# PRÁCTICA 3. MEMORIA TÉCNICA

*Sergio Gavilán Fernández* *sgavil01@ucm.es*

*Alejandro Villar Rubio* *alvill04@ucm.es*

## Parte 1. Regresión logística multi-clase

### Proceso

### Conjunto de datos y funciones principales

Se ha partido de un conjunto de datos proporcionados el archivo "ex3data1.mat". Este conjunto está formado por 5000 imágenes de 20x20 píxeles donde cada píxel está representado por un número real que indica la intensidad en escala de grises de ese punto. Cada matriz de 20x20 se ha desplegado para formar un vector de 400 componentes que ocupa una fila de la matriz *X*. De esta forma, *X* es una matriz de 5000x400 donde cada fila representa la imagen de un número escrito a mano.

El vector *y* es un vector de 5000 componentes que representan las etiquetas de los ejemplos de entrenamiento. El "0" se ha etiquetado como "10", manteniendo las etiquetas naturales del "1" al "9" para el resto de los números.

**Ilustración 1. Muestra de una selección aleatoria de 10 ejemplos**

Se han implementado las versiones vectorizadas y regularizadas de las funciones de gradiente y de coste haciendo uso de la función sigmoide.

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

**Ilustración 2. Versión vectorizada de la función de gradiente regularizada**

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

**Ilustración 3. Versión vectorizada de la función de coste regularizada**

$$g(z) = \frac{1}{1+e^{-z}}$$

**Ilustración 4. Función sigmoide**

## Entrenamiento de clasificadores

El siguiente paso es entrenar un clasificador por regresión logística para cada una de las clases. Para ello se ha implementado la función *oneVsAll(X, y, num_etiquetas, reg)* que devuelve el resultado en una matriz donde cada fila de $\theta$ corresponde al clasificador de cada una de las clases. En nuestro caso se han usado los siguientes parámetros:

- X: Matriz X perteneciente al conjunto de datos.
- Y: Vector Y perteneciente al conjunto de datos.
- num_etiquetas: Número de clases, en nuestro caso, 10.
- reg: Término de regularización, en nuestro caso, 0.1.

Para obtener esta matriz se realizan los siguientes pasos:

1. Se le añade el término independiente a cada las columnas, quedando la matriz *X* con un tamaño de 5000x401.
2. Se crea un vector de $\theta$ de un tamaño de 401 inicializados a 0.
3. Por cada clase del 1 al 10 (bucle del 0 al 9) se llama a la función *scipy.optimize.fmin_tnc.*
4. El resultado se guarda en una matriz, la cual será la solución al final del número de iteraciones.

## Cálculo de probabilidad

Una vez se obtenga la "matriz resultado", calculamos para cada ejemplo de entrenamiento cuál es la "probabilidad" de que pertenezca a cada una de las clases. Se ha implementado la función *testClassificator(Theta, X, Y)* para realizar este porcentaje: para cada ejemplo de entrenamiento se pone a prueba cada uno de los clasificadores; se calcula el sigmoide entre el clasificador y el ejemplo,; se escoge el mejor resultado en cada momento, es decir, el mayor de todos, y por último comprobamos que la clase a la que representa el mejor clasificador es aquella que en el conjunto de datos pertenece a ese ejemplo.

Siguiendo estos pasos obtenemos una probabilidad del 96.46%

# Implementación

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


def mostrar_numeros_ejemplo(X):
    sample = np.random.choice(X.shape[0], 10)
    plt.imshow(X[sample, :].reshape(-1, 20).T)
    plt.axis("off")

    plt.show()

    return plt


def f_gradiente(Theta, X, Y, lam):
    m = len(X)
    tempTheta = np.r_[[0], Theta[1:]]
    return (((1 / m) * np.dot(X.T, sigmoide(np.dot(X, Theta)) - np.
ravel(Y)))
            + ((lam / m) * tempTheta))


def f_coste(Theta, X, Y, lam):
    m = len(X)
    return (((-
1 / m) * (np.dot(np.log(sigmoide(np.dot(X, Theta))).T, Y)
                          + np.dot(np.log(1 - sigmoide(np.dot(X, The
ta))).T, (1 - Y))))
            + ((lam / (2 * m)) * np.sum(Theta**2, initial=1)))


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


def testClassificator(Theta, X, Y):
    aciertos = 0
    for m in range(X.shape[0]):  # Para cada ejemplo de entrenamien
to
        mejorClassificator = -1
        index = 0
        for j in range(Theta.shape[0]):  # Ponemos a prueba cada cl
asificador
```

```
            res = sigmoide(np.dot(Theta[j], X[m]))
            if(res > mejorClassificator):
                mejorClassificator = res
                index = j+1
        if(index == Y[m]):
            aciertos += 1

    print("Porcentaje:", aciertos / X.shape[0])


def oneVsAll(X, y, num_etiquetas, reg):

