P3 - Nestor Cabrero, Marcos Garcia

January 26, 2020

1 Práctica 3: Regresión logística multi-clase y redes neuronales

En esta práctica hemos implementado el algoritmo de regresión logística multi-clase y un primer acercamiento a una red neuronal básica.

1.1 Regresión logística multi-clase

El objetivo de esta primera parte de la práctica es aplicar regresión logística multi-clase al reconocimiento de imágenes que representan números escritos a mano. Para ello vamos a utilizar los siguientes módulos:

[1]:
```
import numpy as np
import scipy.optimize as opt
from scipy.io import loadmat
from sklearn.preprocessing import PolynomialFeatures

from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

1.1.1 Visualización de los datos

Utilizamos una función auxiliar para cargar los datos de *ex3data1.mat* y visualizamos una selección aleatoria de los mismos para comprobar que se han cargado adecuadamente:

[3]:
```
def load_file(filename):
    data = loadmat(filename)
    return data['X'], data['y']
```

[6]:
```
def first_test():
    X, y = load_file('ex3data1.mat')
    sample = np.random.choice(X.shape[0], 10)
    plt.imshow(X[sample, :].reshape(-1, 20).T)
    plt.axis('off')
```

[7]:
```
first_test()
```

1.1.2 Vectorización de la regresión logística

El siguiente paso ha consistido en vectorizar las funciones de coste y gradiente que habíamos utilizado previamente. Además, obtuvimos las versiones regularizadas añadiendo a los valores obtenidos los correspondientes términos de regularización. Las funciones son transcripciones directas de la fórmula del gradiente y el coste para la regresión logística.

[36]:
```
def g(Z):
    return 1 / (1 + np.exp(-Z))
```

[37]:
```
def coste_reg(Theta, X, Y, Lambda):
    m = np.shape(X)[0]
    cost = -(((np.log(g(X@Theta))).T@Y) + ((np.log(1-g(X@Theta))).T@(1 - Y)))/m
    reg = (Lambda/(2*m))*sum(Theta**2)
    return cost + reg
```

[38]:
```
def gradiente_reg(Theta, X, Y, Lambda):
    m = np.shape(X)[0]
    grad = (X.T@(g(X@Theta)-Y))/m
    aux = np.copy(Theta)
    aux[0] = 0.0
    reg =  (Lambda/m)*aux
    return grad + reg
```

1.1.3 Clasificación de uno frente a todos

Para obtener la matriz de thetas óptimas primero tenemos que aplicar una conversión al vector *y*, transformándolo en una matriz con tantas filas como etiquetas tengamos para clasificar. La fila 1 tiene todos los valores a 0 salvo aquellos que sean 1; la fila 2 tiene todos los valores a 0, salvo aquellos que fueran 2; y así sucesivamente.

Significativamente, la fila 0 tiene todos los valores a 0 salvo aquellos que sean 10. Es por eso que la optimización por minimización de esta primera fila se realiza fuera del bucle principal de la función oneVsAll():

[39]:
```
def oneVsAll(X, y, num_etiquetas, reg):
    Theta = np.zeros(X.shape[1])
    Thetas = np.zeros([num_etiquetas, X.shape[1]])

    y_i = np.where(y == 10, 1, 0)
    Thetas[0] = opt.fmin_tnc(func=coste_reg, x0=Theta, fprime=gradiente_reg,␣
 ↪args=(X, y_i, reg))[0]
```

