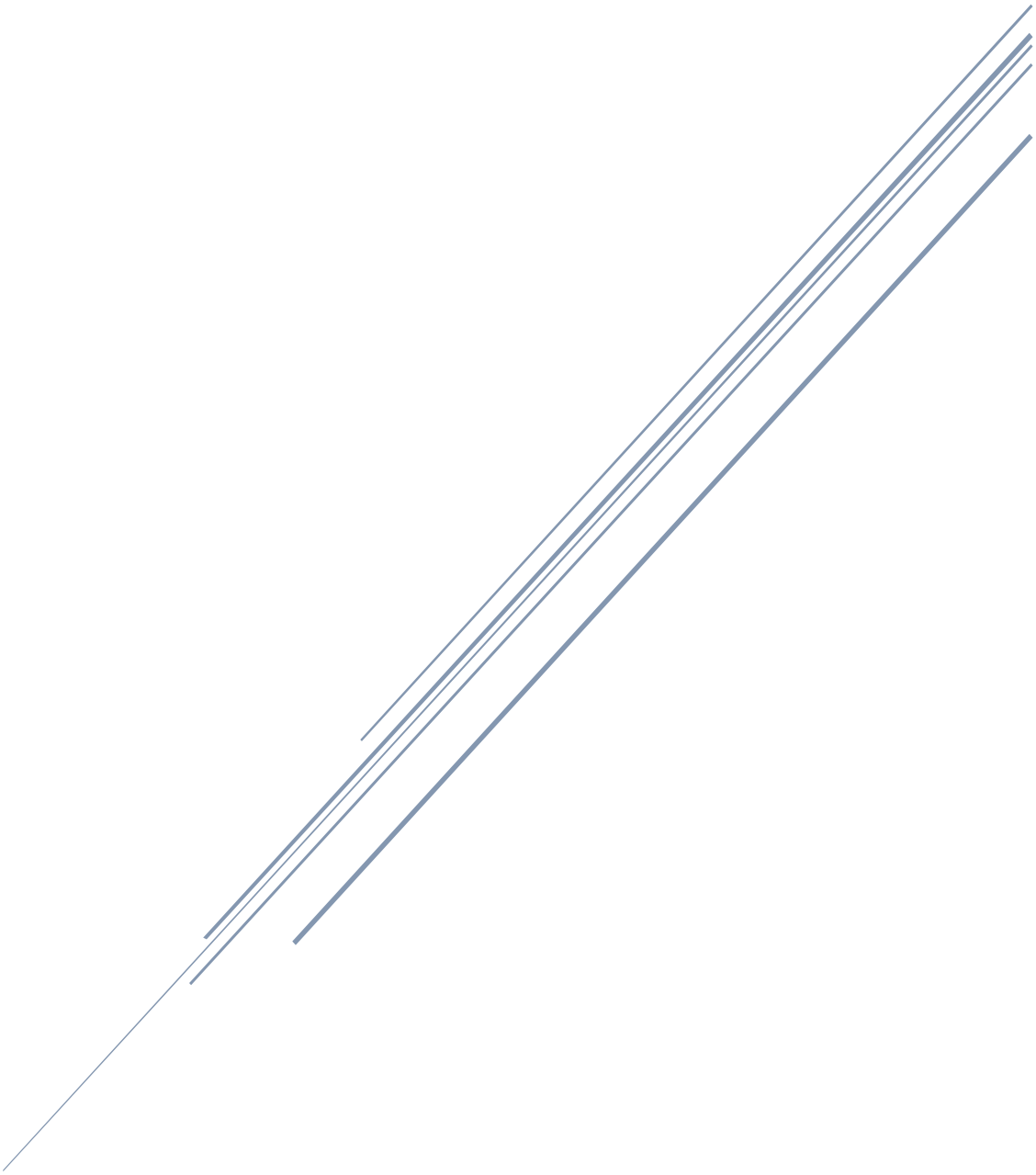
APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 3



FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta cuarta práctica es aplicar la regresión logística multiclase para determinar cifras a partir de imágenes. También utilizaremos redes neuronales ya entrenadas para realizar el mismo cometido.

