Aprendizaje Automático y Big Data, Práctica 3

Romain Contrain y Guillermo Martín Sánchez

Marzo 2020

1 Objetivo

El objetivo de esta práctica es implementar el algoritmo de regresión logística para la clasificación multiclase. Una vez más, buscamos el vector θ tal que la función $h_\theta = \frac{1}{1+e^{-\theta^T x}}$ minimice la función coste regularizada:

$$J(\theta) = [\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(h_\theta(1-x^{(i)}))]] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \quad (1)$$

En este caso, vamos a intentar discernir unos dígitos escritos a mano y clasificarlos en los posibles valores: $0, 1, ..., 9$. Estos dígitos están escritos en una imagen de 20×20 píxeles, por lo que $x^{(i)}$ es un vector con 400 elementos (la linearización de la imagen) que representan la intensidad en cada píxel.

Finalmente, estudiamos como funciona la propagación de información en una red neuronal simple. Cargamos los pesos ya optimizados y calculamos la precisión del modelo con estos valores.

2 Método

De nuevo, hemos usado el método *fmin_tnc* de la librería *scipy.optimize* para calcular el mínimo de la función de coste regularizada (Eq 1).

Hemos implementado una función que calculase los θ óptimos para separar cada uno de los dígitos de los demás. Es decir, implementar un modelo que discierna lo que es un 0 del resto, otro que discierna lo que es un 1 del resto,... y así hasta 10 modelos o, equivalentemente, 10 vectores $\{\theta_i\}_{i=0}^{9}$. Para determinar luego a qué clase pertenece un ejemplo concreto $x^{(j)}$, calculamos $h_{\theta_i}(x^{(j)})$ para cada θ_i y clasificamos el ejemplo como el i que maximice esta cantidad.

En el segundo apartado, hemos implementado la propagación de la red neuronal que se muestra en la Figura 1 (una capa oculta, función de activación logística). Así, para unos pesos que se nos han dado como óptimos, hemos calculado la activación de las neuronas de la última capa para cada uno de los estímulos. De nuevo, clasificamos el ejemplo $x^{(j)}$ como un dígito i si, en la

última capa, la neurona más activa es la i-ésima cuando metemos como input a la red neuronal dicho ejemplo.

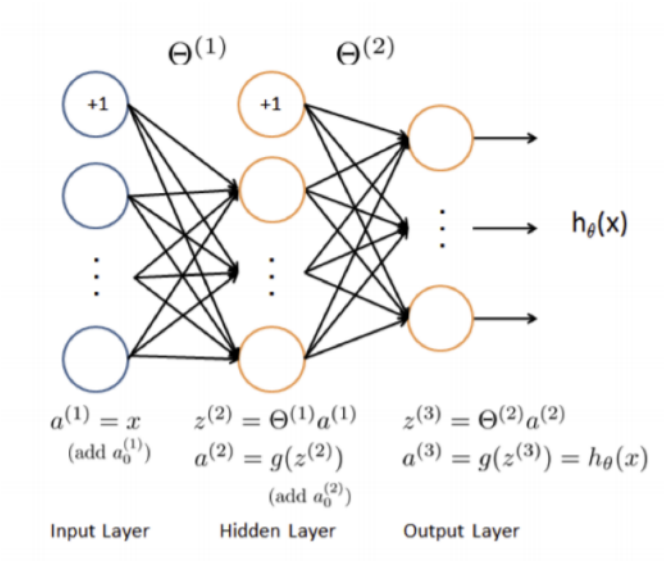

Figure 1: Red neuronal de la que vamos a programar su propagación

3 Resultados

Para la regresión logística, con parámetro de regularización $\lambda = 0.1$ obtenemos una precisión del 0.965 mientras que para la red neuronal obtenemos una precisión del 0.975. Una vez más, hace falta aclarar que esto se ha calculado sobre los ejemplos de entrenamiento y no sobre otros de test.

4 Conclusiones

Una vez más vemos lo general que es el método de la regresión logística. En el caso de la práctica anterior veíamos como podíamos usarlo para clasificación con una frontera que no fuera una recta. En esta, vemos como podemos usarlo en un caso de clasificación multiclase entrenando varios modelos con este mismo método. Por otro lado, nos hemos introducido al mundo de las redes neuronales, viendo como se produce la propagación.