```
    for i in range(1, num_etiquetas):
        y_i = np.where(y == i, 1, 0)
        Thetas[i] = opt.fmin_tnc(func=coste_reg, x0=Theta,␣
↪fprime=gradiente_reg, args=(X, y_i, reg))[0]

    return Thetas
```

La forma en que se clasifican los elementos es, a partir de una muestra y las thetas óptimas, aplicando la función sigmoidea a su producto. La etiqueta que consiga un valor más elevado (y, por tanto, más cercano a 1) es la seleccionada:

[53]:
```
def clasificador(muestra, num_etiquetas, Thetas):

    sigmoides = np.zeros(num_etiquetas)

    for i in range(num_etiquetas):
        sigmoides[i] = g(np.dot(muestra, Thetas[i, :]))

    return np.argmax(sigmoides)
```

Una vez clasificada toda la muestra, sólo queda comparar con los resultados reales para comprobar el porcentaje de precisión de nuestra regresión logística. En este test hemos obtenido una precisión en los resultados de la clasificación del **96.48%**, superior al 95% esperado:

[61]:
```
def porcentajeCoincidencias(a, b):
    comp = a == b
    return 100 * sum(map(lambda comp : comp == True, comp)) / comp.shape
```

[62]:
```
def entrenamiento(X, y, num_etiquetas, reg):
    Thetas = oneVsAll(X, y, num_etiquetas, reg)

    y_ = np.zeros(X.shape[0])
    y = np.where(y == 10, 0, y)

    for i in range(X.shape[0]):
        y_[i] = clasificador(X[i, :], num_etiquetas, Thetas)

    return porcentajeCoincidencias(y_, y)
```

[63]:
```
def second_test():
    Lambda = 0.1
    num_etiquetas = 10

    X, y = load_file('ex3data1.mat')
    X_ones = np.hstack([np.ones([np.shape(X)[0], 1]), X])
    Y_ravel = np.ravel(y)

    percentage = entrenamiento(X_ones, Y_ravel, num_etiquetas, Lambda)
    print("Porcentaje reg logística: ", percentage)
```

[64]:
```
second_test()
```

```
Porcentaje:  [96.48]
```

1.2 Redes neuronales

El objetivo de esta parte de la práctica es utilizar los pesos proporcionados para una red neuronal ya entrenada sobre los ejemplos para evaluar su precisión sobre esos mismos ejemplos.

Utilizamos una función auxiliar para cargar los datos de *ex3data1.mat* y los pesos de la red neuronal entrenada de *ex3weights.mat*, para pasar a comprobar los resultados del entrenamiento.

[71]:
```
def load_neural_data(filename):
    weights = loadmat(filename)
    return weights['Theta1'], weights['Theta2']
```

La función de propagación hacia delante toma una muestra X y dos thetas, una para el paso de la capa de entrada a la capa oculta, y la otra para el paso de la capa oculta a la capa de salida; aplicando regresión logística podemos obtener un valor h(x(i)) para cada ejemplo i.

Tal y como hicimos en el apartado anterior, clasificaremos la muestra atendiendo al sigmoide cuyo valor sea más elevado, tal y como se ve en la función predictor():

[86]:
```
def propagacion(X, theta1, theta2):
    a1 = np.hstack([np.ones([X.shape[0], 1]), X])
    z2 = np.dot(a1, theta1.T)
    a2 = np.hstack([np.ones([X.shape[0], 1]), g(z2)])
    z3 = np.dot(a2, theta2.T)
    return g(z3)
```

[87]:
```
def predictor(sigmoides) :
    y = np.zeros(sigmoides.shape[0])
    for i in range(sigmoides.shape[0]):
        y[i] = np.argmax(sigmoides[i, :]) + 1
    return y
```

[88]:
```
def entrenamiento_neural(X, y, theta1, theta2) :
    sigmoides = propagacion(X, theta1, theta2)
    y_ = predictor(sigmoides)
    return porcentajeCoincidencias(y_, y)
```

Tras comparar los resultados de la clasificación mediante propagación hacia delante con los datos reales, obtenemos que la precisión de la red neuronal es del **97.52%** (se esperaba un valor de entorno al 97.5%):

[91]:
```
def third_test():
    X, y = load_file('ex3data1.mat')
    Y_ravel = np.ravel(y)

    theta1, theta2 = load_neural_data('ex3weights.mat')
    percentage = entrenamiento_neural(X, Y_ravel, theta1, theta2)
    print("Porcentaje red neuronal: ", percentage)
```

[92]:
```
third_test()
```

```
Porcentaje red neuronal:  [97.52]
```