Universidad Complutense de Madrid
Facultad de Informática
Aprendizaje Automático y Big Data

Memoria Práctica 3.

Profesor:
- Alberto Díaz Esteban.

Alumnos:
- Marina de la Cruz López.
- Diego Alejandro Rodríguez Pereira.

Introducción

En el desarrollo de esta práctica se ha implementado dos partes. En la primera parte se ha implementado un algoritmo de regresión logística multi-clase para el reconocimiento de imágenes de números escritos a mano. En la segunda parte se usa una red neuronal para realizar la clasificación, con los pesos ya calculados para red.

Parte1 : Regresión Logística Multi-Clase

Para esta parte hemos comenzado con la visualización de datos, por lo que hemos tenido que leer el conjunto de datos desde el fichero *ex3 data1.mat,* el cual es un fichero en formato nativo Octave/Matlab. Para la carga del fichero hemos utilizado la función de la librería *spicy*: *spicy.io.loadmat*. Esta función devuelve un diccionario del que se van a extraer las variables de las matrices '**X'** e '**y'**.

Cada ejemplo del conjunto de datos leídos pertenece a una imagen de 20x20 pixeles, en el que cada píxel está representado por un número entero entre 0 y 255 que representa la intensidad en escala de grises. Luego, cada matriz de 20x20 aparece desplegada en un vector de 400 componentes la cual ocupa una fila de la matriz X. De esta manera, al leerlo, la matriz X es una matriz de 5000x400 donde cada fila de la matriz representa la imagen de un número y la cantidad de filas la cantidad de números. A su vez el *dataset* contiene un vector de etiquetas '**y'** de 5000 componentes que representan el número de la imagen, donde del "1" al "9" representan los números enteros del 1 al 9 y el "10" representa el número 0.

Clasificación de uno frente a todos (One vs All)

Para realizar el One vs All movemos las etiquetas un número menos para poder hacer la transformación one_hot con los números del 0 al 9, posteriormente transformamos el array de **'y'** en una matrix one_hot donde cada etiqueta del 0 al 9 se convierte en un array de 9 posiciones relleno de ceros menos por la posición que corresponde a la etiqueta que tiene un uno.

En la matriz **'X'** tenemos que añadir, como siempre, la columna de unos para vectorizar los cálculos de las *Thetas*.

Usando la matrix **'y_onehot'** y la **'X'** con las imágenes se entrenan 9 clasificadores binarios por regresión logística para cada una de las clases, en este caso 10 clases (números del 0 al 9) para predecir la probabilidad de que sea la clase esperada. En nueva entrada, para hacer una predicción, se escoge la clase que maximice, como se muestra en la Figura1.

$$\max_i h_\theta^{(i)}(x)$$

Figura 1.

Para el entrenamiento se ha utilizado la función *scipy.optimize.fmin_tnc*. Función devuelve un array de *theta,* para cada clase meternos las *thetas* resultantes en una matriz, donde cada fila de *theta* corresponde a los parámetros aprendidos para el clasificador de una de las clases.

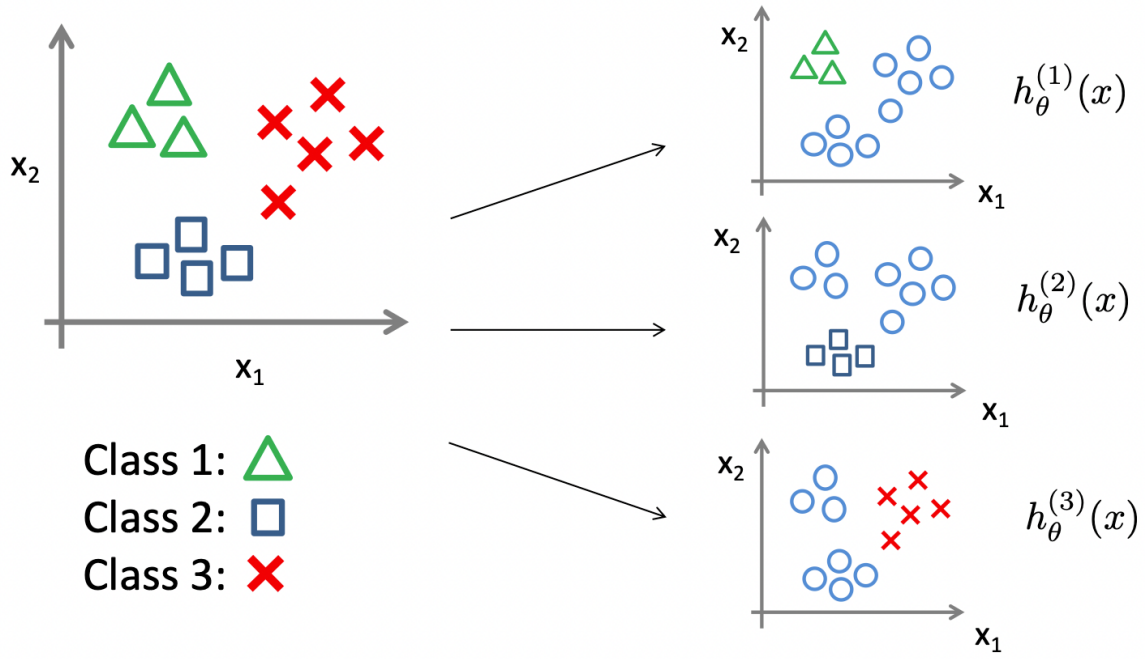

Figura 2

Finalmente usamos los datos de entrenamiento para que nos devuelva la cantidad de aciertos del clasificador frente al total de ejemplos de entrenamiento. Como tenemos 10 arrays de *Thetas* calculamos el valor de la sigmoide para los 10 y nos quedamos con la etiqueta del clasificador que dé un valor más grande para la sigmoide. Las etiquetas resultantes las comparamos con las reales para determinar si la clasificación se ha realizado correctamente.

Parte 2: Redes neuronales

Para esta parte se ha utilizado una red neuronal para resolver el mismo problema que en el apartado anterior. El objetivo es utilizar los pesos calculados por la Red Neuronal en el entrenamiento (que se nos proporciona) con los datos para evaluar su precisión sobre esos mismos.

Leemos el archivo *ex3 data1.mat* y nuevamente separamos la matriz **'X'** con las imágenes y el array de etiquetas **'y'** de la 1 a la 10**,** en este caso no será necesario realizar el one_hot de **'y'.** Añadimos la columna de unos a la matriz **'X'.**

La Red Neuronal está formada por 3 capas, una capa de entrada, una capa oculta y una capa de salida. La primera capa tiene 400 neuronas , la segunda capa (oculta) tiene 25 y la última capa tiene 10 neuronas (tantas como clases) .

Como los pesos de la Red Neuronal ya están calculados, vienen proporcionados por la práctica, por lo que no hace falta que entrenemos el algoritmo nuevamente para obtenerlos. Simplemente tenemos que leerlos y utilizarlos en el forward propagation.

El fichero *ex3weights.mat* contiene los pesos ya calculados para cada capa, es decir, contiene las matrices *theta1* y *theta2* que son el resultado de entrenar la Red Neuronal. El fichero lo cargamos usando la función *scpicy.io.loadmat*.

Una vez cargados los pesos hemos implementado la función que aplica la propagación hacia adelante para calcular el valor de la hipótesis de cada ejemplo, como se muestra en la Figura3.

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Figura 3

Una vez realizada la propagación hacia adelante, calculamos la precisión del algoritmo. Para esto nos quedamos con el índice de la neurona que nos devuelve un valor más alto con *np.argmax*, este corresponde a la clase que mejor se adecua a la entrada, comparamos el índice con el vector **'y-1'** para que coincidan los índices de las neuronas con las clases de la 0 a la 9, si coincide entonces el elemento ha sido bien clasificado, si no mal. Calculamos el porcentaje de aciertos sobre todos los ejemplos de entrenamiento.

Resultados Obtenidos

Primer apartado

Para este apartado, se han realizado las predicciones y se ha obtenido que el algoritmo ha tenido un porcentaje de aciertos del 96,46% para lambda = 0.1 de regularización. Este resultado es el que esperábamos.

Segundo apartado

La Red Neuronal, cuyos pesos estaban ya calculados, ha arrojado como resultado una precisión de 97,52%. Este resultado es el que esperábamos.

