Práctica 3: Regresión logística multi-clase y redes neuronales

Rafael Herrera Troca
Rubén Ruperto Díaz

Aprendizaje Automático y Big Data
Doble Grado en Ingeniería Informática y Matemáticas

4 de septiembre de 2020

1. Introducción

En esta práctica hemos comprobado la eficacia de dos clasificadores, uno por regresión logística regularizada y otro mediante una red neuronal, usando como datos un conjunto de imágenes de números escritos a mano. En este caso, a diferencia de lo que sucedía en la práctica anterior, hay más de dos categorías entre las que clasificar, concretamente 10, por lo que se entrenan 10 clasificadores que indican con qué probabilidad pertenece a cada categoría. La respuesta final se define como aquella con mayor probabilidad.

Hemos programado el código en el entorno *Spyder* con Python 3.7.

2. Procedimiento

2.1. Parte 1: Regresión logística regularizada multiclase

Lo primero que hay que hacer en esta parte de la práctica es definir la función de coste y su gradiente. Las definimos en su forma vectorizada, que hace los cálculos mucho más rápido, y exactamente de la misma forma que en la práctica anterior. Así, la función de coste sigue la fórmula:

$$J(\theta) = -\frac{1}{N}((log(g(X\theta)))^T y + (log(1 - g(X\theta)))^T(1 - y)) + \frac{\lambda}{2N}\sum_{j=1}^{n}\theta_j^2$$

Para su gradiente usamos la fórmula siguiente, calculando todas las componentes a la vez en un vector:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{N}X^T(g(X\theta) - y) + \frac{\lambda}{N}\theta_j$$

Además de la función de coste y su gradiente, definimos también la función *oneVsAll*, que entrena un clasificador por regresión logística para cada etiqueta y devuelve la matriz Θ resultante. Dicha matriz tiene, en la fila j, el valor de θ correspondiente al clasificador para la etiqueta j. Para construir la matriz, *oneVsAll* hace un bucle y utiliza la función *fmin_tnc* de la librería *scipy.optimize* para obtener los valores de θ que minimizan la función de coste.

Una vez hemos definido todas estas funciones, hacemos un bucle para probar a entrenarlo con distintos valores del parámetro de regularización, con una escala logarítmica de base 10, tomándolos entre 10^{-10} y 10^2. Después, calculamos y representamos en una gráfica el porcentaje de aciertos que tiene el modelo utilizando los casos de entrenamiento para cada valor del parámetro de regularización. Para cada caso, consideramos que la respuesta dada por el modelo es la del clasificador que responde con mayor probabilidad. Los detalles sobre el porcentaje de acierto obtenido se pueden encontrar en la sección de resultados.

2.2. Parte 2: Red neuronal

En esta parte, hemos aplicado una red neuronal de tres capas ya entrenada a los casos de prueba para comprobar su porcentaje de acierto. La primera capa tiene 400 neuronas, la capa oculta 25 y la capa de salida, 10. Para ello, hemos definido la función *applyLayer*, que dadas las matrices de datos y de pesos, pasa los datos por la capa correspondiente de la red neuronal multiplicando ambas matrices y aplicando la función sigmoide al resultado.

Una vez tenemos el resultado de la red neuronal, calculamos el porcentaje de acierto, entendiendo que la respuesta dada por la red es la correspondiente a la neurona que devuelve mayor probabilidad. Los detalles sobre el porcentaje de acierto obtenido y los resultados se pueden encontrar en la siguiente sección.

3. Resultados

3.1. Parte 1: Regresión logística regularizada multiclase

Una vez entrenado el modelo de regresión logística multiclase con termino de regularización $\lambda = 0{,}1$, pasamos a evaluar su tasa de acierto con los datos de entrenamiento, obteniendo un porcentaje de 96,48 %.

