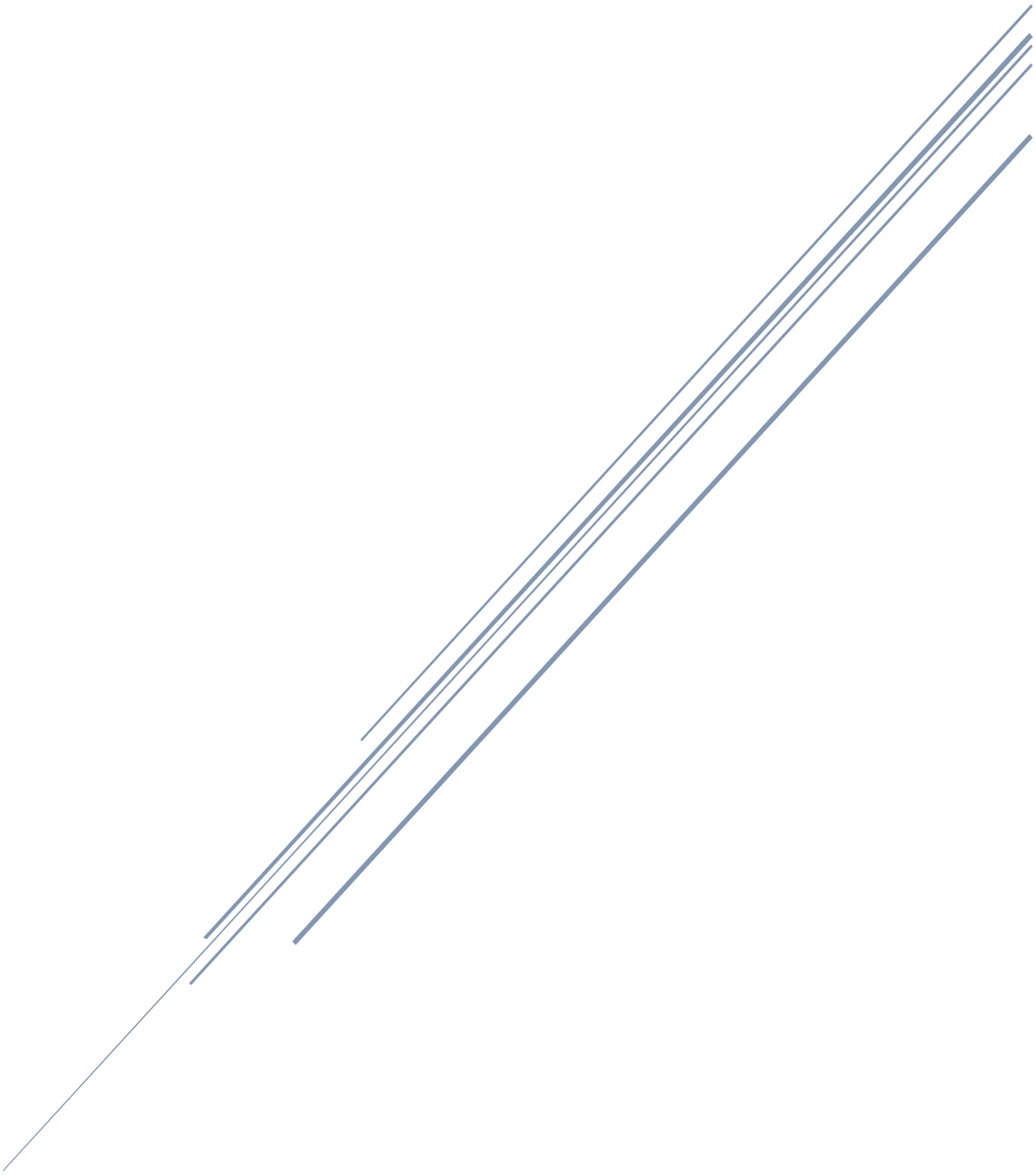
APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 1



FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta segunda práctica es aplicar la regresión lineal, tanto de una variable como de varias.

