Práctica 3

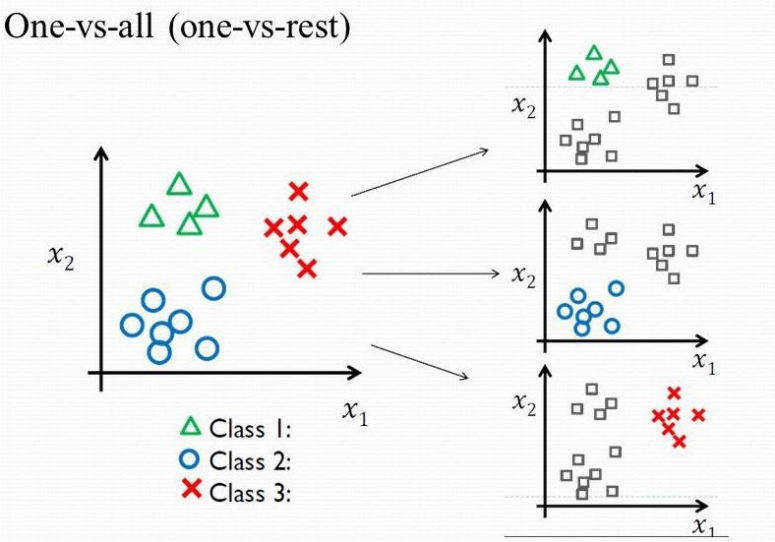
Alejandro Aizel Boto
Miguel Robledo Blanco

Índice

1. Descripción General
2. Explicación de la Solución
3. Resultados
4. Conclusión
5. Código

1. Descripción General

Esta práctica, al igual que la anterior consta de dos parte. La primera consiste en aplicar el método de regresión logística multi-clase para poder distinguir un número de una imagen. Dado que tenemos que entrenar 10 modelos, el proceso a seguir es ir entrenando uno a uno cada uno de los modelos tratando todos como iguales. Este método se denomina one-vs-all.



Para ello contamos con un archivo llamado "ex3data1.mat" que nos permite cargar imágenes como arrays de numpy. A continuación vemos un ejemplo de 10 imágenes aleatorias de este archivo.

El segundo método consiste en utilizar una red neuronal sencilla (y ya entrenada) con una sola capa oculta, que realiza la misma función que la primera, y utilizar la propagación hacia adelante con los ejemplos de entrenamiento para ver el porcentaje de acierto que teníamos.

2. Explicación de la Solución

En este caso la práctica como hemos comentado anteriormente se divide en dos apartados, la regresión logísitica multi-clase y el uso de la red neuronal. En esta práctica se usan varias funciones de la practica anterior ya descritas así que vamos a obviar su explicación. En caso de necesitarse, consultar la práctica 2.

El código comienza con una llamada al método `main()` que contiene dos funciones, una por cada parte de la práctica. Primero preparamos los datos leyendo desde archivo el fichero "ex3data2.mat". Cada ejemplo de entrenamiento es una foto(matriz) de 20×20 la cual ha sido desplegada para formar filas de 400 elementos cada uno de los cuales representa la intensidad en escala de grises de ese punto. De esta forma, X es una matriz de 5000×400 donde cada fila representa la imagen de un número escrito a mano. Lo que hacemos es dividir la X y la y, y añadir unos a las X.

```
def main():
    data = loadmat('Práctica 3/Recursos/ex3data1.mat')

    y = np.squeeze(data['y'])
    X = data['X']

    X = np.hstack([np.ones([len(X), 1]), X])

    multi_class(X, y)
    neural_network(X, y)
```

Vamos a comenzar explicando la primera parte. Para resolver la primera parte nos hemos vuelto a apoyar en el método del descenso del gradiente con su función de coste y la función sigmoide. La primera función tiene esta forma:

```
def multi_class(X, y):
    y[y==10] = 0

    Thetas = oneVsAll(X, y, 10, 0.5)

    evaluation(X, y, Thetas)
```

Como en el vector el cero viene representado por el 10, cambiamos todos estos valores por un 0 lo que nos facilita mucho la programación de la solución. Llamamos a la función onevsAll con estas variables por parámetro y luego evaluamos en resultado obtenido.

Esta función itera 10 veces sobre el siguiente algoritmo. Usando el array de etiquetas de 5000 posiciones, se marcan a 1 aquellas posiciones que sean iguales a la variable i la cual nos dice en qué clasificador nos encontramos (0-9). De esta forma en cada iteración tenemos un vector y que nos marca el clasificador objetivo. Construimos nuestro vector theta y usamos la función opt.fmin_tnc para minimizar los valores de theta.

```
def oneVsAll (X, y, num_labels, reg):
    Thetas = np.zeros((num_labels, len(X[0])))

    for i in range(num_labels):
        Thetas[i] = opt.fmin_tnc(func=cost_reg, x0=Thetas[i],
fprime=gradient_reg, args=(X, [i] == y, reg))[0]

    return Thetas
```

En esta función se aplica el sigmoide a la matriz de theta calculada en la función anterior multiplicada por la matriz X. Obtenemos los valores máximos de los 5000 ejemplos de entrenamiento los cuales marcaran qué numero (0-9) se le asigna y comprobamos cuántos fueron correctamente clasificados:

```
def evaluation (X, y, Thetas):
    h = sigmoid(np.dot(Thetas, np.transpose(X)))

    maxValues = np.argmax(h, axis=0)

    result = maxValues == y

    print("Se han clasificado correctamente: {} %".format(np.sum(result)
         / len(result) * 100))
```

Pasamos a explicar la segunda parte. Esta última función calcula el resultado de la red neuronal. Cargamos las dos matrices de pesos y vamos calculando el resultado de cada capa aplicando la función sigmoide. Una vez tenemos el resultado de la capa exterior nos quedamos con los indices de los valores máximos para saber de qué número se trata. A este le sumamos 1 para que coincida con el vector de y. Pol último cambiamos los valores correctos por true y hacemos la suma para sacar el porcentaje.

```
def neural_network(X, y):
    weights = loadmat('Práctica 3/Recursos/ex3weights.mat')

    theta1, theta2 = weights['Theta1'], weights['Theta2']

    hidden_layer = sigmoid(np.matmul(theta1, np.transpose(X)))
    hidden_layer = np.vstack([np.ones([len(X)]), hidden_layer])

    output_layer = sigmoid(np.matmul(theta2, hidden_layer))

    max_values = np.argmax(output_layer, axis=0) + 1

    correct_values = y == max_values

    print("Se han clasificado correctamente: {} %".format(np.sum(
         correct_values) / len(correct_values) * 100))
```

3. Resultados

En este apartado vamos a ir comentando los distintos resultados que hemos obtenido.

Vamos a comenzar con el primer apartado, correspondiente a la compañía de distribución de comida. Como esperamos, el resultado fue:

```
> print("Se han clasificado correctamente: {} %".format(np.sum(result) /
      len(result) * 100))
```

```
  Se han clasificado correctamente:  96.4 %
```

Para la red neuronal obtuvimos tambien el resultado esperado:

```
> print("Se han clasificado correctamente: {}
      %".format(np.sum(correct_values) / len(correct_values) * 100))
```

```
  Se han clasificado correctamente:  97.5 %
```

4. Conclusión

Esta práctica nos ha servido para entender el funcionamiento de la regresión logística multi-clase y para ver como funciona la propagación hacia adelante de las redes neuronales utilizando ejemplos que le pasamos a la misma para ver que solución da.

En el siguiente apartado se muestra el código completo por si se quiere ver seguido o hacer alguna prueba compilada.

5. Código

```
from scipy . io import loadmat
from sklearn.preprocessing import PolynomialFeatures
from pandas.io.parsers import read_csv
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt

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


def gradient(Theta, XX, Y):
    H = sigmoid(np.matmul(XX, Theta))

    return (1 / len(Y)) * np.matmul(XX.T, H - Y)

def gradient_reg(Theta, XX, Y, Landa):
    G = gradient(Theta, XX, Y)

    G[1:] += Landa / len(XX) * Theta[1:]

    return G

def cost(Theta, X, Y):
    H = sigmoid(np.matmul(X, Theta))

    return (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y),
np.log(1 - H)))

def cost_reg(Theta, X, Y, Landa):
    C = cost(Theta, X, Y)

    return C + Landa / (2 * len(X)) * np.sum(Theta ** 2)

def oneVsAll (X, y, num_labels, reg):
    Thetas = np.zeros((num_labels, len(X[0])))

    for i in range(num_labels):
        Thetas[i] = opt.fmin_tnc(func=cost_reg, x0=Thetas[i],
fprime=gradient_reg, args=(X, [i] == y, reg))[0]

    return Thetas

def evaluation (X, y, Thetas):
    h = sigmoid(np.dot(Thetas, np.transpose(X)))

    maxValues = np.argmax(h, axis=0)

    result = maxValues == y

    print("Se han clasificado correctamente: {} %".format(np.sum(result) /
len(result) * 100))

def multi_class(X, y):
    y[y==10] = 0

    Thetas = oneVsAll(X, y, 10, 0.5)
```

```
    evaluation(X, y, Thetas)


def neural_network(X, y):
    weights = loadmat('Práctica 3/Recursos/ex3weights.mat')

    theta1, theta2 = weights['Theta1'], weights['Theta2']

    hidden_layer = sigmoid(np.matmul(theta1, np.transpose(X)))
    hidden_layer = np.vstack([np.ones([len(X)]), hidden_layer])

    output_layer = sigmoid(np.matmul(theta2, hidden_layer))

    max_values = np.argmax(output_layer, axis=0) + 1

    correct_values = y == max_values

    print("Se han clasificado correctamente: {}
%".format(np.sum(correct_values) / len(correct_values) * 100))

def main():
    data = loadmat('Práctica 3/Recursos/ex3data1.mat')

    y = np.squeeze(data['y'])
    X = data['X']

    X = np.hstack([np.ones([len(X), 1]), X])

    multi_class(X, y)
    neural_network(X, y)


main()
```