Aprendizaje automático y minería de datos

Práctica 3 – Regresión logística multi-clase y redes neuronales

Ramón Arjona Quiñones
Celia Castaños Bornaechea

Contenido

1. Regresión logística multi-clase

Resumen

Aplicar regresión logística multi-clase para reconocer los números escritos a mano de las imágenes.

1.1 Carga de los datos

Se cargan las matrices de los casos de entrenamiento. Para el vector *y* hay que establecer los valores ya que en la matriz cargada son de 1 a 10 y para esta práctica los queremos del 0 al 9. Correspondiendo el 10 con el 0.

```
    data = loadmat('ex3data1.mat') # Devuelve un diccionario
    X = data['X'] # Cada ejemplo de entrenamiento en una fila (5000x400)
    y = data['y'].ravel() # 1 - 10 (el 10 es 0)
```

1.2 Vectorización de la regresión logística

Hay que entrenar diez clasificadores logísticos, uno por cada valor de salida. Para que sea eficiente se calcula de forma vectorizada. Además, se les añade el término de regularización.

Coste regularizado

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

```
def regularizedCost(theta, lamda: float, X, Y):
    '''
    Calcula el coste para los ejemplos, pesos y término de regularización
 dados
    '''
    #Variables auxiliares
    m = X.shape[0]
    H = sigmoid(np.matmul(X, theta))

    # Coste cuando Y = 1
    costeUno = np.dot(Y, np.log(H))
    # Coste cuando Y = 0
    costeCero = np.dot((1 - Y), np.log(1 - H))
    # Término de regularización (sumado al coste original)
    regTerm = lamda / (2 * m) * np.sum(theta**2)

    return -1 / m * (costeUno + costeCero) + regTerm
```

Gradiente regularizado

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{para } j = 0$$
$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{para } j \geq 1$$

```
def regularizedGradient(theta, lamda: float, X, Y):
    '''
    Calcula el gradiente con los ejemplos, pesos y término de regularizac
ión dados
    '''
    #Variables auxiliares
    m = X.shape[0]
    H = sigmoid(np.matmul(X, theta))

    # Cálculo del gradiente
    grad = (1/ m) * np.matmul(X.T, H-Y)

    # No queremos usar la primera componente de theta (nos la guardamos)
    aux = theta[0]
    theta[0] = 0

    # Término de regularización
    regTerm = lamda / m * theta

    #Devolvemos su valor a theta
    theta[0] = aux

    return grad + regTerm
```

1.3 Clasificación One Vs All

Hay que entrenar un clasificador por regresión logística para cada una de las diez clases del conjunto de datos.

```
def oneVsAll(X: np.array, y: np.array, num_etiquetas: int, reg: float):
    '''
    Implementa la regresión lineal multiclase (reg = término de regulariz
ación)
    '''
    #Creamos la matriz de thetas
    thetas = np.zeros((num_etiquetas, X.shape[1]))

    #Clasificador para cada una de las etiquetas
    for i in range (num_etiquetas):
        # Vector de 'y' para la iteración concreta
        iterY = np.copy(y)
```

```
        if(i == 0):
            iterY = np.where (iterY == 10, 1, 0)
        else:
            iterY = np.where (iterY == i, 1, 0)

        # Calculamos el vector de pesos óptimo para ese clasificador
        thetas[i] = fmin_tnc(func = regularizedCost, x0=thetas[i], fprime
=regularizedGradient, args=(reg, X, iterY.ravel()), messages=0)[0]

    return thetas
```

Se comprueba para λ = 0,1 el porcentaje de precisión del entrenador.

```
El entrenador tiene una precisión del  96.46 %
```

Código del Ejercicio

```
def Ejercicio1(lamda):
    '''
    Regresión logística OneVsAll
    '''
    # Leemos los datos de las matrices (en formato .mat) con 5k ejemplos
de entrenamiento
    # Cada ejemplo es una imagen de 20x20 pixeles, cada uno es un número
real en escala de grises
    data = loadmat('ex3data1.mat') # Devuelve un diccionario
    X = data['X'] # Cada ejemplo de entrenamiento en una fila (5000x400)
    y = data['y'].ravel() # 1 - 10 (el 10 es 0)
    num_etiquetas = 10

    #Inicializamos theta y ponemos la columna de 1's a las X
    m = X.shape[0]
    unos = np.ones((m, 1))
    unosX = np.hstack((unos, X))

    # Resolvemos el one vs All (debería estar bien)
    thetas = oneVsAll(unosX, y, num_etiquetas, lamda)
    z = sigmoid(hMatrix(unosX, thetas))

    print("El entrenador tiene una precisión del ", calcula_porcentaje(y,
 z, 4), "%")
```

2. Redes neuronales

Resumen

Utilizando las matrices de pesos proporcionadas, evaluar su precisión sobre los ejemplos de entrenamiento.

2.1 Propagación hacia delante

Los datos de entrada avanzan hacia delante a través de la red neuronal. Cada capa oculta recibe los datos, los procesa según la configuración y los pasa a la siguiente capa.

```
def forward_prop(X, theta1, theta2):
    m = X.shape[0]

    a1 = np.hstack([np.ones([m, 1]), X])
    z2 = np.dot(a1, theta1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])
    z3 = np.dot(a2, theta2.T)
    h = sigmoid(z3)

    return a1, z2, a2, z3, h
```

Se comprueba el porcentaje de precisión de la red.

```
El entrenador tiene una precisión del  97.52 %
```

Código del Ejercicio

```
def Ejercicio2():
    '''
    Redes neuronales con unos pesos ya dados
    '''
    #Cargamos los datos
    data = loadmat('ex3data1.mat')
    X = data['X'] # (5000x400)
    y = data['y'].ravel()
    y = y - 1 #Porque están de 1 - 10 y los queremos del 0 - 9
    num_etiquetas = 10

    # Guardamos las matrices de theta (leídas de archivo) en una lista
    weights = loadmat ( "ex3weights.mat" )
    theta1, theta2 = weights ["Theta1"], weights ["Theta2"]
    # Theta1 es de dimensión 25 x 401
    # Theta2 es de dimensión 10 x 26

    # Hacemos la propagación hacia delante
    a1, z2, a2, z3, h = forward_prop(X, theta1, theta2) # z es de 5000x10

    print("El entrenador tiene una precisión del ", calcula_porcentaje(y,
 h, 4), "%")
```