2. Desarrollo e implementación

2.1. Regresión lineal con una sola variable

En primer lugar, necesitamos aplicar el método de descenso de gradiente para obtener los parámetros θ.

La ecuación a seguir es de la siguiente manera:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Traducido a código, tenemos la siguiente función en la figura 1:

```
function [a, b]  = descensoGradiente(a, b, alfa, iteraciones, datos)
 for j = 1:iteraciones
   #probamos que vaya disminuyendo la funcion de coste
   #miCoste = coste(a, b, datos)
   sumatorio0 = sum(hipotesis(datos(:, 1), a, b) - datos(:, 2));
   sumatorio1 = sum((hipotesis(datos(:, 1), a, b) - datos(:, 2)) .* datos(:, 1));
   a = a - alfa * (1/rows(datos)) * sumatorio0;
   b = b - alfa * (1/rows(datos)) * sumatorio1;
 endfor
endfunction
```

Figura 1

Invocamos a la función *descensoGradiente* con una "a" (θ_0) y una "b" (θ_1) inicializadas a 0, un alfa con un valor de 0,01, por ejemplo, así como un número de iteraciones igual a 1500.

Tras la ejecución de la función obtenemos unos valores de θ_0 y θ_1 (necesarios para poder calcular la función de hipótesis) mínimos que hacen mínima la función de coste.

La función de *hipótesis*, que sigue la siguiente fórmula se muestra en la figura 2:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

```
function y = hipotesis(x, teta0, teta1)
  y = teta0 + teta1 * x;
endfunction
```

Figura 2

Esta última función es necesaria para calcular la función de coste, que sigue la siguiente fórmula y que podemos ver implementada en la figura 3:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

```
function y = coste(teta0, teta1, datos)
  y = 1/(2*rows(datos)) * sum((hipotesis(datos(:, 1),teta0, teta1) - datos(:, 2)).^2);

endfunction
```

Figura 3

Una vez tenemos desarrolladas todas las partes, comprobamos el funcionamiento. Para ello cargamos unos datos proporcionados mediante un fichero (ex1data1.txt). Después, creamos una θ_0 y una θ_1 para inicializarlas a 0 para pasárselas a la función *descensoGradiente*, como podemos observar en la figura 4.

```
>> t0 = 0;
>> t1 = 0;
>> [t0, t1]  = descensoGradiente(t0, t1, 0.01, 1500, ex1data1)
t0 = -3.8781
t1 =  1.1913
```

Figura 4

Una vez obtenido el valor de ambos parámetros, vamos a representar la gráfica de la recta que más se ajusta a los puntos. Para poder pintar la gráfica ejecutamos los siguientes comandos presentes en la figura 5:

```
>> miRecta = @(x)hipotesis(x, t0, t1)
miRecta =

@(x) hipotesis (x, t0, t1)

>> fplot(miRecta, [5 25],50)
>> hold on
>> plot(ex1data1(:,1),ex1data1(:,2),"xr")
```

Figura 5

La figura 6 ilustra la gráfica obtenida.

