Práctica 2:
Regresión logística

Grupo 13:
David Ortiz Fernández.
Andrés Ortiz Loaiza.

Parte 1: Regresión logística.

En esta primera parte de la práctica aplicaremos regresión logística a un conjunto de datos.

Tras cargar dichos datos desarrollaremos el algoritmo que genere un modelo que genere una frontera de separación lineal.

Se desarrollará también una función sigmoide que se utilizará en el cálculo del coste y para la aplicación del descenso de gradiente. Se utilizará la función fminc para la aplicación del descenso de gradiente.

Finalmente, tras visualizar el modelo obtenido que minimiza el coste, se procederá a mostrar el porcentaje de aciertos.

Código:

Función sigmoide.

```
%1.2. Función sigmoide
function g = fsigmoide(z)
%la función se puede aplicar a une escalar, matriz o vector
g = 1./ (1 + e.^-z);

endfunction
```

Función de coste y gradiente.

```
function [J, grad] = coste(theta, X, y)
m = length(y); %numero de ejemplos de entrenamiento

J = 0;
grad = zeros(size(theta));

h = fsigmoide(X*theta); %calculo de h con la función sigmoide

J = (1/m)*(-y'* log(h) - (1 - y)'* log(1-h));
grad = (1/m)*X'*(h - y);

endfunction
```

Función para calcular el porcentaje de aciertos.

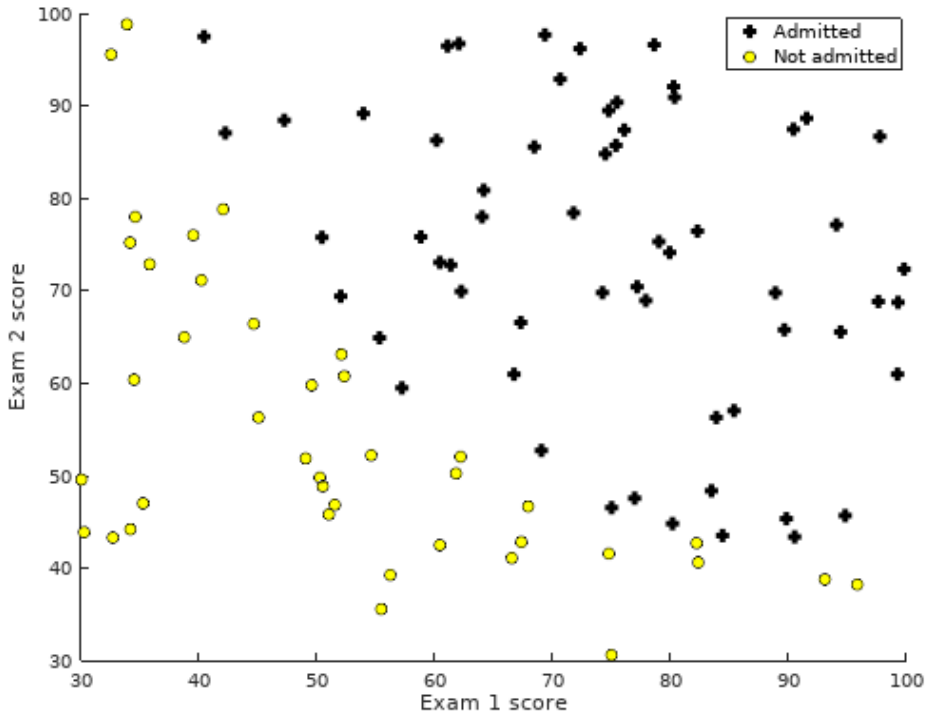
```
function p = porcentaje(theta, X, y)
  m = size(X, 1); % numero de ejemplos de entrenamiento
  p = zeros(m, 1);

  p = (fsigmoide(X*theta) >= 0.5);

  printf('Porcentaje de aciertos: %f\n', mean(double(p == y)) * 100);
endfunction
```

