Aprendizaje automático

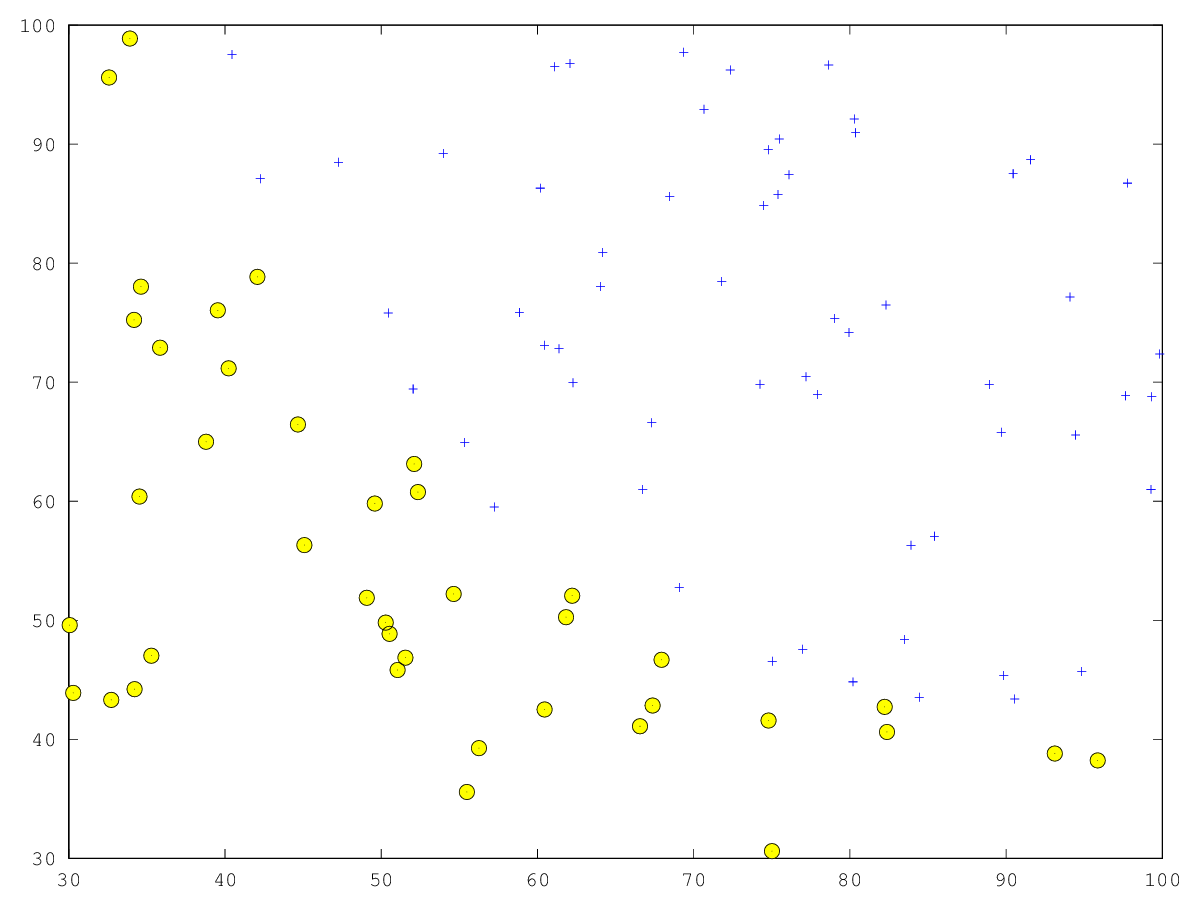
Práctica 2

Miguel Ascanio Gómez

Esther Ávila Benito

# Parte 1

## Visualización de los datos



## Función sigmoide

function s = sigmoide(z)

zNeg = zeros(length(z(:,1)), length(z(1,:))) .- z;

s = 1./(1+e.^(zNeg));

endfunction

## Función de coste y gradiente

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

warning("off", "Octave:broadcast");

m = length(X(:,1));

valoresH = (theta' \* X')';

h = sigmoide(valoresH);

a = (0.-Y) .\* log(h) .- ((1 .- Y) .\* log(1 .- h));

J = (1/m) \* sum(a);

a = (h .- Y) .\* X;

grad = 1/m \* sum(a);

warning("on", "Octave:broadcast");

endfunction

## Cálculo y representación

# Lectura de datos

s = load("ex2data1.txt");

X = s(:,1:2);

y = s(:,3);

m = length(X(:,1));

X = [ones(m,1), X];

# Inicialización de theta

thetaIni = zeros(length(X(1,:)),1);

# Cálculo de coste y gradiente

[J, grad] = coste(thetaIni, X, y);