Adicionalmente, hemos probado a entrenar la regresión logística con distintos valores de $\lambda \in \{10^i \mid i \in [-10, 2], i \in \mathbb{Z}\}$. Observamos que para valores de λ mayores que 0,1, la tasa de aciertos va disminuyendo progresivamente, mientras que para valores menores se llega a una tasa que ronda el 97 %, aunque probablemente a costa de sobreaprendizaje en el modelo.

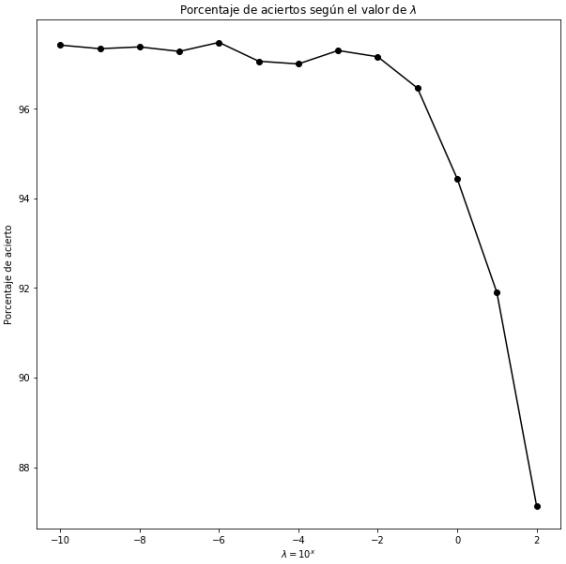

Figura 1: Comparativa de éxito al clasificar los datos de entrenamiento según λ

3.2. Parte 2: Red neuronal

En cuanto a la red neuronal, calculamos la salida a partir de las matrices de pesos ya entrenados y evaluamos su precisión con los datos de entrenamiento, que en este caso es ligeramente mayor que la regresión logística, llegando a un porcentaje de éxito del 97,52 %.

Por curiosidad, hemos mirado las imágenes de algunos de los números fallados por la red neuronal. En muchas ocasiones hemos notado que los fallos se debían a que los números presentaban trazos inusuales o formas que se podían confundir con otros números. Un ejemplo es el de la figura 2, en el que la respuesta correcta era 1, pero la red neuronal respondió 2.

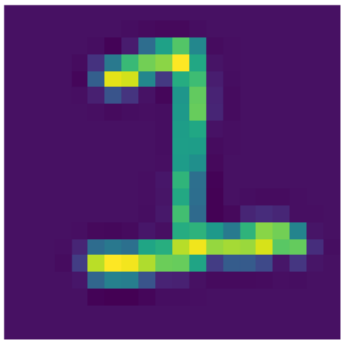

Figura 2: Caso mal interpretado por la red neuronal

4. Conclusiones

Hemos estudiado y comparado el comportamiento de la regresión logística multiclase y una red neuronal al reconocer imágenes de números manuscritos. Ambos modelos han obtenido un porcentaje de aciertos muy alto al clasificar los datos de entrenamiento.

Sin embargo, debido al gran número de variables, concretamente 400 variables que se corresponden con cada píxel de las imágenes, es imposible representar gráficamente qué criterios ha usado la regresión logística para clasificar una imagen como un número u otro. Tampoco es posible hacerlo con la red neuronal, por su propia naturaleza poco explicable. Además, no podemos saber si los modelos han sufrido sobreaprendizaje, ya que no disponemos de más datos de prueba que los usados para el entrenamiento.

