Práctica 5

Pablo Arranz Ropero
Juan Alberto Camino Sáez
Grupo 2

Práctica 5: Regresión lineal regularizada: sesgo y varianza

En esta práctica comprobaremos los efectos del sesgo y la varianza en una regresión lineal, y ver los efectos del overfiting y el underfiting que tiene al elegir distintos valores del parámetro lamba en la regularización y del grado del polinomio, además de usar curvas de aprendizaje para comprobar estos cambios.

En primer lugar ejecutamos la siguiente instrucción para cargar los datos necesarios de esta práctica.

```
1. load("ex5data.mat");
```

Después hemos implementado la función coste en un archivo llamado *coste.m*, que se encarga de calcular el coste y el gradiente de forma vectorizada. Este es el código:

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

  X(:,2:columns(X)+1) = X;
  X(:,1) = ones(rows(X),1);

  valhipotesis = X*theta;
  m = rows(X);

  J = ((1/(2*m))*(valhipotesis-y)' * (valhipotesis-y)) + (lambda/(2*m)) * sum((theta(2:end,:)).^2);

  grad = ((1/m) * (valhipotesis - y)' * X)';

  grad(2:end, :) = grad(2:end, :) + (lambda/m) * theta(2:end,:);

endfunction
```

Al comprobar los datos con Θ = [1; 1], dan los datos correctos según indica la práctica, siendo aproximadamente J = 303.99 y el gradiente [−15,30; 598,250].

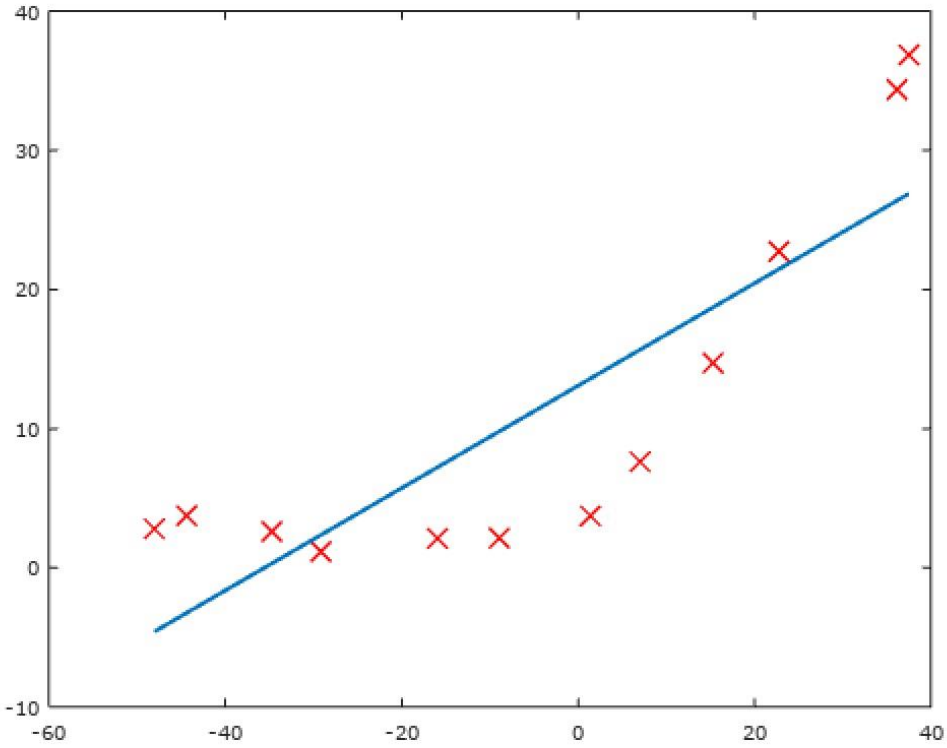
A continuación, llamamos a la función *fmincg* proporcionada por la práctica, donde le pasamos como argumento esta función, y con 200 iteraciones de la siguiente forma:

```
theta = fmincg(@(t)(coste(t,X, y, 0)),[1;1],optimset('GradObj',
'on', 'MaxIter', 200));
```

Al terminar, las thetas resultantes son Θ_0 = 13.087 y Θ_1 = 0.367. Para generar la gráfica, hemos usado el siguiente código:

```
plot(X, y, 'rx', 'MarkerSize', 5, 'LineWidth', 1.5);
hold on;
plot(X, [ones(m, 1) X]*theta, 'LineWidth', 2)
hold off;
```

Y esta es la gráfica resultante:

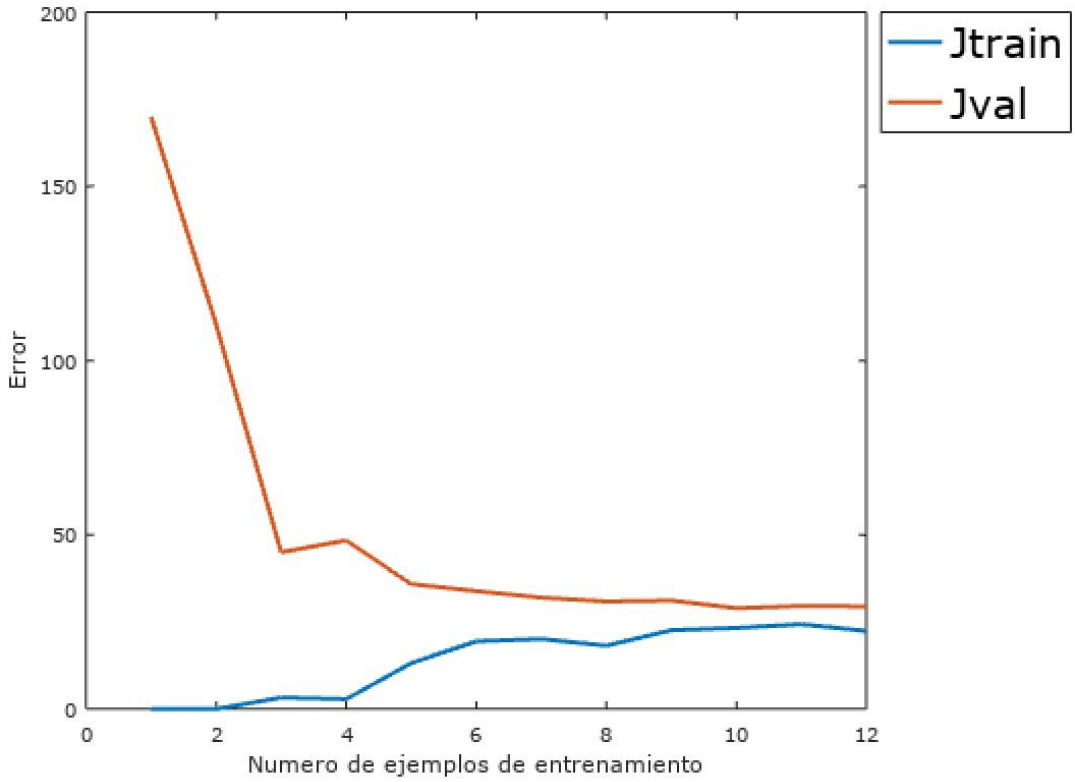


CURVAS DE APRENDIZAJE

A continuación, hemos aplicado curvas de aprendizaje para ver cómo varía el error a medida que vas metiendo más datos de entrenamiento, tanto para los datos de entrenamiento como en los datos del cross validation. Para ello, hemos creado la función generarlc.m, que se encarga de calcular los thetas con el coste más bajo y de calcular este coste, añadiendo cada vez más datos de entrenamiento, dibujando una gráfica donde se ve cómo varian estos costes. Este es el código:

```
function generarlc(initheta, X, y, lambda, Xval, yval)

  for i = 1:rows(X)

    theta = fmincg(@(t)(coste(t,X(1:i,:), y(1:i), lambda)),initheta,optimset('GradObj', 'on', 'MaxIter', 200));
    [Jtrain(i), grad] = coste(theta, X(1:i,:), y(1:i), lambda);