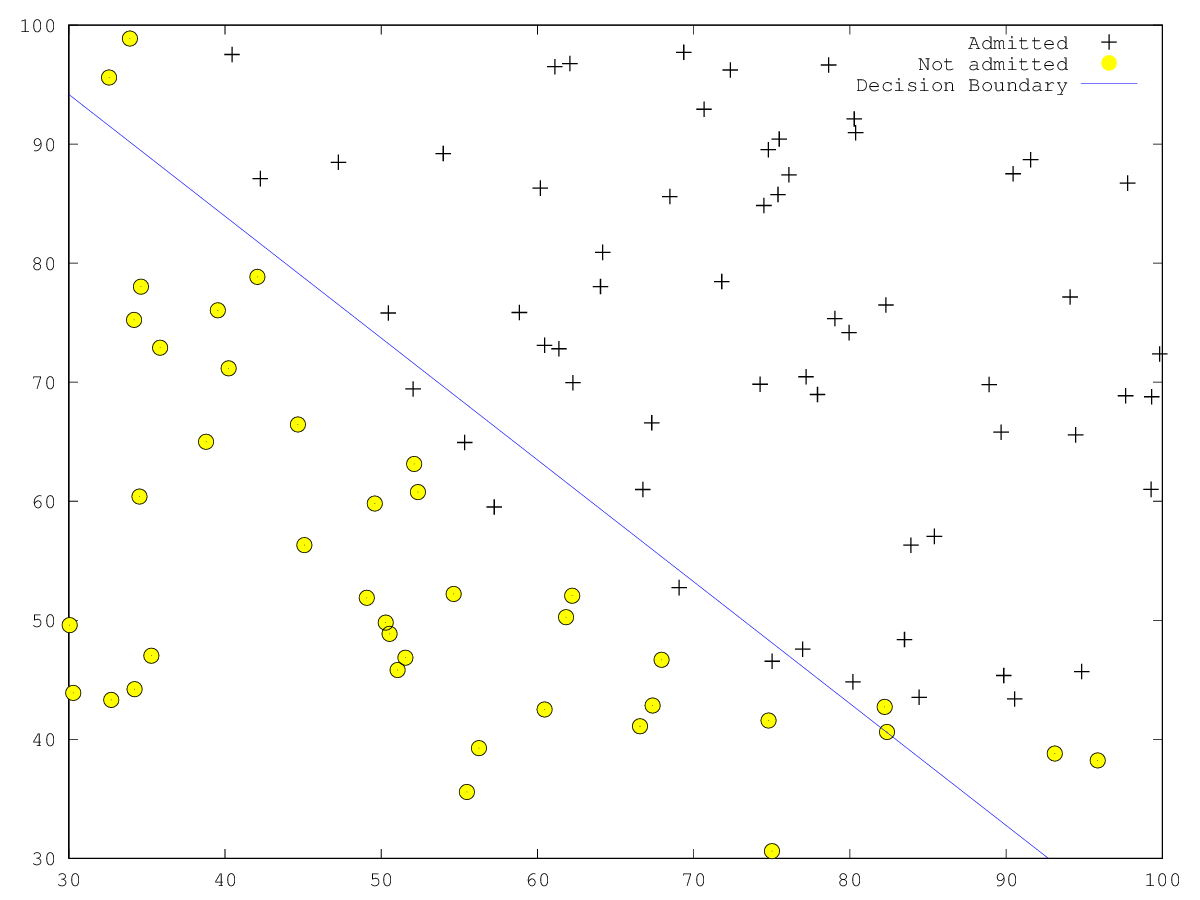
# Representación

opciones = optimset ('Gradobj', 'on', 'MaxIter', 400);

[theta, cost] = fminunc (@(t)(coste(t, X, y)), thetaIni, opciones)

plotDecisionBoundary(theta, X, y);

El coste obtenido es de 0.20350, y la representación es la siguiente:



## EVALUACIÓN LOGÍSTICA

Se ha implementado la siguiente función para evaluar el porcentaje de ejemplos de entrenamiento que utilizando el vector theta quedan correctamente clasificados:

function porcentaje = prueba(theta, X, y)

m = length(X(:,1));

valoresH = (theta' \* X')';

h = sigmoide(valoresH);

h = h >= 0.5;

porcentaje = length(find(h == y)) / m \* 100;

endfunction

Llamando a porcentaje(tehta, X, y) al final del código de la parte Cálculo y representación, se obtiene que el 89% de datos han sido correctamente clasificados, lo que es un valor bastante aceptable, teniendo en cuenta las propias anomalías existentes en los casos de entrenamiento.

# PARTE 2

## FUNCIÓN DE COSTE Y GRADIENTE

function [J, grad] = coste2(theta, X, Y, lambda)

warning("off", "Octave:broadcast");

m = length(X(:,1));

valoresH = (theta' \* X')';

h = sigmoide(valoresH);

# Sumandos

a = (0.-Y) .\* log(h) .- ((1 .- Y) .\* log(1 .- h));

# Theta cuadrado

t = theta .^ 2;

J = (1/m) \* sum(a) + ((lambda / m / 2) \* sum(t));

# Sumandos

a = (h .- Y) .\* X;

# Poner a cero para no incluir el primero

theta(1) = 0;

grad = (1/m \* sum(a)) .- ((lambda/m) .\* theta)';

warning("on", "Octave:broadcast");

endfunction

## CÓDIGO PARA LOS CÁLCULOS Y REPRESENTACIÓN

# Leer datos

s = load("ex2data2.txt");

X = s(:,1:2);

y = s(:,3);

m = length(X(:,1));

# Mapear datos de entrenamiento

out = mapFeature(X(:,1),X(:,2));

# Valores de prueba de lambda

lambdaVals = [100, 50, 25, 10, 5, 3, 1, 0.5, 0.05, 0.005];

resultado = [];

for lambda = lambdaVals

# Resetear valores de theta para cada iteración

thetaIni = zeros(length(out(1,:)),1);

# Calcular coste y gradiente

[J, grad] = coste2(thetaIni, out, y, lambda);

opciones = optimset ('Gradobj', 'on', 'MaxIter', 400);

[theta, cost] = fminunc (@(t)(coste2(t, out, y, lambda)), thetaIni, opciones);

# Rrepresentación gráfica

plotDecisionBoundary(theta, out, y, lambda);

print(["pruebaLambda", num2str(lambda), ".png"], "-dpng");

# Guardar métricas

porcentaje = prueba(theta, out, y);

aux.lambda = lambda;

aux.cost = cost;

aux.porcentaje = porcentaje;

resultado = [resultado, aux];

endfor

# Guardar datos en archivo

for i = 1:length(lambdaVals)

r = resultado(i);

