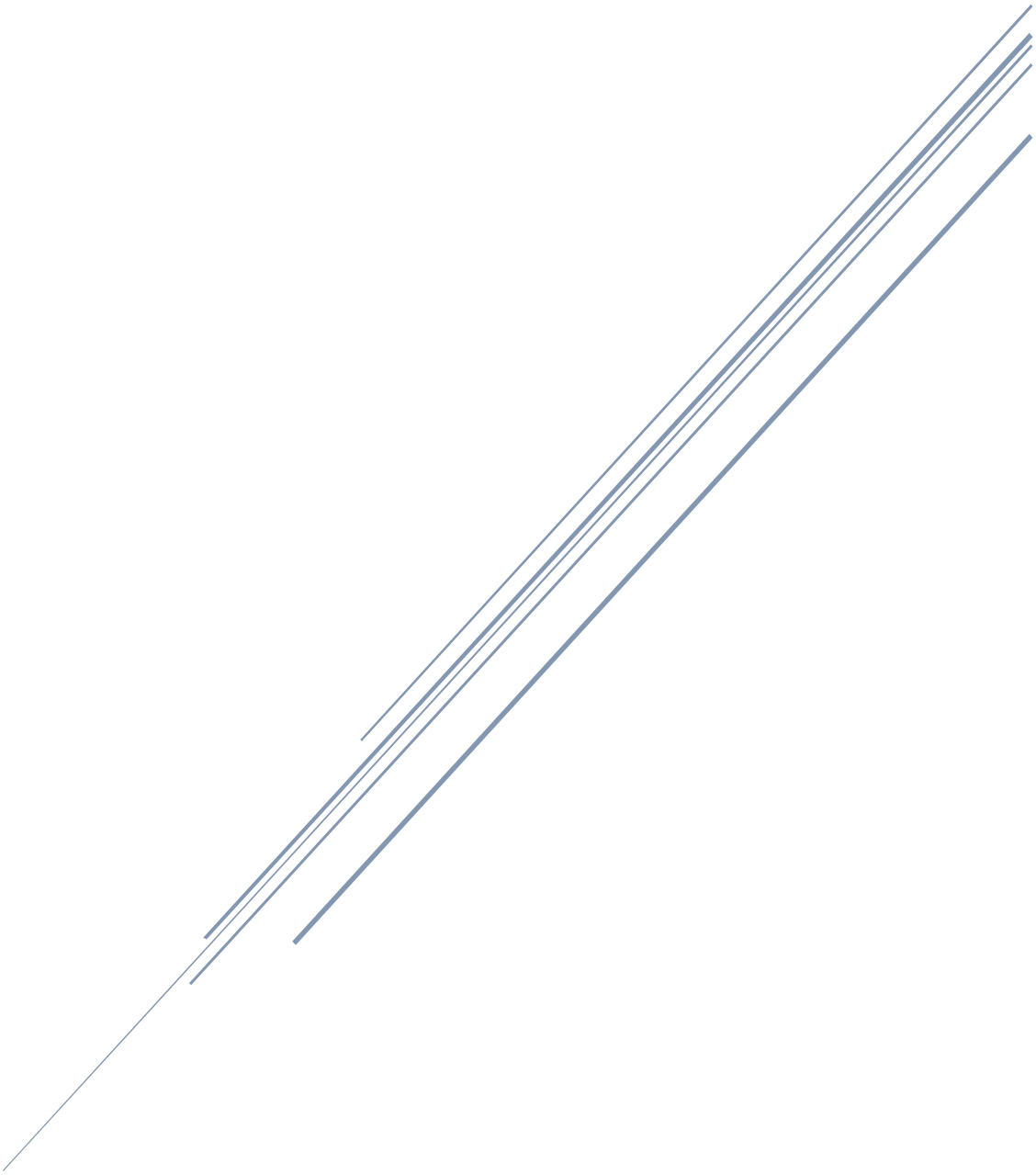
APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 2



FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta tercera práctica es aplicar la regresión logística y la regresión logística regularizada sobre un conjunto de datos.