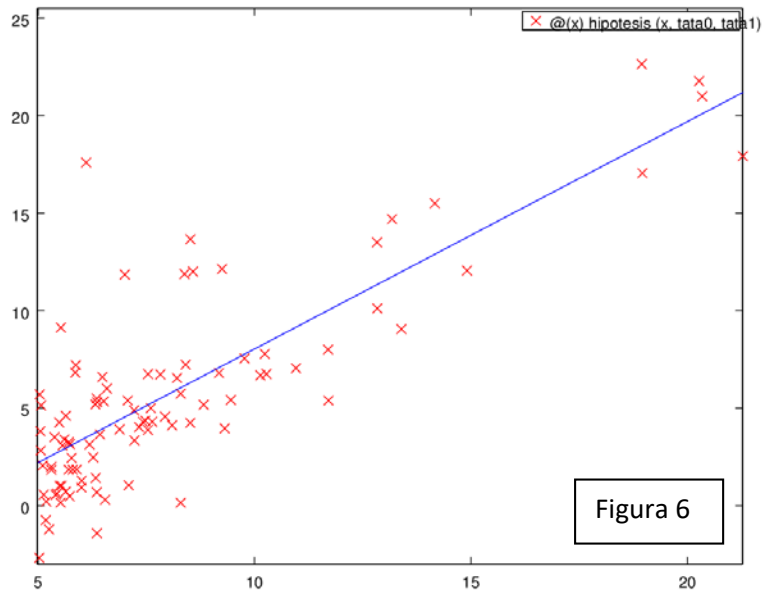


Figura 6

Para enriquecer los resultados, vamos a generar otras gráficas, donde se ve el resultado de llamar a la funciones de Octave *Surface* y *contorne* con los valores de la función de coste en el intervalo $\theta_0 \, \epsilon \, [-10, 10] \, y \;\; \theta_1 \, \epsilon \, [-1, \;\; 4]$. Para ello hemos implementado la siguiente función auxiliar, como se muestra en la figura 7:

```
function [X,Y,Z] = misurface(datos)
  vector1 = [-10:0.2:10];
  vector2 = [-1:0.1:4];

  [X Y] = meshgrid(vector1, vector2);

  for j=1:length(vector2)
    for i=1:length(vector1)
      Z(j,i) = coste(vector1(i),vector2(j), datos);
    endfor
  endfor

endfunction
```

Figura 7

Llamamos a *misurface* con los siguientes comandos de la figura 8:

```
>> [X Y Z] = misurface(ex1data1);
>> surface(X,Y,Z);
```

Figura 8

Se genera la gráfica que podemos observar en la figura 9:

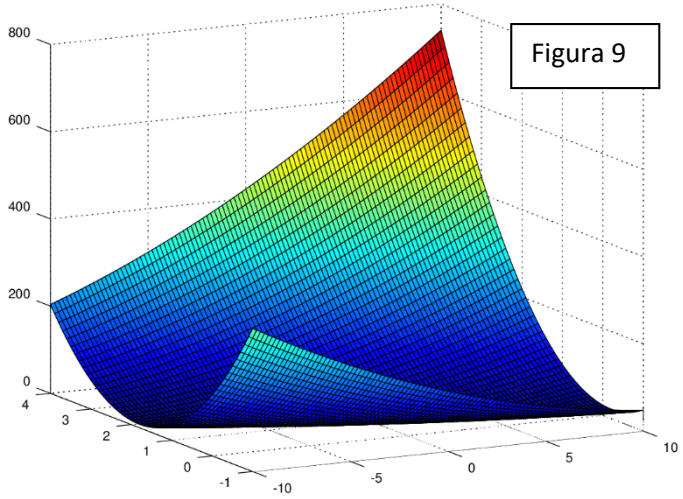


Figura 9

También podemos generar otro tipo de gráfica con la función *contorne*, para ellos ejecutamos los siguientes comandos mostrados en la figura 10:

```
>> contour(X,Y,Z,logspace(-2,3,20))
>> hold on
>> plot(t0,t1,"xr")
```

Figura 10

La gráfica que se genera y podemos ver en la figura 11 nos ilustra que el punto (t0, t1) está situado en el mínimo de la función de coste.