    [Jval(i), grad] = coste(theta, Xval, yval, lambda);

  endfor

  plot([1:1:rows(X)], Jtrain, 'LineWidth', 2);
  xlabel('Numero de ejemplos de entrenamiento')
  ylabel('Error')
  hold on;
  plot([1:1:rows(X)], Jval, 'LineWidth', 2);
  hold off;

  h = legend ({'Jtrain'}, 'Jval');
  legend (h, 'location', 'northeastoutside');
  set (h, 'fontsize', 20);

endfunction
```

Esta es la gráfica obtenida:



REGRESIÓN POLINOMIAL

En este apartado, lo primero que hemos tenido que hacer es una función que, dado una matriz de datos X y un número p, devuelva una matriz del mismo tamaño de X y con p columnas, cada una de ellas elevada a su índice de columna. Esta es la función que hemos creado y utilizado:

```
function X = generatepoldata(X, p)

  for i=2:p
    X(:,i) = X(:,1).^i;
  endfor

endfunction
```

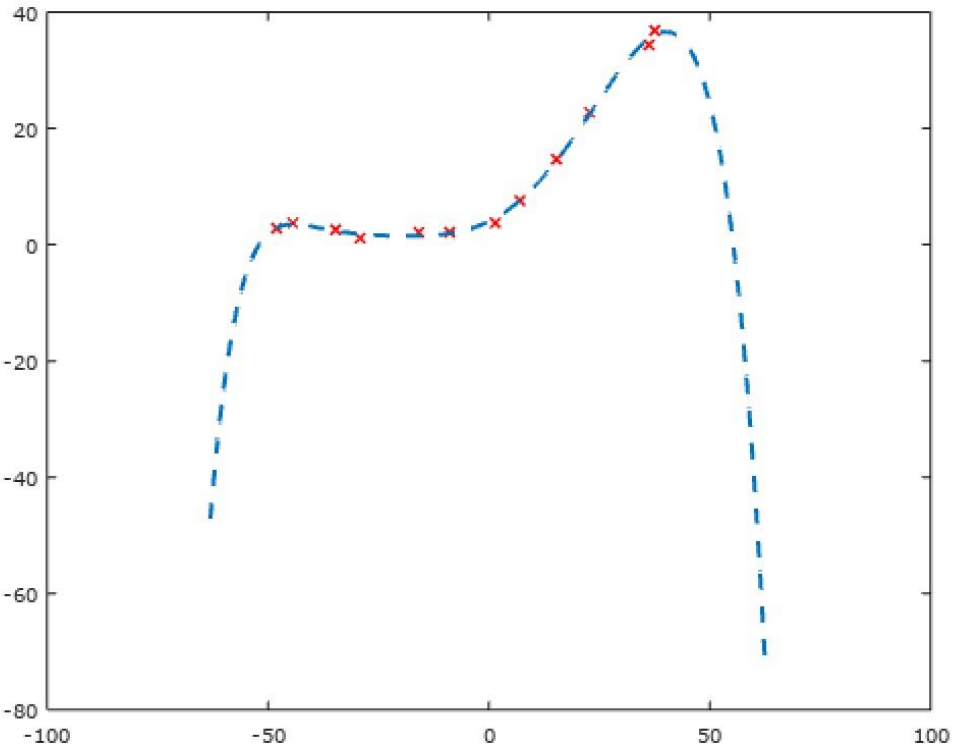
A continuación, creamos una matriz de 8 columnas con los datos de entrenamiento llamando a la función anterior, y además, la normalizamos usando la función proporcionada por la práctica llamada *featureNormalize.m*. Este es el código usado:

```
[newX, media, desv] = featureNormalize(generatepoldata(X,8));
```

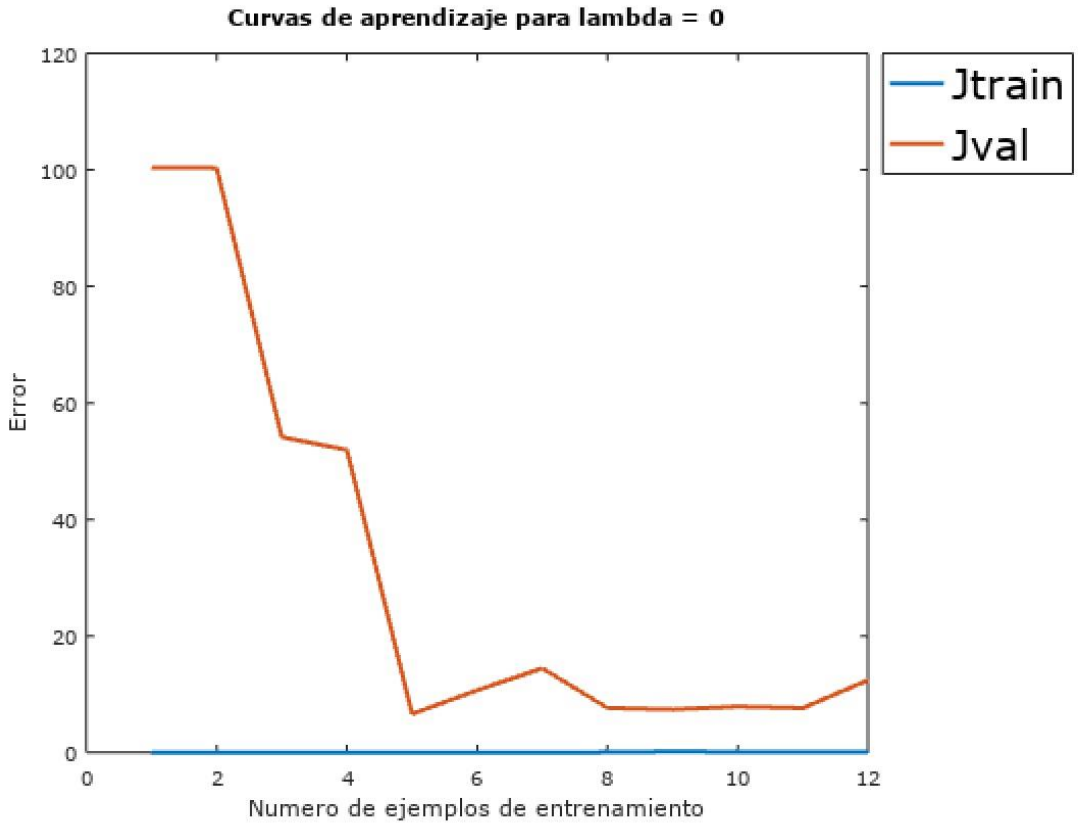
Tras ejecutar la función *fmincg.m* con la nueva matriz X, nos proporciona las thetas con el error más bajo, y hemos representado la función generada, donde se aprecia claramente el overfiting existente. Para realizar esta función se ha usado la función *plotFit.m* proporcionada. Este es el código usado:

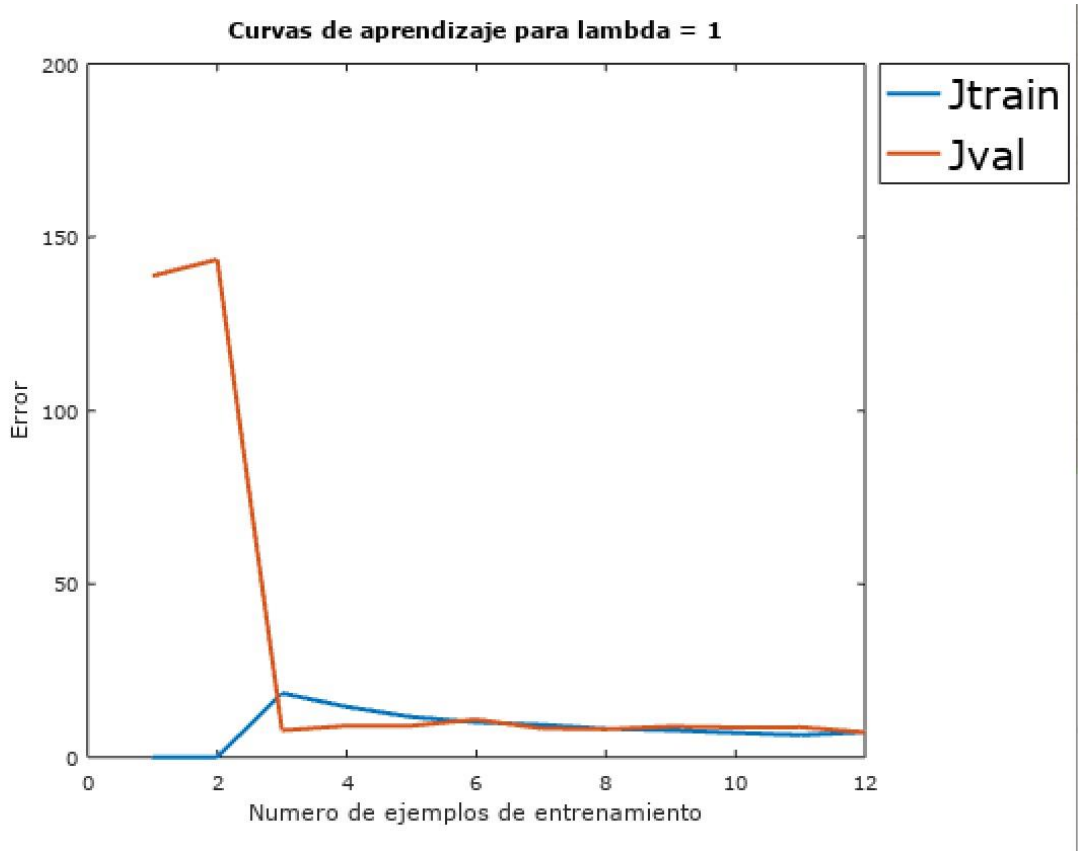
```