2. Desarrollo e implementación

2.1. Regresión logística

En primer lugar vamos a visualizar los datos para ver a que nos estamos enfrentando. Para ello ejecutamos a una función que se encargue de la visualización. Dicha función la tenemos presenta en la figura 1:

```
function visualizaEx2data1(datos)
  negativos = find(datos(:, columns(datos)) == 0);
  plot(datos(negativos, 1), datos(negativos, 2), 'ko', 'MarkerFaceColor', 'y',
'MarkerSize', 7);
  xlabel("Exam1 score")
  ylabel("Exam2 score")

  hold on
  positivos = find(datos(:, columns(datos)) == 1);
  plot(datos(positivos, 1), datos(positivos, 2), '+', 'MarkerFaceColor', 'y',
'MarkerSize', 7);

  legend(gca(), 'Admitted', 'Not admitted');
endfunction
```

Figura 1

Posteriormente, se nos mostrará el gráfico de la figura 2:

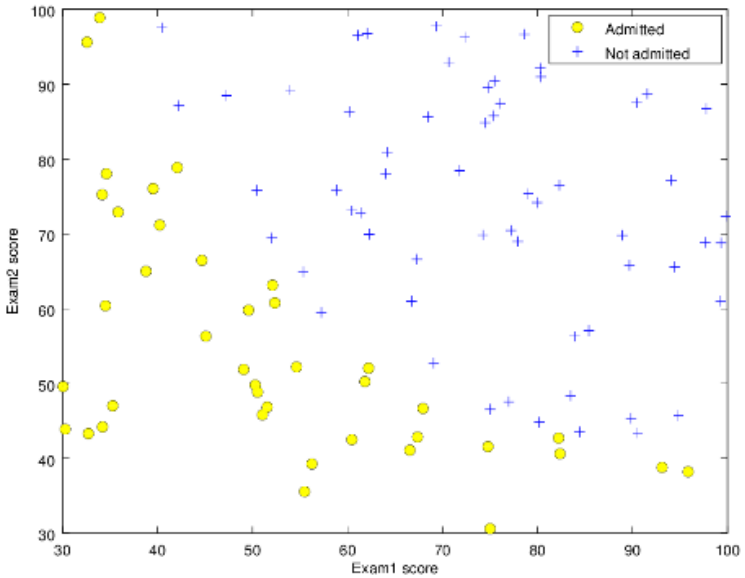


Figura 2

A diferencia de la regresión lineal, ahora haremos uso de la función sigmoide, utilizada por la función de hipótesis. La expresión sigmoide tiene la siguiente expresión:

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

Implementamos la función sigmoide, tal y como se observa en la figura 3:

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

Figura 3

Abstrayendonos, la función sigmoide la llamamos en nuestra nueva función d hipótesis, quedando implentada tal y como se observa en la figura 4:

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

Figura 4

Para que podamos multiplicar las matrices 'theta' y 'x', 'theta' será pasada como transpuesta a la función sigmoide.

Esta función de hipótesis la utilizaremos en la función de coste. A diferencia de en la regresión lineal, la función de cote, además de devolver un valor, devolverá un vector de thetas. Por lo tanto, la función de coste tendrá la siguiente expresión:

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

El gradiente de la función de coste tendrá esta otra expresión:

$$grad_j = \frac{1}{m}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})\,x_j^{(i)}$$

Estos dos cálculos estarán dentro de la función *coste*, que está implementado tal y como se observa en la figura 5a:

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

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

  sumatorio = zeros(numCols, 1);
  suma = 0;

  for col = 1: numCols
    for fila = 1: numFils
        sumatorio(col, 1) =  sumatorio(col, 1) + (hipotesis((X(fila, :))', theta) -
y(fila)) .* X(fila, col);
    endfor
  endfor

  for fila = 1:numFils
    suma = suma + (-y(fila)*log(hipotesis(X(fila, :)', theta))) - ((1 - y(fila))*log(1
- hipotesis(X(fila, :)', theta)));
  endfor

  grad = (1/numFils) * sumatorio;
  J = (1/numFils) * suma;

endfunction
```

Figura 5a

Tras el aviso del profesor, procedemos realizar la función de coste de forma vectorizada, tal y como se muestra en la figura 5b:

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


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

  grad = (1/numFils) * ((hipotesis(X', theta) - y') * X);
  grad = grad';

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

endfunction
```

Figura 5b

Probamos la función de coste con un vector de thetas inicializado a 0 y obtenemos el valor que podemos observar en la figura 6:

```
X = ex2data1(:,1:2);
X = [ones(100,1) X];
y = ex2data1(:,3);
theta = zeros(3,1);

[J, grad] = coste(theta, X, y)
J =  0.69315
grad =

 -0.10000
-12.00922
-11.26284
```

Figura 5

Para el correcto funcionamiento, a la matriz X le hemos añadido una primera columna de unos.

En esta práctica introducimos el uso de la función *fminunc*. En la anterior práctica, implementamos una función que se encargaba de encontrar las θ que minimizaran la función de coste. En esta práctica utilizaremos *fminunc* englobada en una función llamada *valorOptimo*, tal y como se muestra en la figura 6:

```
function [theta,cost] = valorOptimo(X, y, theta_inicial)
  opciones = optimset('GradObj', 'on', 'MaxIter', 400);
  [theta, cost] = fminunc(@(t)(coste(t, X, y)), theta_inicial,
opciones);
endfunction
```

Figura 6

Ejecutamos esta última función obteniendo el resultado que podemos observar en la figura 7:

```
>>theta_inicial = zeros(3,1);
>>[theta,cost] = valorOptimo(X, y, theta_inicial)
theta =

 -25.16127
0.20623
0.20147

cost =  0.20350
```

Figura 7

En la figura 8 visualizamos la recta que mejor separa los puntos (frontera de decisión):

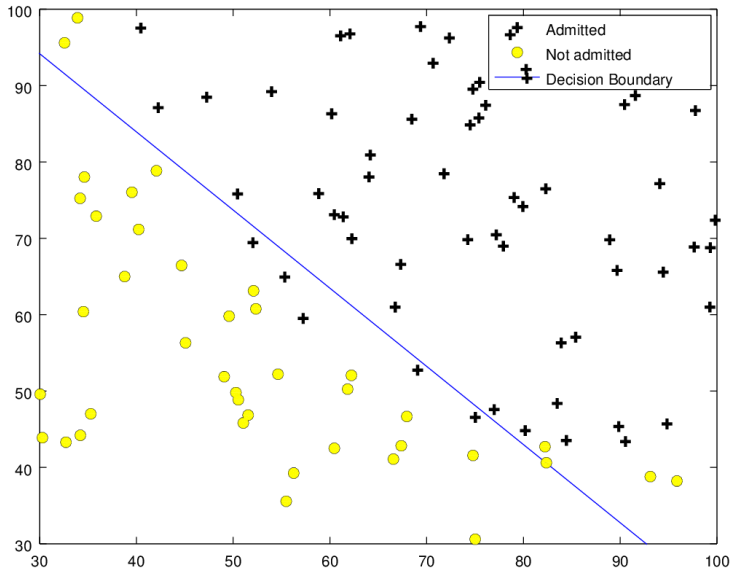


Figura 8

Una vez tenemos el resultado, vamos a evaluarlo. Para ello hacemos uso de la función *obtengoPorcentaje*, implementada tal y como podemos observar en la figura 9:

```
function p = obtengoPorcentaje(X, y, theta)
  numFils = rows(X);
  puntosTotales = 0;
  bienPredecidos = 0;

  for fila = 1:numFils
    h = hipotesis(X(fila,:)', theta);

    if(h >= 0.5)
      h = 1;
    else
      h = 0;
    endif

    if(y(fila) == h)
      bienPredecidos = bienPredecidos + 1;
    endif

  endfor
  p = (bienPredecidos/numFils) * 100;
endfunction
```