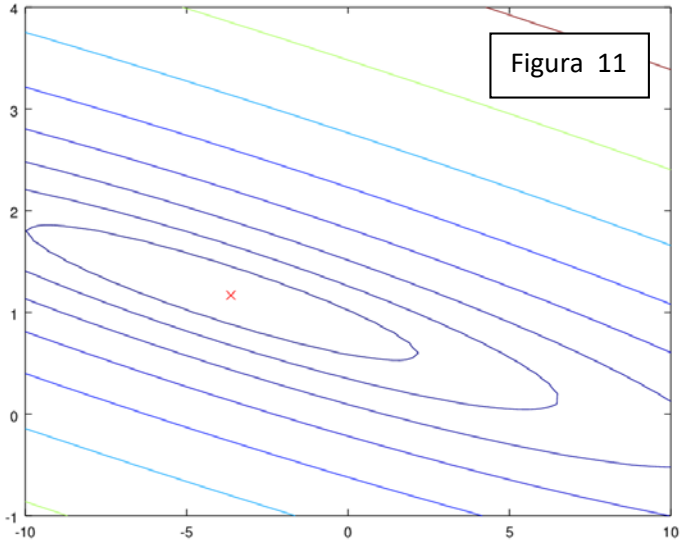


Figura 11

2.2. Regresión lineal con varias variables

En esta segunda parte aplicaremos el método de regresión lineal con varias variables. A diferencia del anterior punto, los datos con los que trabajaremos tendrán que estar normalizados. Para ello implementamos la siguiente función, como podemos ver en la figura 12:

```
function [X_norm, mu, sigma] = normalizaAtributo(X)

  cols = columns(X)-1;
  matrizFtrs = X(:,1:cols);

  mu = mean(matrizFtrs);
  sigma = std(matrizFtrs);

  X_norm = [(matrizFtrs - mu)./sigma X(:,columns(X))];

endfunction
```

Figura 12

Ahora, la función de *descensoGradiente* sufre unas pequeñas modificaciones, como podemos ver en la figura 13:

```
function T = descensoGradienteMultiples(T, alfa, iteraciones, datos)

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

  for j = 1:iteraciones
    sumatorio = zeros(numCols - 1, 1);
    for col = 1: numCols - 1
      for fila = 1:numFils
          X = datos(fila,1:numCols - 1);
          sumatorio(col, 1) =  sumatorio(col, 1) + (hipotesisMultiples(X', T) -
datos(fila, numCols)) .* datos(fila, col);
      endfor
    endfor
    J(j) = costeMultiples(T,datos);
    T = T - (alfa * (1/rows(datos))* sumatorio);
  endfor
  ################################################
  ##Descomentar para pintar la funcion de coste ##
  ################################################
  #plot(0:(iteraciones-1), J(1:iteraciones), '-')
  #xlabel('Numero de iteraciones')
  #ylabel('Coste J')

endfunction
```

Figura 13

Ahora, los valores de θ los guardamos en un vector columna llamado T. Al llamar a la función de hipótesis, hacemos la traspuesta de la matriz de datos (que no contiene las "y").

Adaptando la nueva función de hipótesis, su nueva implementación queda reflejada en la figura 14:

```
function y = hipotesisMultiples(X, T)
  y = T' * X;
endfunction
```

Figura 14

Del mismo modo, realizamos una implementación vectorizada de la función de coste que sigue la siguiente fórmula y que podemos ver implementada en la figura 15:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

```
function w = costeMultiples(T, datos)
  colsFtr = columns(datos) - 1;
  X = datos(:,1:colsFtr);
  y = datos(:,colsFtr + 1);

  w = (1/(2*rows(datos))) * ((X * T - y )' * (X * T - y))

endfunction
```

Figura 15

Una vez tenemos todas las funciones implementadas comenzamos con la ejecución. Para ello vamos ejecutar los comandos de la figura 16:

```
>> T = zeros(3,1);
>> [X_norm, mu, sigma] = normalizaAtributo(ex1data2);
>> X_norm = [ones(47,1) X_norm ];
>> T = descensoGradienteMultiples(T, 0.01, 1500, X_norm)
T =

  3.4041e+05
  1.1063e+05
 -6.6493e+03
```

Figura 16

Inicializamos con ceros el vector columna de θ (T), normalizamos los datos con los que vamos a trabajar y una vez normalizados añadimos una columnas de unos. Una vez tenemos la matriz de datos terminada aplicamos el descenso de gradiente obteniendo los siguientes valores de las distintas θ.