theta = fmincg(@(t)(coste(t,newX, y, 0)),ones(9,1),optimset('GradObj', 'on', 'MaxIter', 200));
plot(X, y, 'rx', 'MarkerSize', 5, 'LineWidth', 1.5);
hold on;
plotFit(min(X), max(X), media, desv, theta, 8);
hold off;
```

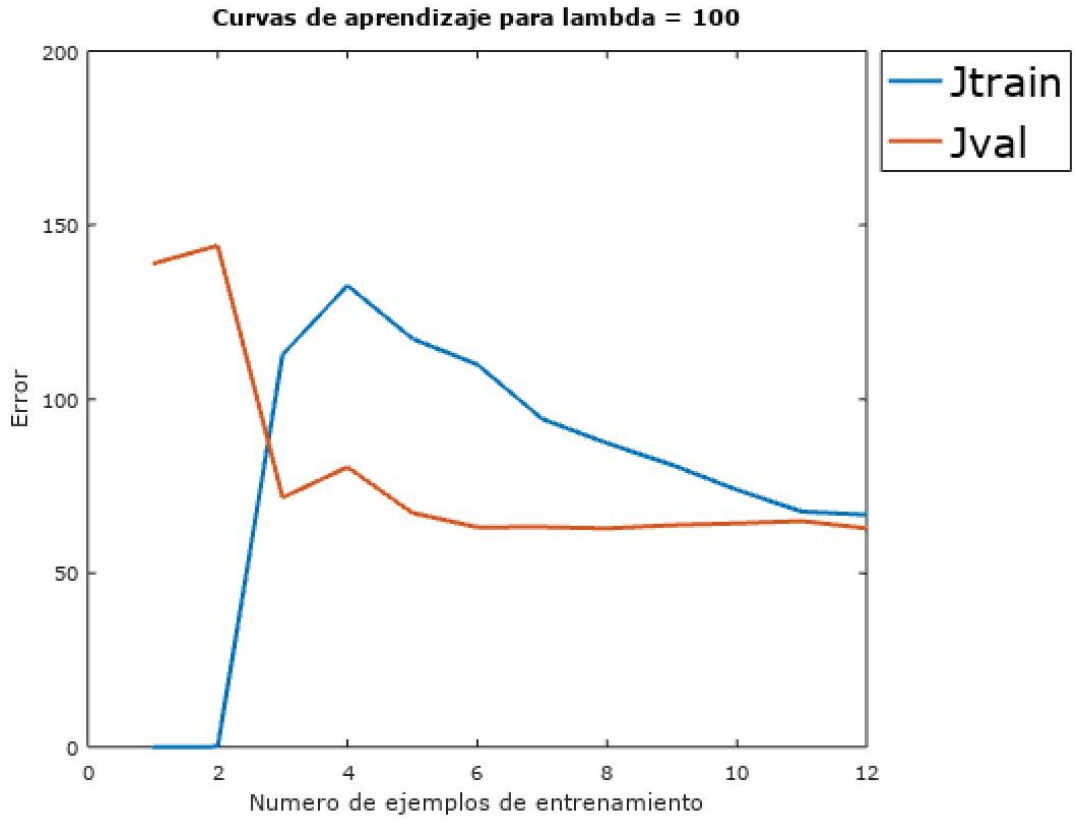
Y esta es la gráfica generada:



A continuación, hemos generado las curvas de aprendizaje para lamba = 0, 1 y 100. Las siguientes imágenes muestran estas curvas de aprendizaje:


Curvas de aprendizaje para lambda = 0