Figura 9

Ejecuto la anterior y obtengo el porcentaje de puntos en los que se han clasificado correctamente, tal y como podemos observar en la figura 10:

```
>> p = obtengoPorcentaje(X, y, theta)
p =  89
```

Figura 10

2.2. Regresión logística regularizada

En este apartado comenzaremos de la misma manera que el anterior. Visualizaremos los datos para ver a que nos estamos enfrentando. Para ello ejecutamos la función *visualizaEx2Data2* implementada en la figura 11:

```
function visualizaEx2data2(datos)

  negativos = find(datos(:, columns(datos)) == 0);

  plot(datos(negativos, 1), datos(negativos, 2), 'ko', 'MarkerFaceColor', 'y',
'MarkerSize', 7);
  xlabel("Microchip test 1")
  ylabel("Microchip test 2")

  hold on

  positivos = find(datos(:, columns(datos)) == 1);

  plot(datos(positivos, 1), datos(positivos, 2), '+', 'MarkerFaceColor', 'y',
'MarkerSize', 7);

  legend(gca(), 'y = 0', 'y = 1');
endfunction
```

Figura 11

Tras la ejecución observamos el gráfico de la figura 12.

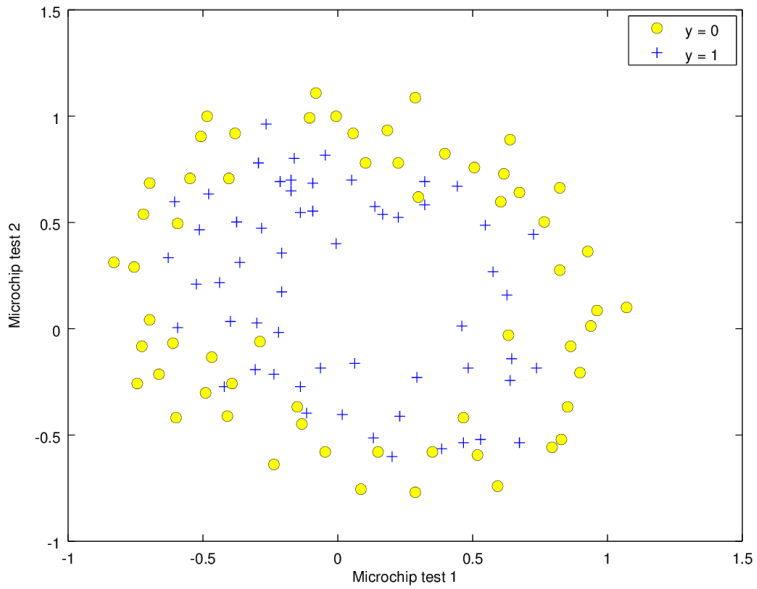


Figura 12

Debido a la imposibilidad de clasificar mediante una recta y con el objetivo de obtener un mejor ajuste vamos a hacer uso de una función proporcionada que nos permita añadir atributos a la descripción de ejemplos. Para ello haremos uso de la función *mapFeature*, tal y como podemos observar en la figura 13:

```
>> X1 = ex2data2(:,1);
>> X2 = ex2data2(:,2);
>> out = mapFeature(X1, X2);
```

Figura 13

Por otro lado, tenemos un cambio en la expresión de la función de coste y el cálculo de gradiente, que ahora seguirán las siguientes expresiones respectivamente:

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

Para el cálculo del gradiente identificaremos dos posibles casos. Por un lado cuando queramos calcular el gradiente de θ_0:

$$grad(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Para todos los demás se aplicarán la siguiente expresión:

$$grad(\theta) = \left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j$$

La implementación de la nueva función *costeRegularizado* la podemos observar en la figura 14a.

Ahora ejecutaremos esta última función tal y como se observa en la figura 15 obteniendo el siguiente resultado.