Código

Primer Apartado

```
#Imports
import numpy as np
```

```
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from matplotlib.axes import Axes
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import math
import scipy.optimize as opt
from sklearn.preprocessing import PolynomialFeatures
from scipy.io import loadmat
```

```
def sigmoid_funct(z):
    return 1.0 / (1.0 + np.exp(-z))
```

```
def func_coste_reg(Thetas, X, Y, lmb):
    m = np.shape(X)[0]
    return cost_funct(Thetas, X, Y) + regularizacion(Thetas[1:], lmb, m)
```

```
def cost_funct(Theta, X, Y):
    g = sigmoid_funct(np.matmul(X, Theta))
    m = np.shape(X)[0]
    J = (np.matmul(np.transpose(np.log(g)), Y)) + (np.matmul(np.transpose(np.log(1-
g)), (1 - Y)))
    return -J/m
```

```
def regularizacion(Thetas, lmb, m):
    return (lmb/(2*m))*np.sum(Thetas**2)
```

```
def func_grad_reg(Thetas, X, Y, lmb):
    m = np.shape(X)[0]
    return np.add(gradient(Thetas, X, Y) , reg_grad(Thetas[1:], lmb, m))
```

```
def gradient(Theta, X, Y):
    m = np.shape(X)[0]
    g = sigmoid_funct(np.matmul(X,Theta))
    J = np.dot(np.transpose(X), (g - Y))
    return J/m
```

```
def reg_grad(Thetas, lmb, m):
        return np.insert((lmb/m)*Thetas, 0, values=[0])
```

```
def oneVsAll(X, y, n_labels,cost, grad, reg):
    #Initialize Thetas
    all_theta = np.zeros((n_labels, X.shape[1]))

    #Minimize for all labels
    for i in range(n_labels):
        Thetas = np.zeros((X.shape[1],1))
```

```
        result = opt.fmin_tnc(func=cost, x0=Thetas, fprime=grad, args=(X,
y[:,i],reg))
        all_theta[i] = result[0]

    return all_theta
```

```
def porcentaje_aciertos(Theta, X, Y):
    # Calculamos los valores estimados segun la theta que hemos obtenido

    h = sigmoid_funct(np.matmul(Theta, X.T))

    #Nos quedamos con el máximo
    indexes = np.argmax(h , axis=0)

    m = np.shape(X)[0]
    # Se compara la estimacion nuestra con el resultado real (Y).
    acc = (np.sum(indexes == Y)/m)*100

    # Devolvemos el porcentaje
    return acc
```

```
data = loadmat('ex3data1.mat')
# se pueden consultar las claves con data.keys()
y = data['y']
X = data['X']
# almacena los datos leídos en X, y
y = np.ravel(y)

m = np.shape(X)[0]
print(m)
# Agregamos una columna de 1s
X_ones = np.hstack([np.ones([m, 1]), X])

num_labels = 10

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

#Movemos los labels para que se encuentren entre 0 y 9
y = (y - 1)

y_onehot = np.zeros((m, num_labels))  # 5000 x 10
for i in range(m):
    y_onehot[i][y[i]] = 1


all_thetas = oneVsAll(X_ones,y_onehot,num_labels,func_coste_reg,func_grad_reg,
0.1)
```

```
porcentaje_aciertos(all_thetas, X_ones, y)
```

Segundo Apartado

```
#Imports
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from matplotlib.axes import Axes
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import math
import scipy.optimize as opt
from sklearn.preprocessing import PolynomialFeatures
from scipy.io import loadmat


def sigmoid_funct(z):
    return 1.0 / (1.0 + np.exp(-z))
```

```
def forward_propagation(X, theta1, theta2, m):

    #Capa entrada asignamos la X con los unos incluidos
    a1 = X

    #capa intermedia (hidden) calculamos las ecuaciones de la anterior, aplicamos
la sigmoide e incluimos los unos de la neurona 0
    z2 = np.matmul(a1, theta1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoid_funct(z2)])

    #Capa salida calcuamos las ecuaciones con theta2 y aplicamos la sigmoide, nos
devuelve la matriz de salida 5000x10
    z3 = np.dot(a2, theta2.T)
    a3 = sigmoid_funct(z3)

    return a3
```

```
def main():

    data = loadmat ("ex3data1.mat")

    #almacenamos los datos leídos en X e y
    X = data['X']
    m = X.shape[0]
    X = np.hstack([np.ones([m, 1]), X])
    y = data['y']
    y = np.ravel(y)
```

```
    #Cargamos los weights y los asignamos a theta1 y theta2
    weights = loadmat("ex3weights.mat")
    theta1, theta2 = weights["Theta1"], weights["Theta2"]

    #hacemos el forward propagation
    h = forward_propagation(X, theta1, theta2, m)

    #Sacamos la cantidad de elementos bien clasificados
    indexes = np.argmax(h, axis=1)

    #Hacemos el porcentaje de aciertos y lo devolvemos
    acc = (np.sum(indexes == (y-1))/m)*100

    print(acc)
    return acc
```

```
main()
```