5 Código

```
"""
Coursework 3: Multi-class logistical regression and neural networks
"""

#%%
"""
Imports and definitions
"""



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

def load_dat(file_name):
    """
    carga el fichero dat (Matlab) especificado y lo
    devuelve en un array de numpy
    """
    data = loadmat(file_name)
    y = data['y']
    X = data['X']
    return X,y

def sigmoid(z):
    """
    sigmoid function
    can be applied on numbers and on numpy arrays of any dimensions
    """
    return 1 / (1 + np.exp(-z))

def coste_reg(theta, X, Y, lamb):
    """
    cost function with regularization
    computes J(theta) for a given dataset
    with a regularization factor of lamb
    """
    m = np.shape(X)[0]
    H = sigmoid((np.dot(X,theta)))
    J = -1/m * ( np.log(H).transpose().dot(Y)
                 + np.log(1-H).transpose().dot(1-Y))
    reg = lamb/(2*m)*np.sum(theta[1:]**2)
    return J + reg
```

```
def gradiente_reg(theta, X, Y, lamb):
    """
    gradient function with regularization
    computes the gradient of J at a given theta for a given dataset
    with a regularization factor of lamb
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    H = sigmoid(np.dot(X,theta))
    G = 1/m * X.transpose().dot(H - Y)
    reg = np.ones(n,)
    reg[0] = 0
    reg = (lamb/m)*reg*theta
    return G + reg

def precision(theta,X,Y):
    """
    accuracy function
    computes the accuracy of the logistic model theta on X with true target variable Y
    """
    m = np.shape(X)[0]
    H = sigmoid(np.dot(X,theta.T))
    labels = np.argmax(H,axis = 1) + 1
    Y = Y.ravel()
    return np.sum(labels == Y)/m

def propagation(theta1,theta2,X):
    """
    forward propagation function
    performs the forward propagation in a neural network with the stucture
    decribed in the subject,
    with weights given by theta1 and theta2
    on the dataset X
    """
    m = np.shape(X)[0]
    a2 = sigmoid(np.dot(X,theta1.T))
    a2 = np.hstack([np.ones([m,1]),a2])
    H = sigmoid(np.dot(a2,theta2.T))
    return H

def precision_neur(theta1,theta2,X,Y):
    """
    accuracy function
    computes the accuracy of the neural network described by theta1 and theta2
    on dataset X with true target variable Y
```

```
    """
    m = np.shape(X)[0]
    H = propagation(theta1,theta2,X)
    labels = np.argmax(H,axis = 1) + 1
    Y = Y.ravel()
    return np.sum(labels == Y)/m

def oneVsAll(X,y,num_etiquetas,reg):
    """
    one-vs-all logistical regression
    trains num_etiquetas logistical classifiers to classify as many classes
    uses dataset X and true target variable y
    uses regularization with factor reg
    returns a matrix containing each models' parameters in its lines
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    X = np.hstack([np.ones([m,1]),X])
    Th = np.zeros((num_etiquetas, n+1))
    # For each class
    for i in range(num_etiquetas):
        # Mark the points as belonging to the class or not
        labels = np.zeros((m,1))
        labels[y == i+1] = 1  # the class at index 0 has value 1
        labels = labels.ravel()

        theta = np.zeros((n+1,))
        # Compute the optimal value for theta
        result = opt.fmin_tnc(func=coste_reg, x0=theta, fprime=gradiente_reg,
                                        args=(X,labels,reg), messages = 0)
        Th[i,:] = result[0]

    return Th



#%%
"""
1 - Multi-class logistical regression
"""


# loading the data
X,y = load_dat('ex3data1.mat')
m = np.shape(X)[0]
```

```
# displaying a random sample of the data
sample = np.random.choice(X.shape[0],10)
plt.imshow(X[sample,:].reshape(-1,20).T)
plt.axis('off')

K = 10  # number of classes
reg = 0.1 # regularization factor

# Training the models
Th = oneVsAll(X,y,K,reg)

# Computing precision
X = np.hstack([np.ones([m,1]),X])
p = precision(Th,X,y)
print('Logistical model accuracy : {0:1.3g}'.format(p))

#%%
"""
2 - Neural networks
"""

# Loading the weights
weights = loadmat('ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']

# Computing the precision
p = precision_neur(theta1,theta2,X,y)
print('Neural network model accuracy : {0:1.3g}'.format(p))
```