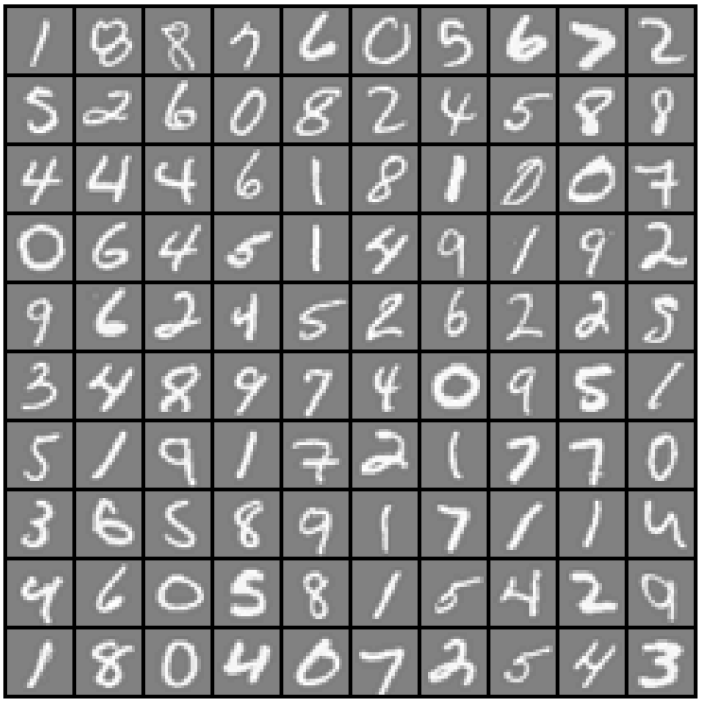
2. Desarrollo e implementación

2.1. Regresión logística multiclase

Comenzaremos visualizando en la figura 1 los datos que vamos a tratar (una serie aleatoria de 100 ejemplos de entrenamiento).

```
>> m = size(X, 1);
>> rand_indices = randperm(m);
>> sel = X(rand_indices(1:100), :);
>> displayData(sel);
```

Figura 1



Partimos de la función de coste regularizada implementada en la práctica anterior de forma vectorizada, cuya implementación podemos ver en la figura 2:

```
function [J, grad] = firstCostFunction(theta, X, y, lambda)

  numCols = columns(X);
  numFils = rows(X);

  grad = (1/numFils) * ((hipotesis(X', theta) - y') * X);
  grad = grad';
  grad(2:numCols, 1) = grad(2:numCols, 1) + ((lambda/numFils) *
theta(2:numCols));


  J = ((1/numFils) * sum((-y .* (log(hipotesis(X', theta)))') - ((1 - y) .* (log(1 -
hipotesis(X', theta)))'))) + ((lambda/(2*numFils)) * sum(theta.^2));

endfunction
```

Figura 2

Esta función de coste es utilizada en la función *oneVsAll*, que entrena varios clasificadores por regresión logística y devuelve el resultado en una matriz 'all_thetha", donde la fila i-ésima corresponde a la etiqueta i-ésima. Dentro de esta función, llamaremos a la función de coste con la matriz "X" con una columna de unos añadidos. En esta ocasión se hará uso de la función de optimización *fmincg*, ya que es más eficiente que *fminunc* cuando trabajamos con ejemplos con muchos parámetros. Podemos ver su implementación en la figura 3:

```
function [all_theta] = oneVsAll(X, y, num_etiquetas, lambda)
  n = columns(X);
  X = [ones(rows(X), 1) X];

  initial_theta = zeros (n + 1, 1);

  options = optimset ( "GradObj" , "on" , "MaxIter" , 50);

  for i = 1:num_etiquetas
    all_theta(i, :) = fmincg (@(t)(firstCostFunction(t , X, ( y == i ), lambda)),
initial_theta, options);
  endfor

endfunction
```

Figura 3

Ejecutamos esta última función con los parámetros “num_etiquetas” igual a 10 y un valor de “lambda” igual a 0, lo que nos devuelve la matriz de thetas.

```
>> [all_theta] = oneVsAll(X, y, 10, 0);
```

Figura 4

Conseguido los valores de theta mejor ajustados, vamos a calcular el porcentaje de aciertos de nuestro sistema. Para ello hemos implementado la siguiente función presente en la figura 5:

```
function porcentaje = damePorcentaje(theta, X, y)
  numFils = rows(X);
  bienPredecidos = 0;
  h = hipotesis(X', theta');

  [maximo clase] = max(h);

  comparacion = (clase' == y);

  bienPredecidos = length(find(comparacion == 1));
  porcentaje = (bienPredecidos/length(comparacion)) * 100;
endfunction
```

Figura 5

Se consigue el resultado que podemos observar en la figura 6 (*damePorcentaje* necesita la columna de unos en la matriz X para poder funcionar):

```
>> miX = [ones(5000,1) X];
>> porcentaje = damePorcentaje(all_theta, miX, y)
     porcentaje =  95.100
```

Figura 6

Llamando a la función *oneVsAll* con el parámetro “lambda” igual a 1 obtenemos un porcentaje menor de aciertos (94.060%).