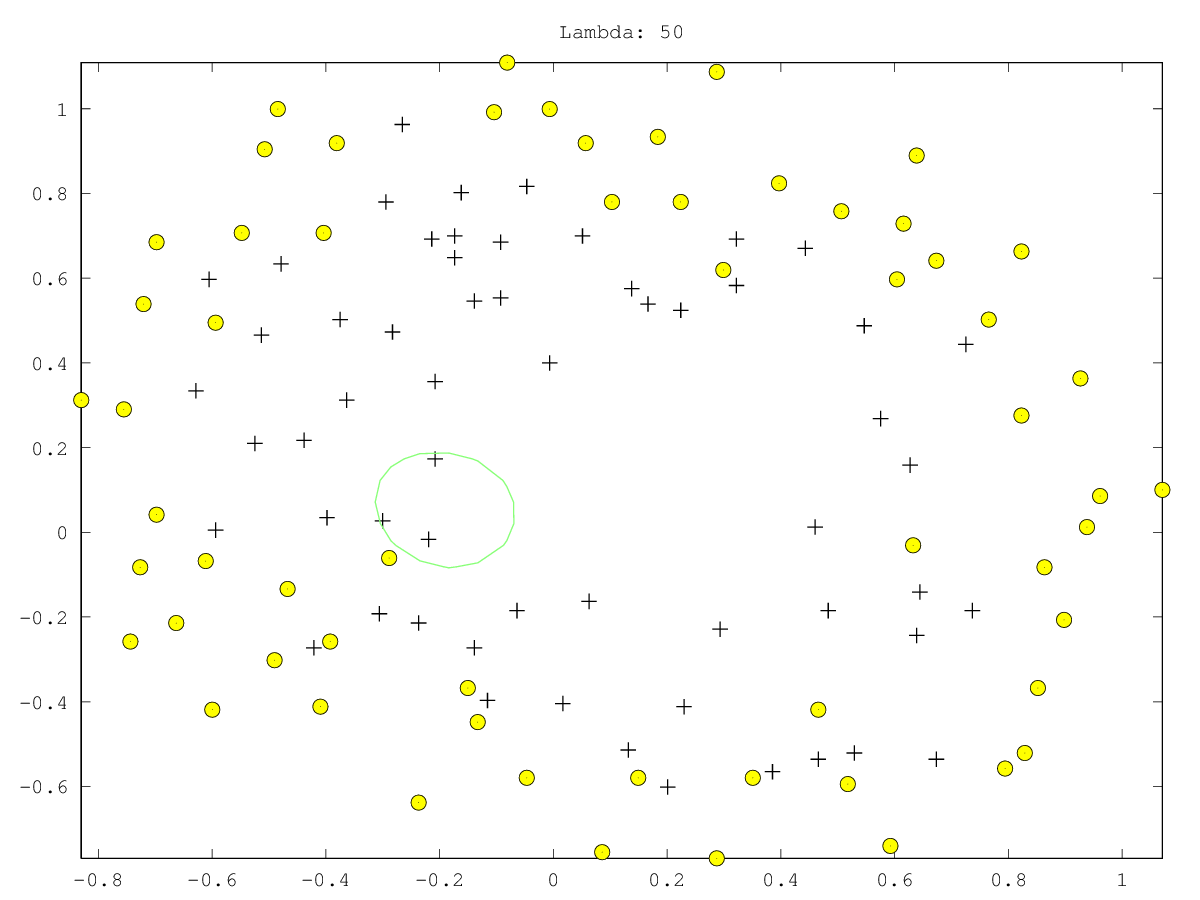
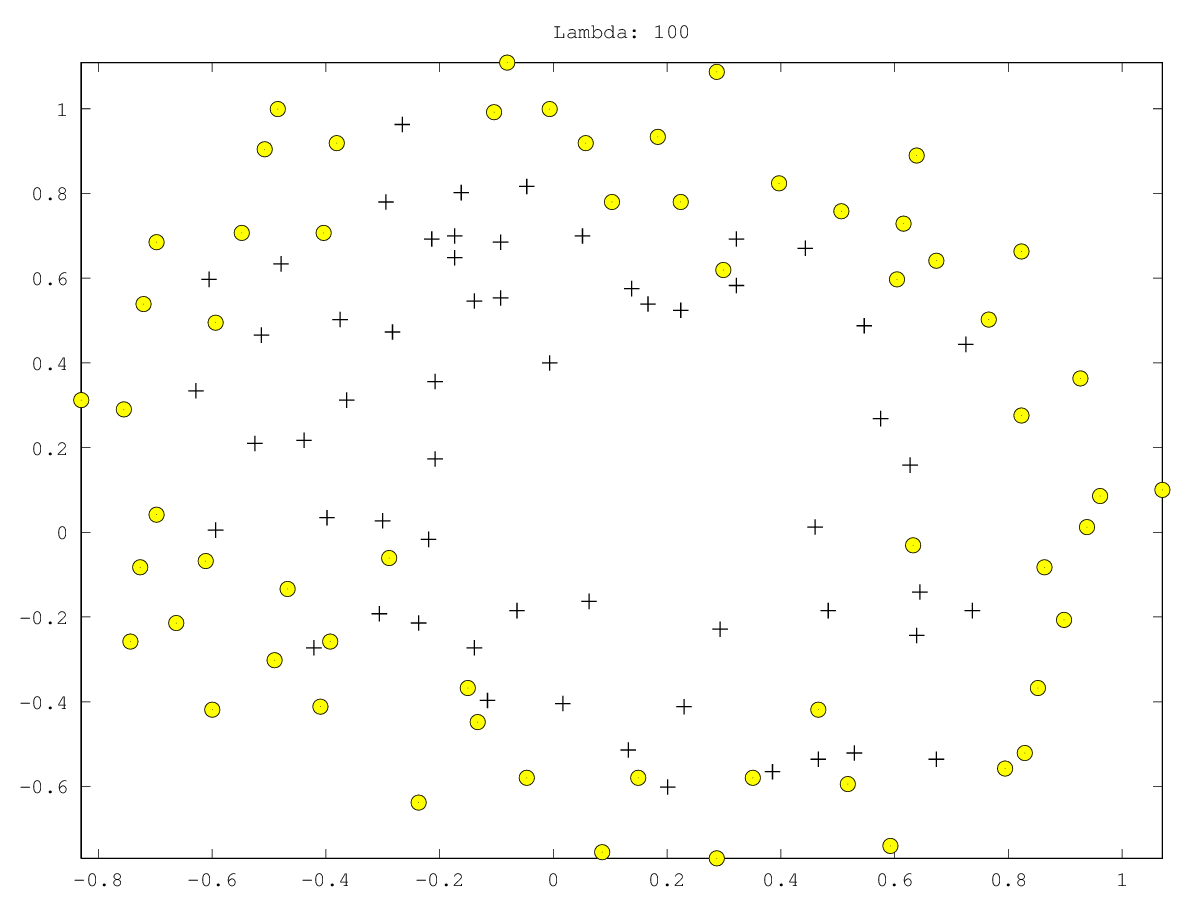
save resultado -append r;