Si descomentamos las últimas líneas de la figura 13 podremos obtener una gráfica que nos indique con que alfa nos disminuye la función de coste más rápidamente con menos iteraciones. Para ello, ejecutamos el descenso de gradiente con varias alfas, tal y como se muestra en la figura 17, obteniendo la gráfica de la figura 18:

```
>> T = zeros(3,1);
>> T = descensoGradienteMultiples(T, 0.01, 50, X_norm);
>> hold on
>> T = zeros(3,1);
>> T = descensoGradienteMultiples(T, 0.1, 50, X_norm);
>> hold on
>> T = zeros(3,1);
>> T = descensoGradienteMultiples(T, 1, 50, X_norm);
```

Figura 17

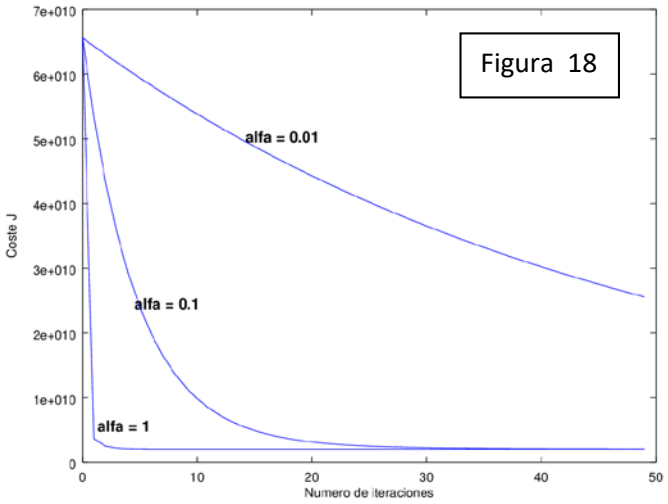


Figura 18

Podemos observar en la figura 18 que con alfa = 1 se disminuye más rápidamente con menos iteraciones. Por lo tanto usaremos alfa = 1. Vamos a comprobar de dos maneras distintas como obtenemos el mismo resultado. En primer lugar usaremos la función de *descensoGradienteMultiples* con el alfa anterior para obtener nuestro vector de θ. Para ello ejecutamos los comandos de la figura 19:

```
>> T = zeros(3,1);
>> T = descensoGradienteMultiples(T, 1, 100, X_norm)
T =

 3.4041e+005
 1.1063e+005
-6.6495e+003
```

Figura 19

Ahora predecimos el precio de una casa 1650 pies cuadrados y 3 habitaciones con los valores obtenidos en la figura 19 de la siguiente manera (teniendo que normalizar estos datos):

$$Precio = \theta_0 + (\theta_1 * superficie) + (\theta_2 * nºhabitaciones)$$

$$Precio = 3.4041 * 10^5 + \left(1.1034 * 10^5 * \left(\frac{1650 - 2000.7}{794.70}\right)\right) + \left(-6.5252 * 10^3 * \left(\frac{3 - 3.1702}{0.76098}\right)\right) = 293076.35$$

En segundo lugar utilizaremos la técnica de la ecuación normal llamando a la función *descensoGradientesMultiplesEcNornal* de la figura 20 tal y como se realiza en figura 21:

```
function T = descensoGradienteMultiplesEcNormal(datos)

  colsFtr = columns(datos) - 1;
  X = datos(:,1:colsFtr);
  y = datos(:,colsFtr + 1);

  T = pinv(X'*X)*X'*y;
endfunction
```

Figura 20

```
>> T2 = zeros(3,1);
>> X_sinnorm = [ones(47,1) ex1data2];
>> T2 = descensoGradienteMultiplesEcNormal(X_sinnorm)
T2 =

 8.9598e+004
 1.3921e+002
 -8.7380e+003
```

Figura 21

De la misma manera que hemos predicho antes el precio lo vamos a hacer ahora con los valores de θ que hemos obtenido en la figura 21 (ahora con los datos sin normalizar):

$$Precio = 8.9598 * 10^4 + (1.3921 * 10^2 * 1650) + (-8.7380 * 10^3 * 3) = 293080.5$$

Como podemos observar, el valor es casi idéntico $293076.35 \cong 293080.5$