5. Anexo: Código

A continuación se ofrece el código escrito para resolver los ejercicios propuestos en la práctica.

```
'''
Práctica 3

Rubén Ruperto Díaz y Rafael Herrera Troca
'''

import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from scipy.optimize import fmin_tnc

os.chdir("./resources")


# Función sigmoide
def sigmoide(z):
    return 1 / (1 + np.exp(-z))

# Función de coste regularizada
def coste(theta, X, Y, lmb=1):
    gXTheta = sigmoide(np.dot(X, theta))
    factor = np.dot(np.log(gXTheta).T, Y) + np.dot(np.log(1 - gXTheta).T, 1-Y)
    return -1 / len(Y) * factor + lmb / (2 * len(Y)) * np.sum(theta**2)

# Gradiente de la función de coste regularizada
def gradiente(theta, X, Y, lmb=1):
    gXTheta = sigmoide(np.dot(X, theta))
    thetaJ = np.concatenate(([0], theta[1:]))
    return 1 / len(Y) * np.dot(X.T, gXTheta-Y) + lmb / len(Y) * thetaJ

# Entrena varios clasificadores por regresión logística con el
# término de regularización dado en 'reg' y devuelve el resultado
# en una matriz con el clasificador de la etiqueta i-ésima en dicha
# fila
def oneVsAll(X, y, num_et, reg):
    theta0 = np.zeros(np.shape(X)[1])

    Y = ((1 == y[:]) * 1)
    result = np.array([fmin_tnc(func=coste, x0=theta0, fprime=gradiente, args=(X, Y, reg))[0]])
    for i in range(2, num_et+1):
        Y = ((i == y[:]) * 1)
        result = np.vstack((result, np.array([fmin_tnc(func=coste, x0=theta0, fprime=gradiente, args=(X, Y, reg))[0]])))
```

```
44
45      return result
46
47  # Dada la entrada 'X' y los pesos 'theta' de una capa de una red
48  # neuronal, aplica los pesos y devuelve la salida de la capa
49  def applyLayer(X, theta):
50      thetaX = np.dot(X, theta.T)
51      return sigmoide(thetaX)
52
53  # Calcula el porcentaje de acierto obtenido con la entrada dada
54  # y las etiquetas correctas en y
55  def acierto(X, Y):
56      resultados = X.argmax(axis=1) + 1
57      return np.count_nonzero(resultados == Y.ravel()) / len(Y)
58
59
60  ## Parte 1
61
62  # Leemos los datos
63  data = loadmat('ex3data1.mat')
64  y = data['y'].ravel()
65  X = data['X']
66
67  # Entrenamos el clasificador con distintos valores para el término
68  # de regularización y almacenamos el porcentaje de acierto para
69  # cada uno
70  X2 = np.hstack((np.array([np.ones(len(y))]).T,X))
71  porc_ac = []
72  for lmb in range(-10, 3):
73      clas = oneVsAll(X2, y, 10, 10**lmb)
74      result = sigmoide(np.dot(X2, clas.T))
75      porc_ac.append(acierto(result, y)*100)
76      print("Con lambda = 10^" + str(lmb) + ", el porcentaje de
        aciertos del clasificador es un " + str(porc_ac[-1]) + "%.")
77
78  # Representamos el porcentaje de aciertos del clasificador en
79  # función del término de regularización
80  plt.figure(figsize=(10,10))
81  plt.plot(range(-10,3), porc_ac, 'ko-')
82  plt.title(r"Porcentaje de aciertos según el valor de $\lambda$")
83  plt.xlabel(r"$\lambda = 10^{x}$")
84  plt.ylabel("Porcentaje de acierto")
85  plt.savefig("AcXLambda.png")
86  plt.show()
87
88
89  ## Parte 2
90
91  # Leemos los datos
92  weights = loadmat('ex3weights.mat')
93  theta1, theta2 = weights['Theta1'], weights['Theta2']
94
95  # Aplicamos las dos capas de la red neuronal
96  lay1 = applyLayer(X2, theta1)
97  lay1 = np.hstack((np.array([np.ones(np.shape(lay1)[0])]).T,lay1))
```

```
98 lay2 = applyLayer(lay1, theta2)
99
100 # Comprobamos el porcentaje de aciertos
101 acR = acierto(lay2, y)
102 print("El porcentaje de aciertos de la red neuronal es un " + str(
       acR*100) + "%.")
```