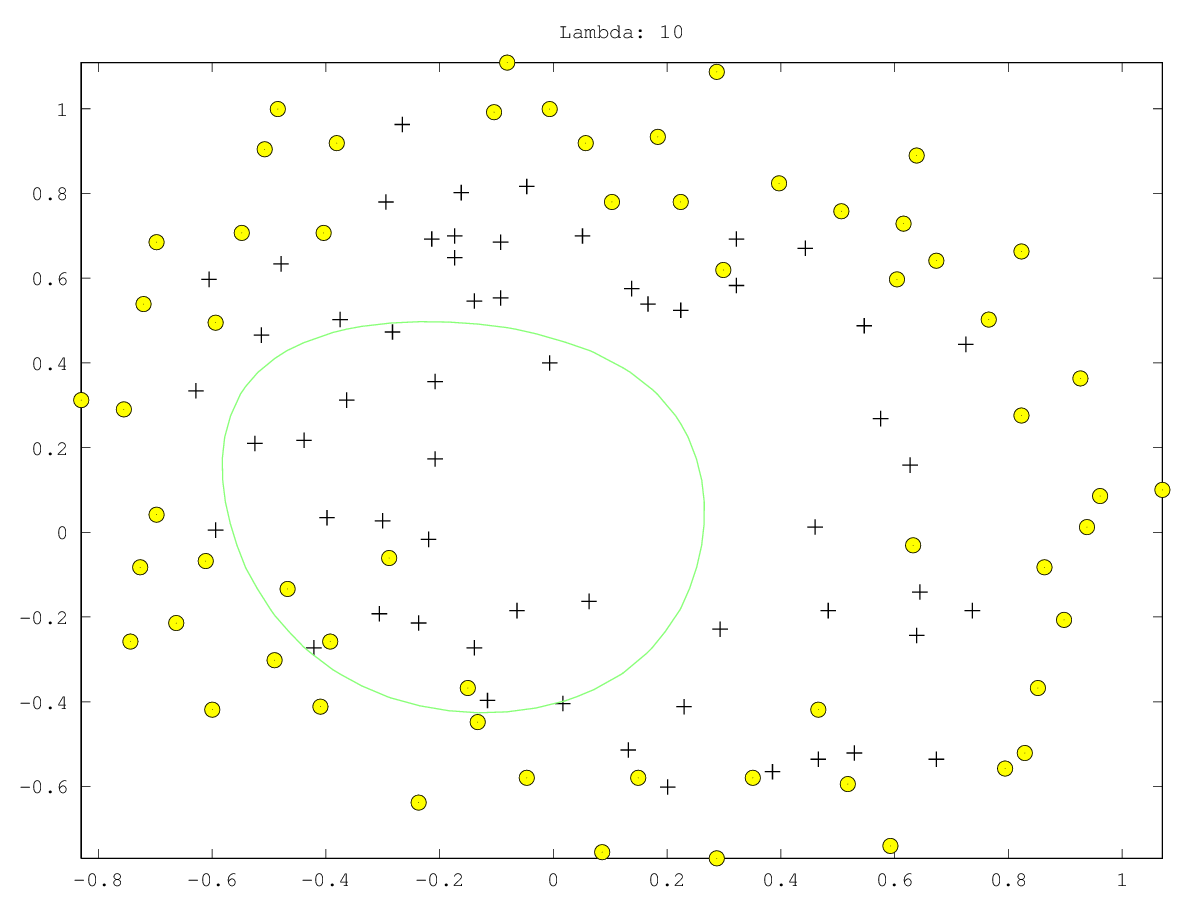
endfor

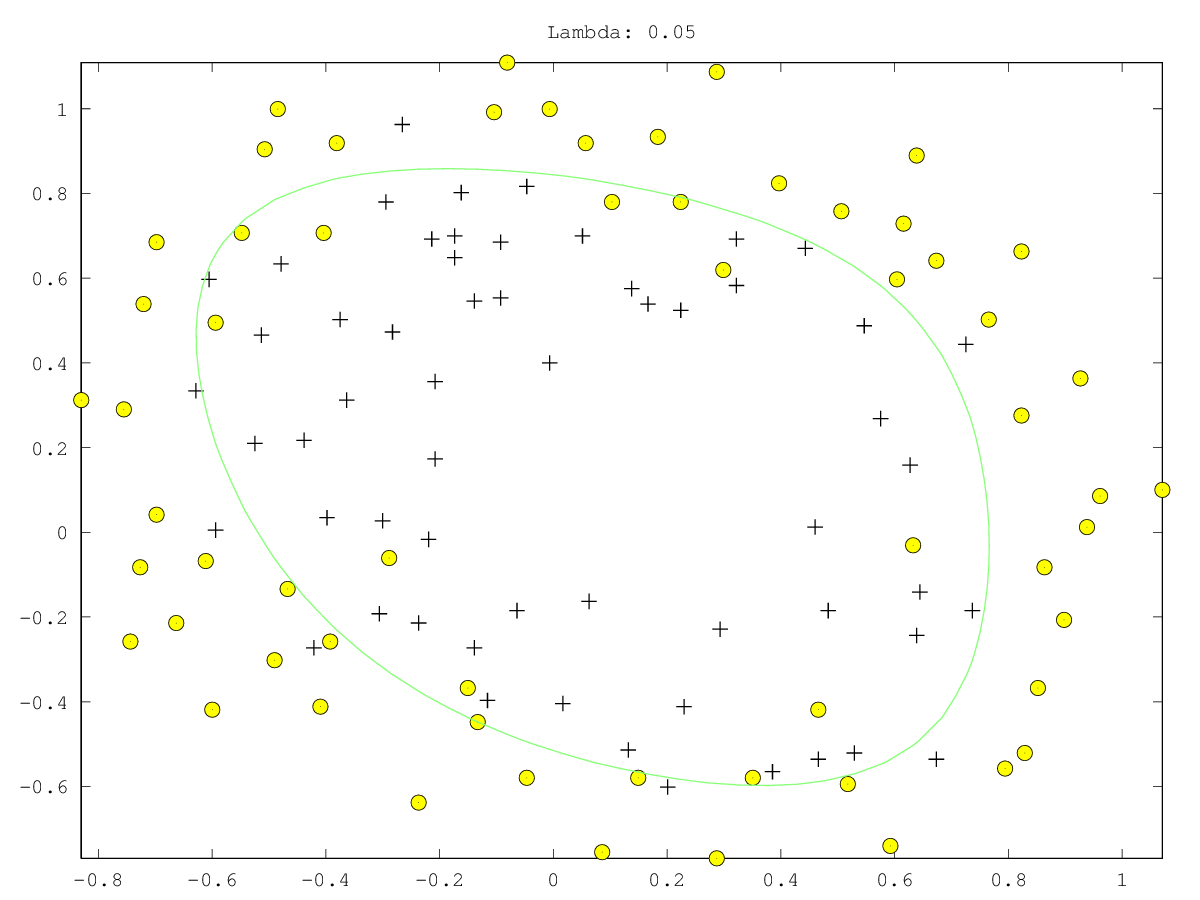
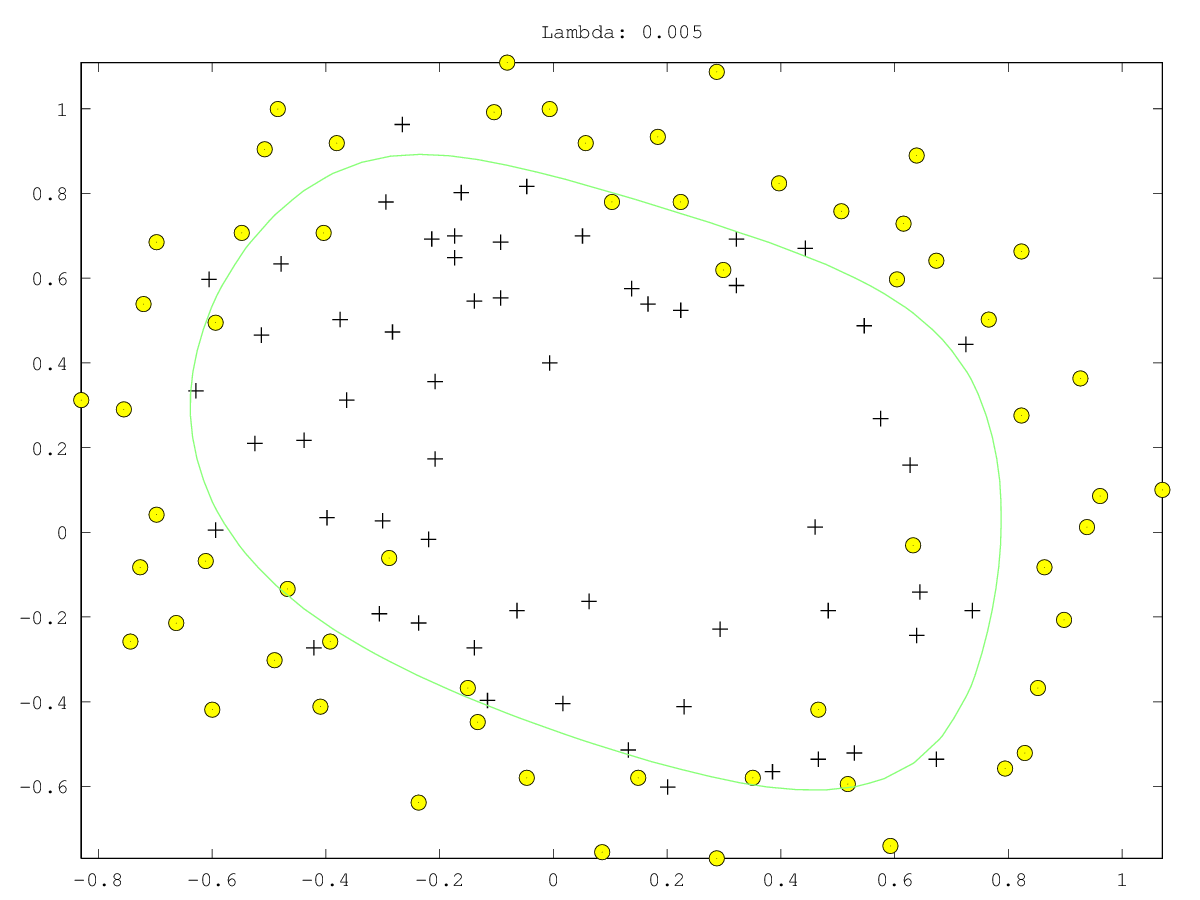
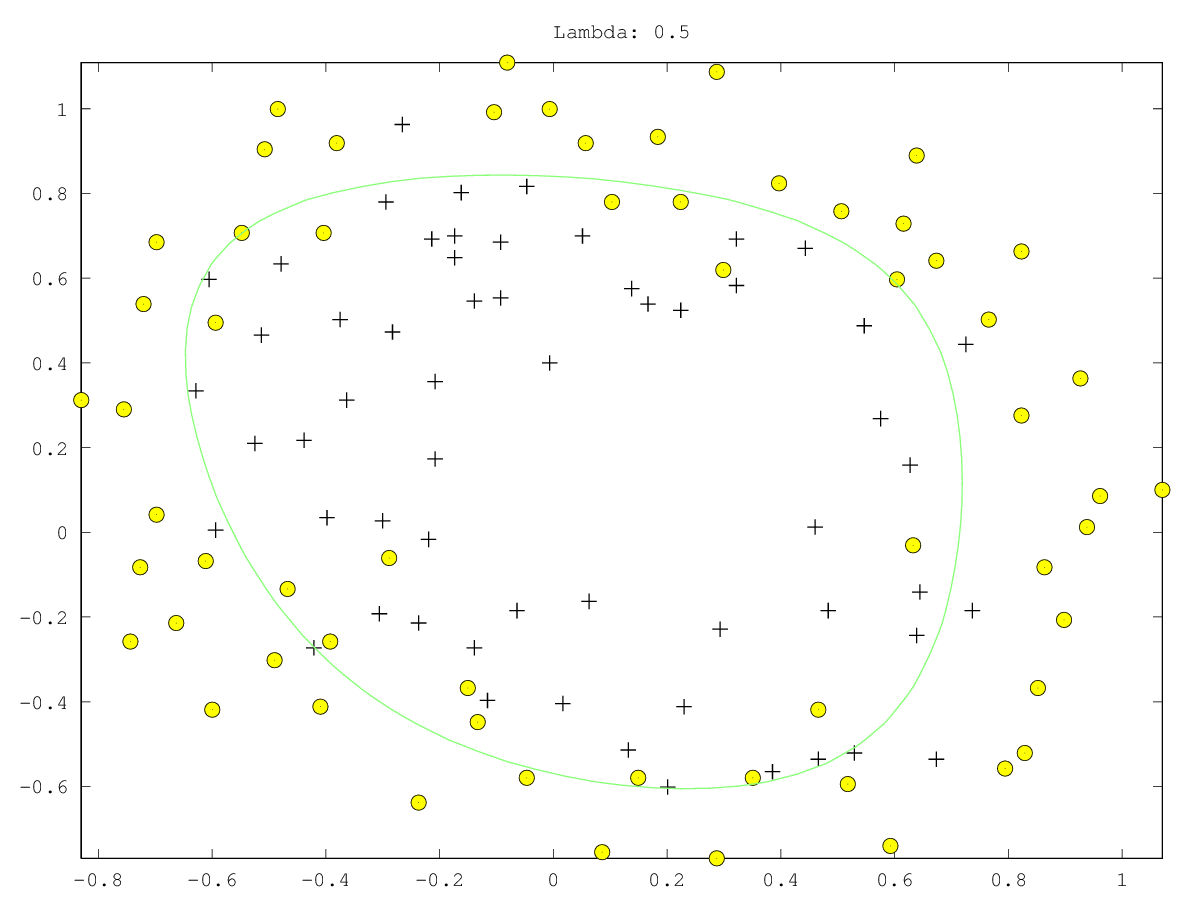
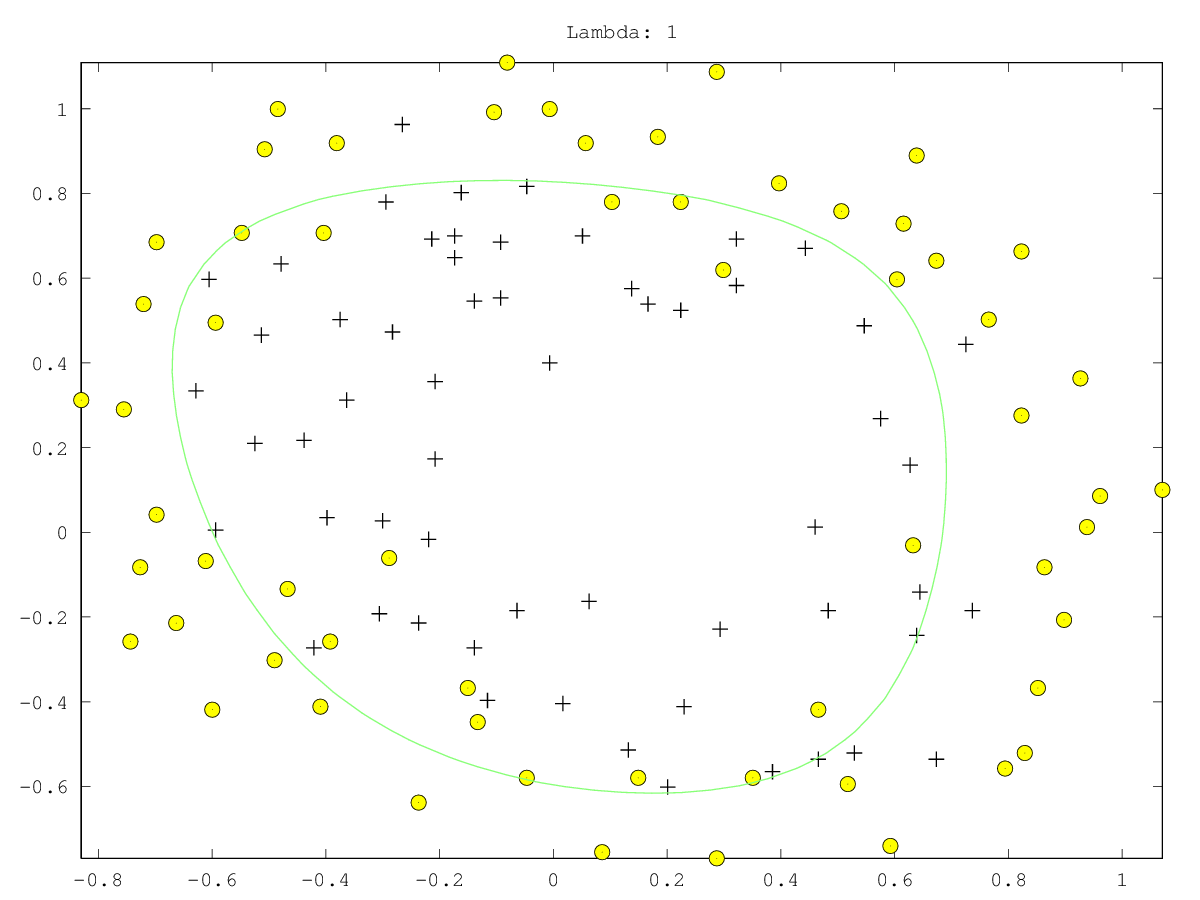