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

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

  sumatorio = zeros(numCols, 1);
  suma = 0;
  suma2 = 0;

  for col = 1: numCols
    for fila = 1: numFils
      sumatorio(col, 1) =  sumatorio(col, 1) + (hipotesis((X(fila, :))', theta) -
y(fila)) .* X(fila, col);
      if col >= 1
        sumatorio(col, 1) = sumatorio(col, 1) + ((lambda/numFils) * theta(col, 1));
      endif
    endfor
  endfor

  for fila = 1:numFils
    suma = suma + ((-y(fila)*log(hipotesis(X(fila, :)', theta))) - ((1 - y(fila))*log(1
- hipotesis(X(fila, :)', theta))));
  endfor

  for fila = 1:rows(theta)
    suma2 = suma2 + (theta(fila, 1)^2);
  endfor

  grad = (1/numFils) * sumatorio;

  J = ((1/numFils) * suma) + ((lambda/(2 * numFils)) * suma2);

Endfunction
```

Figura 14a

Tras un aviso del profesor y detectar fallo en el funcionamiento de la anterior función, se ha reimplementado la función de *costeRegularizado* de manera vectorizada prestando atención en la distinción para el cálculo del gradiente cuando j = 0 y j >= 1. Así, la función queda tal y como se puede observar en la figura 14b (para los cálculos y las gráficas posteriormente mostradas se ha utilizado la versión vectorizada de la función de coste):

```
function [J, grad] = costeRegularizado(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 14b

```
>> y = ex2data2(:,3)
>> theta_inicial = zeros(28,1);
>> [J, grad] = costeRegularizado(theta_inicial, out, y, 0)
J =  0.69315
grad =

  8.4746e-03
  1.8788e-02
  7.7771e-05
  5.0345e-02
  1.1501e-02
  … (y otros 23 más)
```

Figura 15

De la misma manera que hicimos en el apartado 2.1, ahora volvemos a utilizar la función *fminunc* adaptando la función que la envolvía, llamándose ahora *valorOptimoParte2*, que viene implementada en la figura 16:

```
function [theta,cost] = valorOptimoParte2(X, y, theta_inicial, lambda)
  opciones = optimset('GradObj', 'on', 'MaxIter', 400);
  [theta, cost] = fminunc(@(t)(costeRegularizado(t, X, y, lambda)), theta_inicial,
opciones);
endfunction
```

Figura 16

Ejecutamos esta última función con diferentes valores del factor de aprendizaje lamba y obtenemos el resultado que podemos observar en las siguientes figuras. En la figura 17 aplicamos un lamba igual a 0, obteniendo la gráfica de la figura 18 y consiguiendo el porcentaje de aciertos que se observa en la figura 19. En la figura 20 aplicamos un lamba igual a 1, obteniendo la gráfica de la figura 21 y consiguiendo el porcentaje de aciertos de la figura 22. Y por último en la figura 23 hemos aplicado un

lambda igual a 7 (aplicamos un poco de separación), obteniendo la gráfica de la figura 24 y consiguiendo el porcentaje de aciertos de la figura 25.

```
>> [theta,cost] = valorOptimoParte2(out, y, theta_inicial, 0)
theta =

1.79571
 -2.88733
1.12066
  -1.54554
  -9.71673
  … (y 23 más)

cost =  0.28371
>> plotDecisionBoundary(theta, out, y)
```