Jtrain
Jval
Error
Numero de ejemplos de entrenamiento
0
20
40
60
80
100
120
0
2
4
6
8
10
12

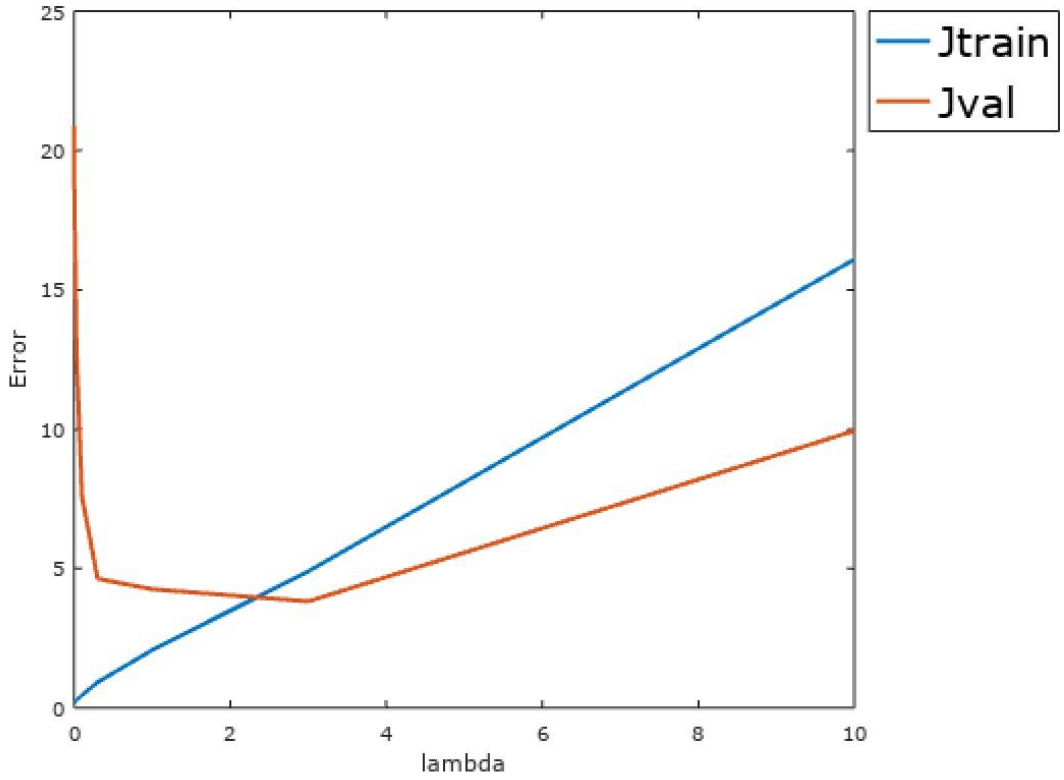

Curvas de aprendizaje para lambda = 1
Jtrain
Jval
Error
Numero de ejemplos de entrenamiento
0
50
100
150
200
0
2
4
6
8
10
12



Para ello, hemos utilizado la función *generarlcpool.m*, parecida a la anterior pero ahora se generan polinomios para los datos del check validation y de entrenamiento. Este es el código