2.2. Redes neuronales

En esta segunda parte utilizaremos unos vectores “Theta1” y “Theta2” ya proporcionados para una red neuronal ya entrenada sobre los ejemplo de entrenamientos anteriormente utilizados para evaluar su precisión sobre los mismo ejemplos.

Partimos de una red neuronal que tiene la estructura que presenta la figura 7:

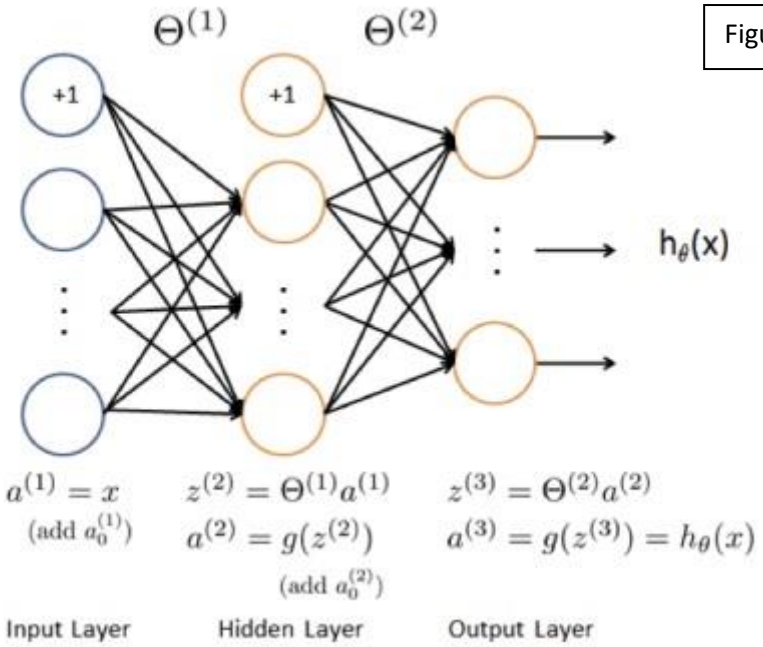


Figura 7

Tenemos una red de tres capas. La primera está compuesta por tantas unidades como columnas tiene nuestra matriz de datos (400 unidades). La segunda capa está compuesta por 25 unidades. En la tercera capa estará compuesta por 10 unidades, una por cada clase. Para poder operar, las dos primeras capas tendrán una unidad extra (de valor 1).

Para evaluar la precisión de la red neuronal hemos implementado la función *propagaciónHaciaDelante*, que se encarga de obtener el valor de salida de la red para cada ejemplo i-ésimo. Podemos ver su implementación en la figura 8.

```
function h = propagacionHaciaDelante(theta1, theta2, X)
  a2 = sigmoide(theta1 * X');
  a2 = a2';
  a2 = [ones(rows(a2), 1) a2];

  h = sigmoide(theta2 * a2');
endfunction
```

Figura 8

Esta última función nos devuelve en “h” una matriz de 10 filas y 5000 columnas, siendo cada columna un ejemplo de entrenamiento y cada fila la probabilidad de que el ejemplo pertenezca a la clase i-ésima.

Para el cálculo del porcentaje utilizaremos una función similar a la utilizada en la regresión. Aprovechamos la funcionalidad de la función max, que nos devuelve el porcentaje máximo y el índice donde se ubica el máximo dentro de cada columna. Podemos ver su implementación en la figura 9:

```
function porcentaje = damePorcentajeRedNeuronal(h, y)
  [maximo clase] = max(h);

  comparacion = (clase' == y);

  bienPredecidos = length(find(comparacion == 1));
  porcentaje = (bienPredecidos/length(comparacion)) * 100;
endfunction
```

Figura 9

Tras la ejecución obtenemos el resultado de la figura 10:

```
>> h = propagacionHaciaDelante(Theta1, Theta2, miX);

>> porcentaje = damePorcentajeRedNeuronal(h, y)
porcentaje =  97.520
```

Figura 10

Para la realización de la práctica se ha partido de la implementación de las funciones *hipótesis* y *sigmoide* utilizadas en la práctica 2. Podemos ver su implementación el figura 11:

```
function s = sigmoide(z)
  s = 1 ./ (1 + exp(-z));
endfunction


function h = hipotesis(x, theta)
  h = sigmoide(theta' * x);
endfunction
```

Figura 11