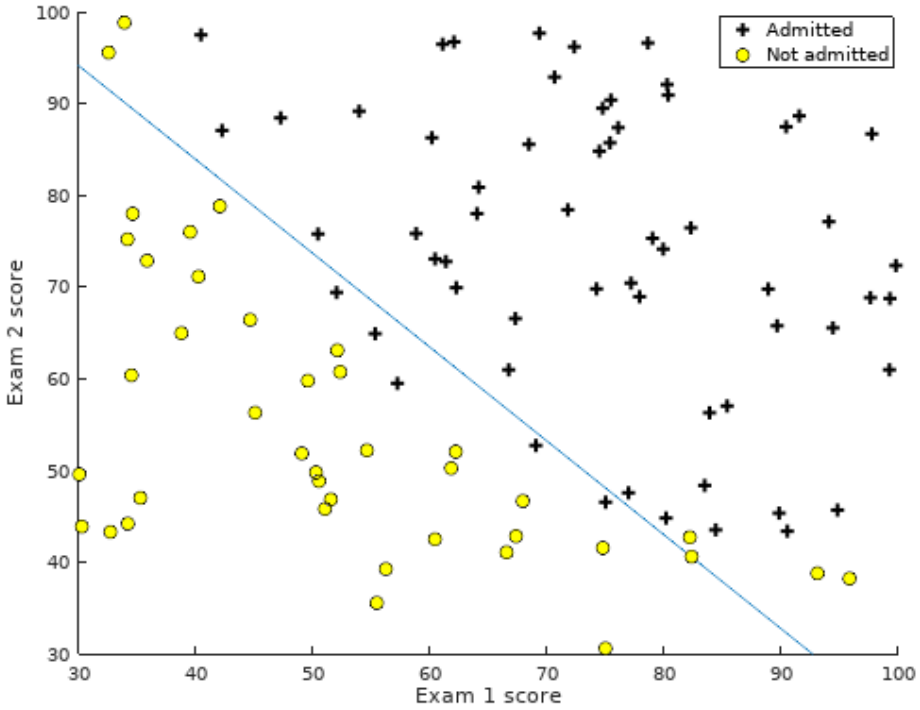
Flujo de datos principal

```
clear all;
close all;
%1.1. Visualización de los datos
%cargamos los datos
data = load('ex2data1.txt');
X = data(:, [1, 2]);
y = data(:, 3);

%Pintamos los datos
figure;
hold on;
positivos =find (y==1);
negativos = find(y==0);

plot(X(positivos, 1), X(positivos, 2), 'k+','LineWidth', 3, 'MarkerSize', 7)
plot(X(negativos,1), X(negativos, 2),'ko','MarkerFaceColor','y')
xlabel('Exam 1 score')
ylabel('Exam 2 score')
legend('Admitted', 'Not admitted')
hold off;

%1.3. Cálculo de la función de coste y su gradiente
[m, n] = size(X);
X = [ones(m, 1) X]; %como en regresión lineal añadimos 1

% Inicializamos los valores de theta a 0
thetainicial = zeros(n + 1, 1);

% Coste inicial y gradiente
[cost, grad] = coste(thetainicial, X, y);
```

```
printf('Coste para valores theta iniciales: %f\n', cost);
printf('Presione Enter para continuar.\n');
pause;


%1.4. Cálculo del valor óptimo de los parámetros con fminunc
opciones = optimset('GradObj', 'on', 'MaxIter', 400);
% Obtención d el valor óptimo de theta
[theta, cost] = fminunc(@(t)(coste(t, X, y)), thetainicial, opciones);

printf('Coste para valores theta calculados con fminunc: %f\n', cost);
printf('Presione Enter para continuar.\n');
pause;

%para obtener la gráfica utilizamos la función plotDecisionBoundary
plotDecisionBoundary(theta, X, y);
hold on;
xlabel('Exam 1 score')
ylabel('Exam 2 score')
legend('Admitted', 'Not admitted')
hold off;

%1.5. Evaluación de la regresión logística
p = porcentaje(theta, X, y);
```

Resultados :





Coste para valores theta iniciales: 0.693147.
Coste para valores theta calculados con fminunc: 0.203498
Porcentaje de aciertos: 89.00 %.

A pesar de utilizar un modelo lineal, se ha obtenido un alto porcentaje de aciertos.

2. Regresión logística regularizada.

En esta segunda parte implementaremos la regresión logística regularizada. Tras cargar los datos invocaremos a la función mapFeature que se encargará de generar términos polinómicos de la matriz de features X.
Por último, como en la parte anterior se minimizará el coste aplicando para ello el descenso de gradiente, obteniendo así los valores óptimos de theta asociados al menos coste.
Código:

2.1 Cálculo de la función de coste y su gradiente.

```
function [J, grad] = costeReg(theta, X, y, lambda)
m = length(y);
J = 0;
grad = zeros(size(theta));

h = fsigmoide(X*theta);

stheta = theta(2:size(theta));
thetaReg = [0;stheta];

J = (1/m)*(-y'* log(h) - (1 - y)'*log(1-h))+(lambda/(2*m))*thetaReg'*thetaReg;

grad = (1/m)*(X'*(h-y)+lambda*thetaReg);

endfunction
```

2.2 Regresión

```
clear all;
close all;
%Visualización de los datos
%cargamos los datos
data = load('ex2data2.txt');
X = data(:, [1, 2]);
y = data(:, 3);

%Pintamos los datos
figure;
hold on;
positivos =find (y==1);
negativos = find(y==0);

plot(X(positivos, 1), X(positivos, 2), 'k+','LineWidth', 3, 'MarkerSize', 7)
plot(X(negativos,1), X(negativos, 2),'ko','MarkerFaceColor','y')
xlabel('Microchip Test 1')
ylabel('Microchip Test 2')
legend('y = 1', 'y = 0')
hold off;

%2.1. Mapeo de los atributos
X = mapFeature(X(:,1), X(:,2));

%2.2. Cálculo de la función de coste y su gradiente
thetainicial = zeros(size(X, 2), 1);
lambda = 1;

[cost, grad] = costeReg(thetainicial, X, y, lambda);
printf('Coste para valores theta iniciales: %f\n', cost);
printf('Presione Enter para continuar.\n');
pause;

%2.3. y 2.4. Cálculo del valor óptimo de los parámetros con fminunc
%lambda = 0;
%lambda = 1;
lambda = 2;
%lambda = 3;
%lambda = 10;
%lambda = 20
%lambda = 50;
opciones = optimset('GradObj', 'on', 'MaxIter', 400);
[theta, J] = fminunc(@(t)(costeReg(t, X, y, lambda)), thetainicial, opciones);
```

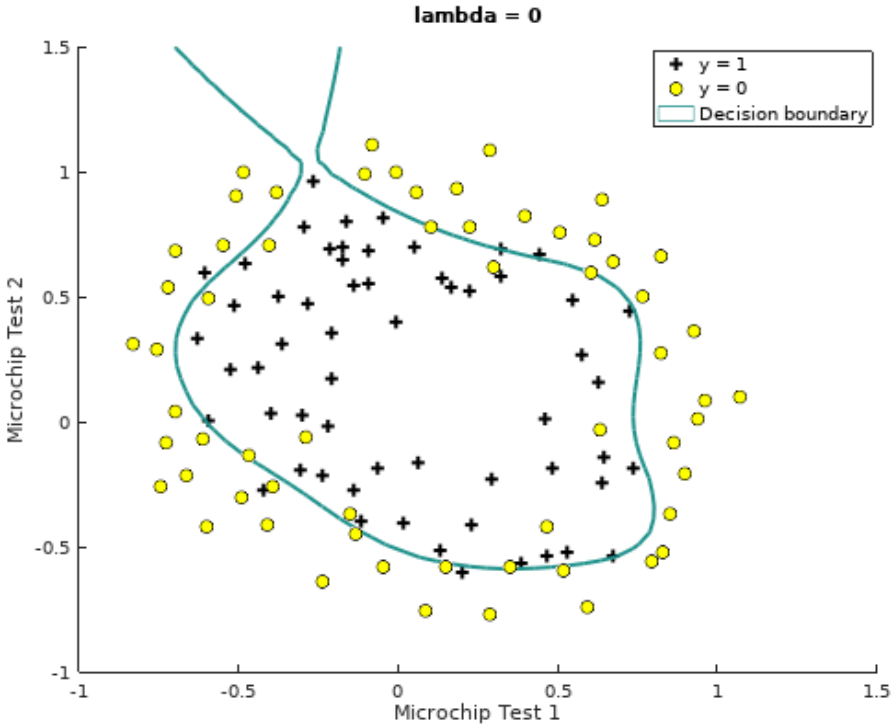
```
printf('Coste para valores theta calculados con fminunc: %f\n', cost);
printf('Presione Enter para continuar.\n');
pause;


plotDecisionBoundary(theta, X, y);
title(sprintf('lambda = %g', lambda))
xlabel('Microchip Test 1')
ylabel('Microchip Test 2')
legend('y = 1', 'y = 0', 'Decision boundary')
hold off;

p = porcentaje(theta, X, y);
```

2.3 Resultados :

Tras aplicar descenso de gradiente y minimizar el coste, calculando para ello las thetas óptimas, se muestran los siguientes valores:

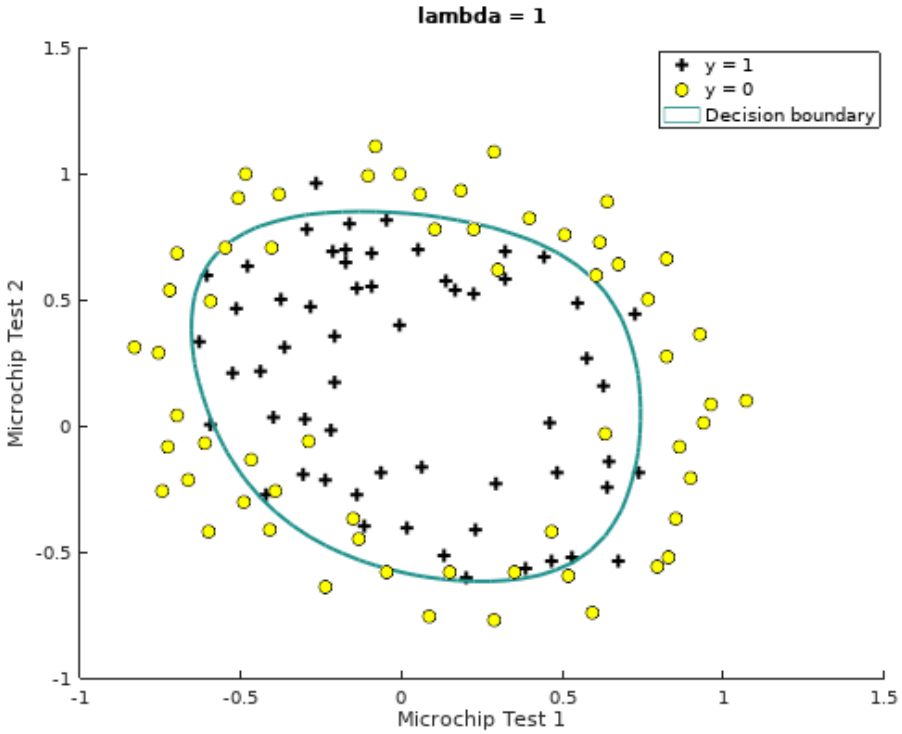
-Coste para valores iniciales de theta : 0.693147
-Coste para valores theta óptimos con fminunc :0.693147

Lambda = 0 .

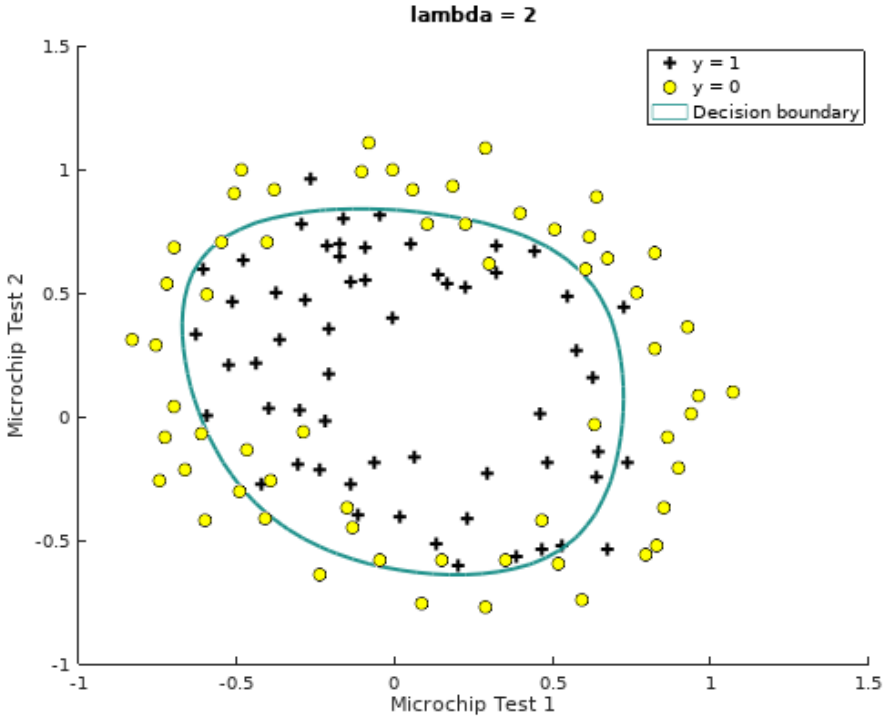


Para este valor se produce overiftting en los datos de entrenamiento como podemos observar en la gráfica.

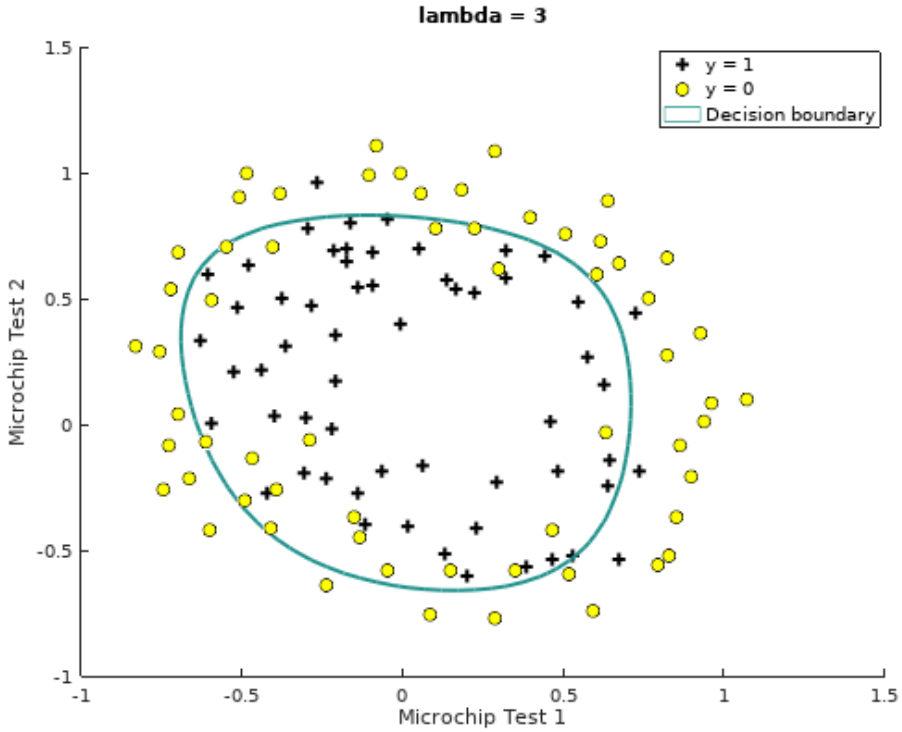
Lambda = 1.



Lambda = 2 .

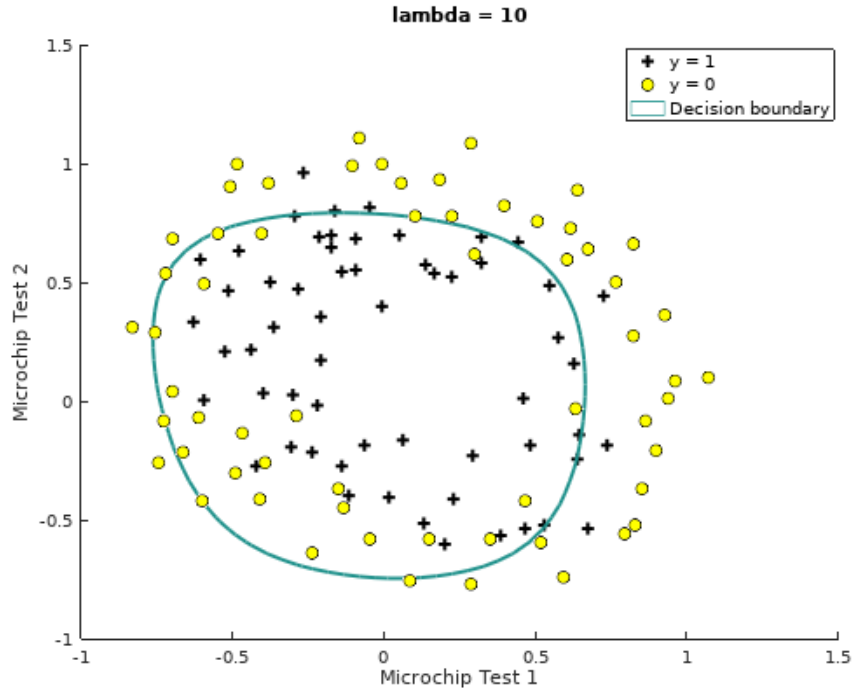


Lambda = 3 .



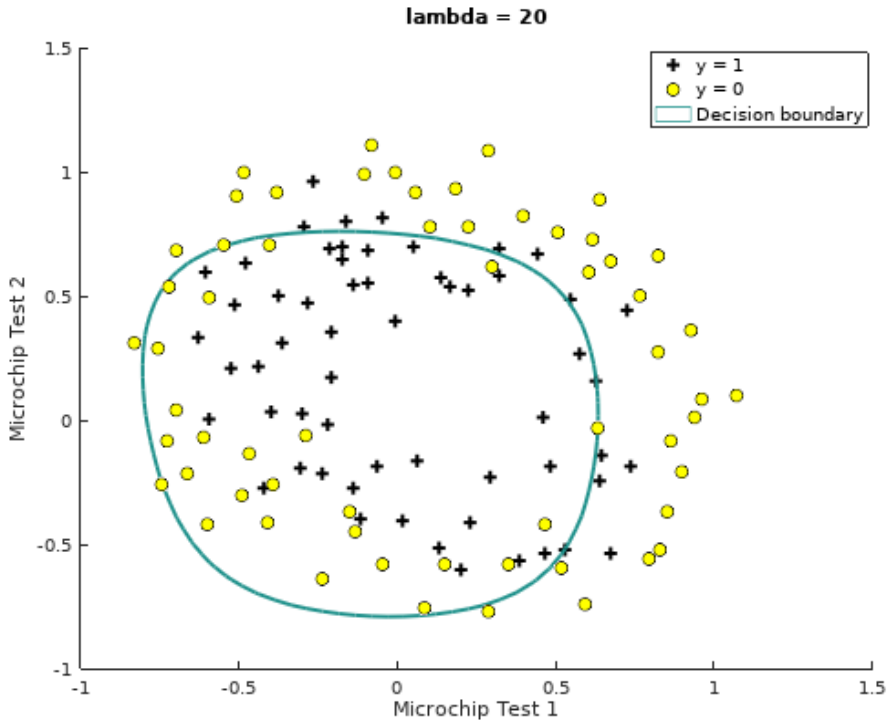
Para lambda = (1 , 2 , 3) obtenemos un modelo que reconoce bien los datos de entrenamiento sin llegar al sobreentrenamiento (overfitting).
Aunque los mejores resultados se dan con lambda = 1 y lambda = 2 con un porcentaje de aciertos de: 83.050847 , mientras que con lambda = 3 bajamos a un 80.508475 %. Estos valore de lambda proporcionan buenos modelos.

Lambda = 10.



Porcentaje de aciertos: 74.576271 , un valor bajo en comparación con lambdas anteriores.

Lambda = 20.



Porcentaje de aciertos: 68.644068, como se puede observar, es un ejemplo de underfitting, al aumentar lambda tanto.

Observamos que para valores altos de lambda el modelo no logra reconocer todos los chips averiados que debería, produce bastantes falsos negativos.
El porcentaje de aciertos desciende bastante.