Figura 17

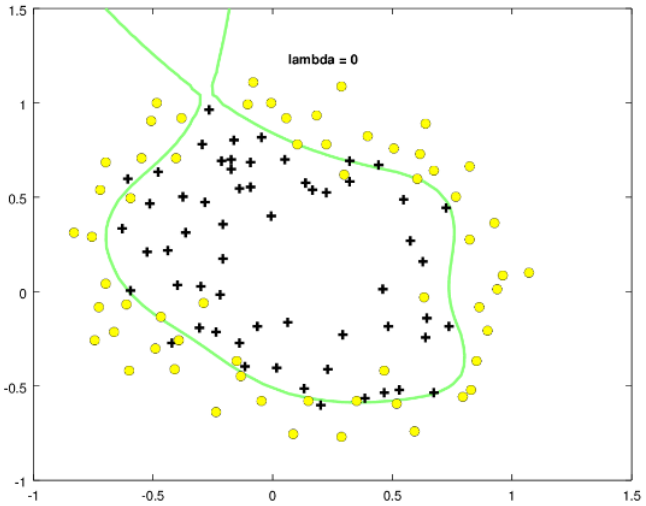


Figura 18

```
>> p = obtengoPorcentaje(out, y, theta)
p =  86.441
```

Figura 19

```
>> [theta,cost] = valorOptimoParte2(out, y, theta_inicial, 1)
theta =

1.1974178
0.6480685
1.1569607
-1.9641830
-0.8685547… (y 23 más)

cost =  0.53553
>> plotDecisionBoundary(theta, out, y)
```

Figura 20

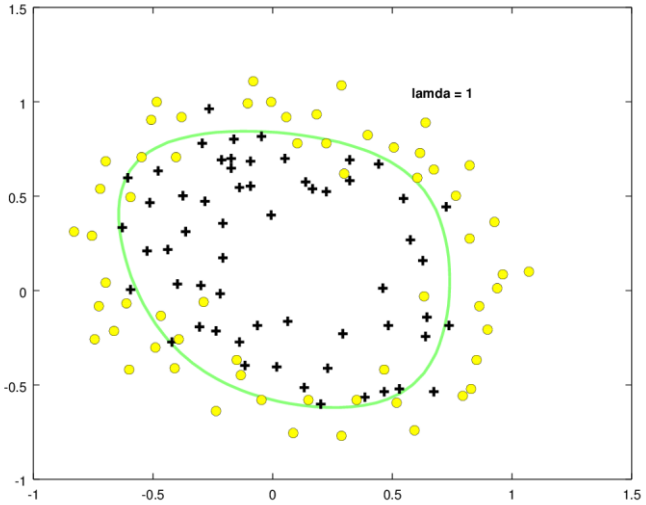


Figura 21

```
>> p = obtengoPorcentaje(out, y, theta)
p =  83.898
```

Figura 22

```
>> [theta,cost] = valorOptimoParte2(out, y, theta_inicial, 7)
theta =

3.8450e-01
 -9.8229e-03
2.2742e-01
-5.3144e-01
-1.4360e-01

cost =  0.64054
>> plotDecisionBoundary(theta, out, y)
```

Figura 23

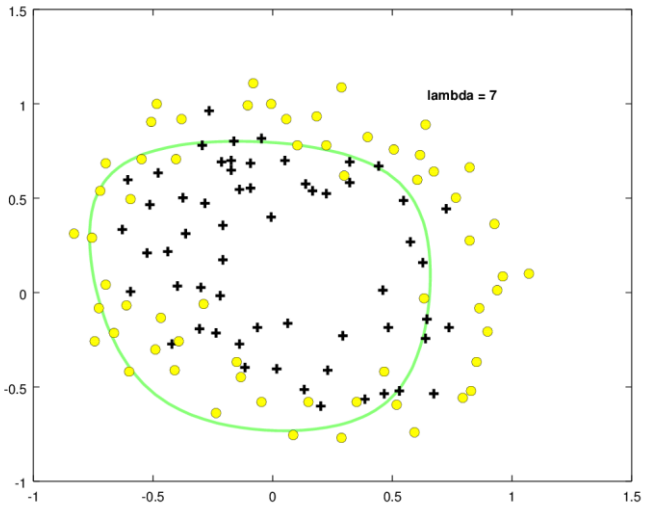


Figura 24

```
>> p = obtengoPorcentaje(out, y, theta)
p =  72.881
```

Figura 25

A partir de estos casos de ejemplo, podemos observar que cuando damos un valor de lambda superior, el porcentaje de aciertos se ve reducido.