],thetas1, thetas2))
    if(aux2+1) == y[i]:
        numAciertos +=1

print("Porcentaje de aciertos: ", numAciertos / X.shape[0])
#endregion
```

```
My guess are:  [6. 3. 7. 7. 3. 5. 4. 2. 3. 2.]
Porcentaje de aciertos:  0.9752
```

```
#region ######################### 2. Redes neuronales #########################
def h(X, thetas1, thetas2):
    # primera capa de la red
    a1 = X
    # segunda capa de la red
    z2 = np.matmul(thetas1, np.insert(a1,0,1))
    a2 = func_sigmoid(z2)
    # tercera capa de la red
    z3 = np.matmul(thetas2, np.insert(a2,0,1))
    a3 = func_sigmoid(z3)
    return a3
#endregion
```