    X = np.hstack((np.ones((X.shape[0], 1)), X))  # (5000,401)

    matResult = np.zeros((num_etiquetas, X.shape[1]))  # ()
    Theta = np.zeros(X.shape[1])  # (401,)
    m = X.shape[1]

    for n in range(num_etiquetas):
        nuevaY = np.array((y == n+1)*1)
        result = opt.fmin_tnc(func=f_coste, x0=Theta,
                              fprime=f_gradiente, args=(X, nuevaY,
reg))

        matResult[n] = result[0]  # (10,401)

    testClassificator(matResult, X, y)


def main():
    data = loadmat("ex3data1.mat")

    Y = data["y"]
    X = data["X"]

    plt = mostrar_numeros_ejemplo(X)

    oneVsAll(X, Y, 10, 0.1)

main()
```

# Parte 2. Redes neuronales

## Proceso

### Conjunto de datos y funciones principales

En esta sección de la práctica se parte del mismo conjunto de datos que en el primer ejercicio, es decir, el proporcionado por el archivo "ex3data1.mat". En esta ocasión también se usan unos pesos predeterminados (formados por el resultado de haber entrenado la red neuronal) que se encuentran en "ex3weights.mat". Estos pesos se dividen en dos matrices "Theta" con dimensiones 25x401 y 10x26, respectivamente.

Ahora solo se usará la función sigmoide anteriormente nombrada, ya que las funciones de gradiente y de coste no son necesarias al tener ya calculadas las matrices de pesos.

### Propagación hacia delante y precisión

El objetivo de este ejercicio es implementar la propagación hacia delante para computar el valor de $h_\theta(x^{(i)})$ para cada ejemplo $i$. Además de calcular la precisión de la red neuronal.

Para ello, primero hay que ser consciente de la estructura de la red. En este caso se pide que esté formada por tres capas, con 400 unidades en la primera capa (además de la primera fijada siempre a +1), 25 en la capa oculta y 10 en la capa de salida.

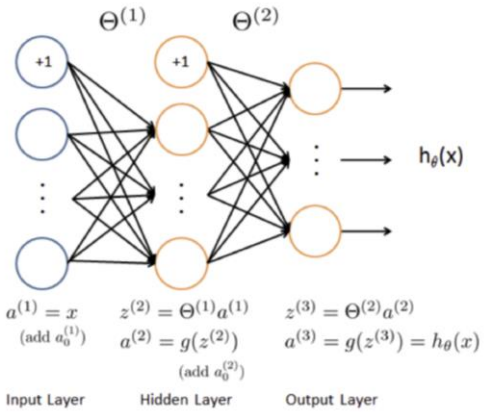


**Ilustración 5. Ejemplo de la estructura de una red neuronal sencilla**

La propagación hacia delante se ha programado tal y como viene en el temario de "Aprendizaje de redes neuronales", por lo que no es una función escalable, solo funciona con este tipo de estructura. El resultado a tener en cuenta es la variable llamada "h", de dimensiones 5000x10, que contiene un valor entre [0, 1] que indica "cuanta probabilidad hay de que ese ejemplo pertenezca a esa clase". Este valor se usará para calcular la precisión.

De la misma forma que en la regresión logística, se interpreta que la clase asignada por la red neuronal a un ejemplo es la correspondiente a la salida de la red con el máximo valor. Después de calcular esta precisión obtenemos una precisión del 97.52%

Implementación

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

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

def testClassificator(h, Y):
    aciertos = 0
    for i in range (h.shape[0]):
        max = np.argmax(h[i])

        max += 1

        if max == Y[i]:
            aciertos += 1

    precision = (aciertos / h.shape[0]) * 100
    print("La precisión es del", precision)


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

    a1 = np.hstack([np.ones([m, 1]), X])    # (5000, 401)
    z2 = np.dot(a1, theta1.T)   # (5000, 25)

    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])  # (5000, 26)
    z3 = np.dot(a2, theta2.T)   # (5000, 10)

    h = sigmoid(z3) # (5000, 10)

    return a1, z2, a2, z3, h


def main():
    data = loadmat("ex3data1.mat")
    Y = data["y"] # Y (5000, 1)
    X = data["X"] # X (5000, 400)

    weights = loadmat("ex3weights.mat")
    theta1, theta2 = weights["Theta1"], weights["Theta2"]
```

```
    # Theta1 es de dimensión 25 x 401
    # Theta2 es de dimensión 10 x 26

    delta1 = np.zeros(theta1.shape)
    delta2 = np.zeros(theta2.shape)

    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

    testClassificator(h, Y)

main()
```