```
function generarlcpol(X, y, lambda, Xval, yval, p)

  inittheta = ones(p + 1, 1);

  X = generatepoldata(X, p);
  Xval = generatepoldata(Xval, p);

  [newX, mediaX, desvX] = featureNormalize(X);

  Xval = bsxfun(@minus, Xval, mediaX);
  newXval = bsxfun(@rdivide, Xval, desvX);

  for i = 1:rows(X)

    theta = fmincg(@(t)(coste(t,newX(1:i,:), y(1:i), lambda)),inittheta,optimset('GradObj', 'on', 'MaxIter', 200));
    [Jtrain(i), grad] = coste(theta, newX(1:i,:), y(1:i), lambda);

    [Jval(i), grad] = coste(theta, newXval, yval, lambda);

  endfor

  plot([1:1:rows(X)], Jtrain, 'LineWidth', 2);
  xlabel('Numero de ejemplos de entrenamiento')
  ylabel('Error')
  title(['Curvas de aprendizaje para lambda = ',num2str(lambda)])
  hold on;
  plot([1:1:rows(X)], Jval, 'LineWidth', 2);
  hold off;

  h = legend ({'Jtrain'}, 'Jval');
  legend (h, 'location', 'northeastoutside');
  set (h, 'fontsize', 20);

endfunction
```

Por último, comprobamos que se ha realizado la regresión correctamente aplicando los datos de test a nuestra hipótesis, en base a distintas lambdas.
La siguiente función muestra la diferencia de error en los datos de entrenamiento y de validación usando diferentes lambas



Y este es el código usado:

```
function seleccionlambda(X, y, Xval, yval, Xtest, ytest, p)

  lambda = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10];

  inittheta = ones(p + 1, 1);

  X = generatepoldata(X, p);
  Xval = generatepoldata(Xval, p);
  Xtest = generatepoldata(Xtest, p);

  [newX, mediaX, desvX] = featureNormalize(X);

  Xval = bsxfun(@minus, Xval, mediaX);
  newXval = bsxfun(@rdivide, Xval, desvX);

  Xtest = bsxfun(@minus, Xtest, mediaX);
  newXtest = bsxfun(@rdivide, Xtest, desvX);

for i = 1:columns(lambda)

    theta = fmincg(@(t)(coste(t, newX, y, lambda(i))),inittheta,optimset('GradObj', 'on', 'MaxIter', 200));
    [Jtrain(i), grad] = coste(theta, newX, y, 0);

    [Jval(i), grad] = coste(theta, newXval, yval, 0);

endfor

  plot(lambda, Jtrain, 'LineWidth', 2);
  xlabel('lambda')
  ylabel('Error')
  hold on;
  plot(lambda, Jval, 'LineWidth', 2);
  hold off;

  h = legend ({'Jtrain'}, 'Jval');
  legend (h, 'location', 'northeastoutside');
  set (h, 'fontsize', 20);

endfunction
```

Como podemos observar el mínimo coste se alcanza con lambda = 3. Con dicho valor de lambda, el error con los datos de test es de 3.8599. Dicho coste lo obtenemos con el siguiente código:

```
theta = fmincg(@(t)(coste(t, newX, y, 3)),inittheta,optimset('GradObj', 'on', 'MaxIter', 200));
[Jtest, grad] = coste(theta, newXtest, ytest, 0)
```