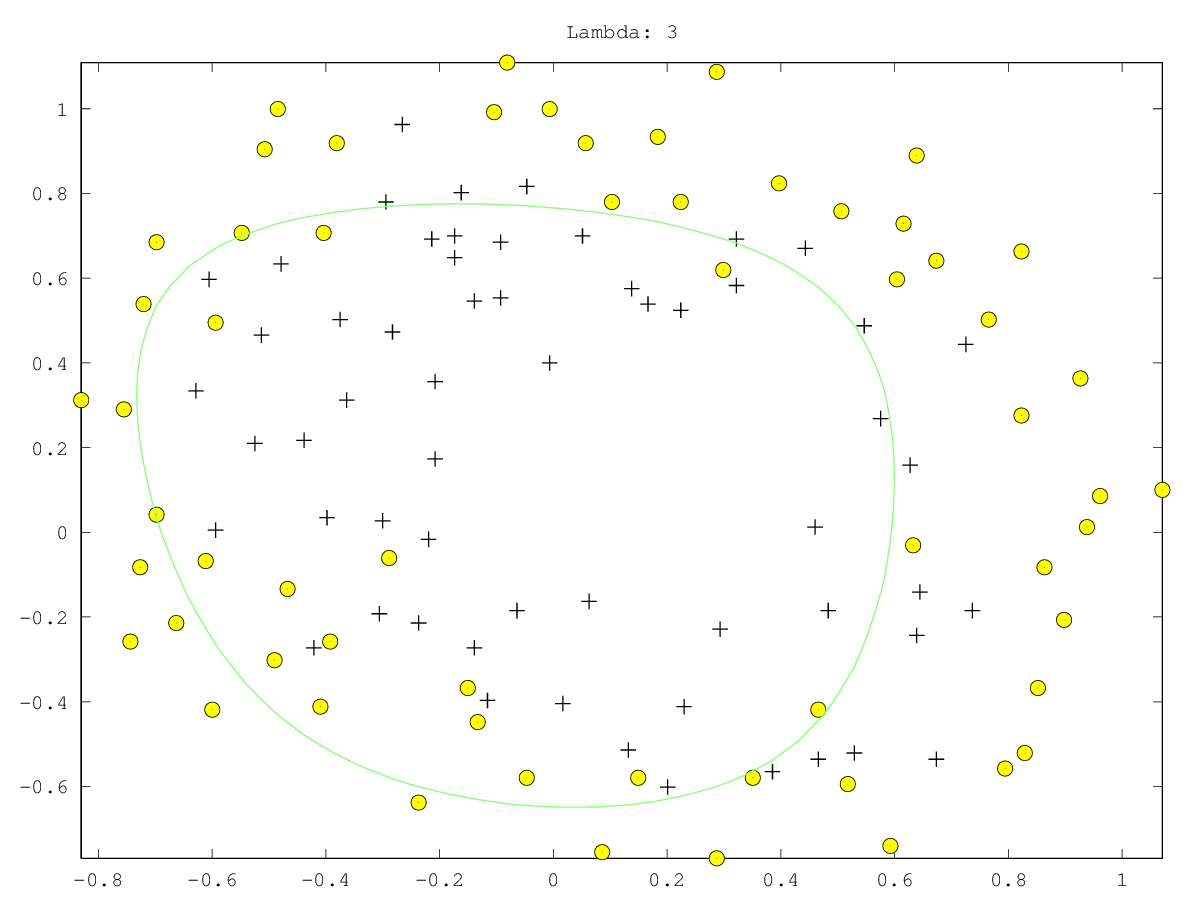
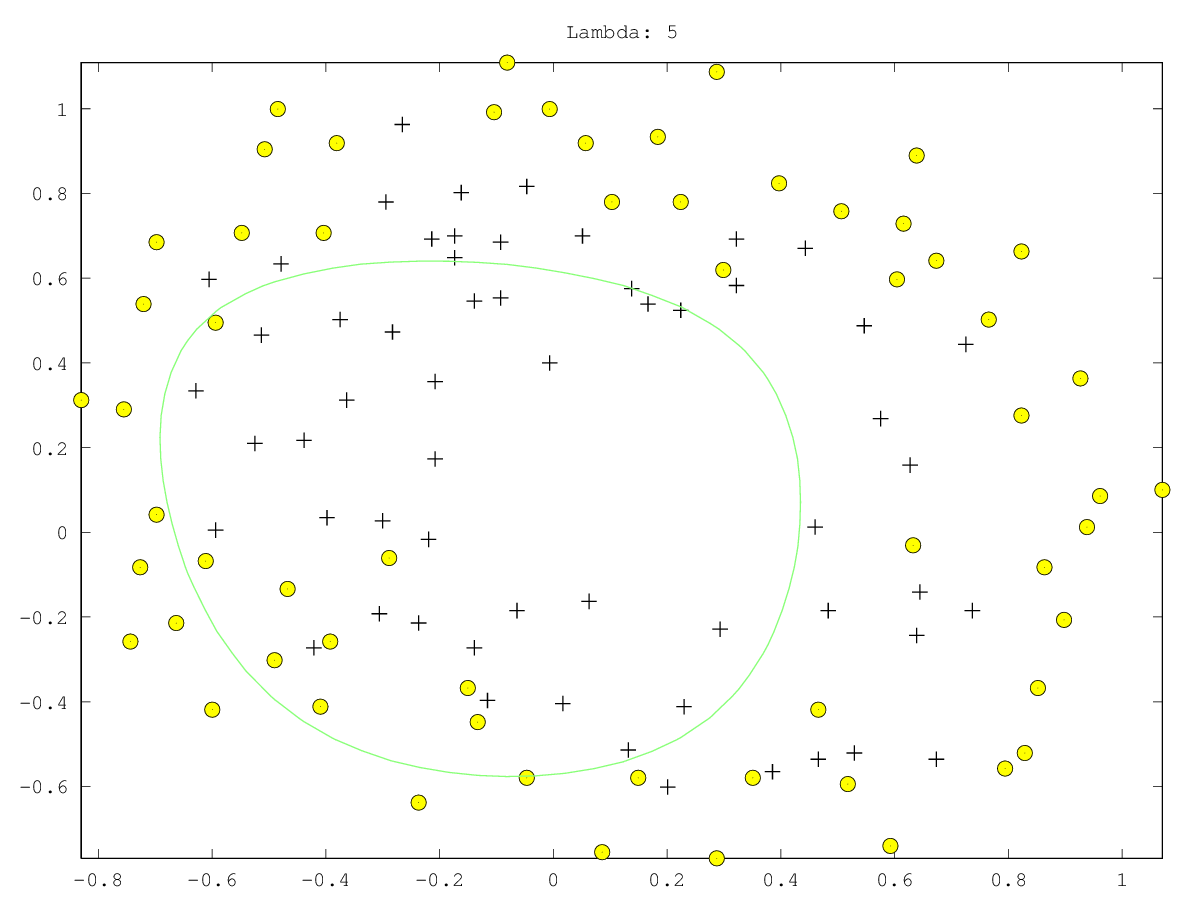
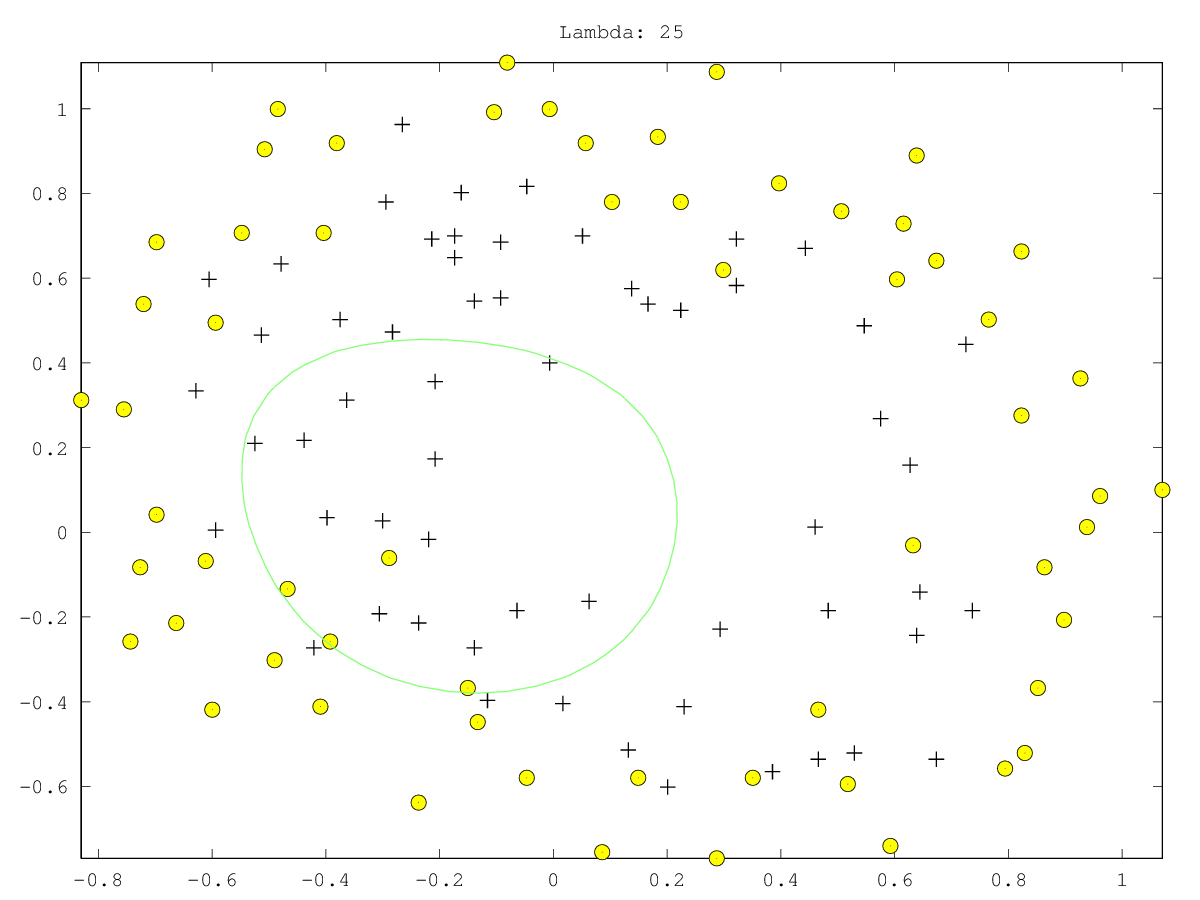
El anterior código calcula, para distintos valores del parámetro lambda, los valores del vector theta que mejor se ajustan a los ejemplos de entrenamiento, según el valor lambda; mientras va guardando la representación gráfica, el coste y el porcentaje de cada valor de lamdba, para luego analizar dichos valores.

Nota: se modificó mínimamente plotDecisionBoundary para que muestre el valor de la lamdba usada en la gráfica

## PRUEBAS REALIZADAS







Con las gráficas anteriores, se observa que para valores pequeños de lambda se obtiene una mejor aproximación. Esto queda también reflejado en la siguiente gráfica, que representa la precisión obtenida respecto de la lambda usada:

