APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 5

FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta sexta práctica es observar los efectos del sesgo y varianza y cómo conseguir reducir este efecto, utilizando regresión lineal.

2. Implementación

Para poder trabajar, lo primero que necesitaremos es una función de coste. Esta función seguirá la siguiente expresión:

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

Para el cálculo del gradiente seguiremos esta otra expresión:

$$j = 0 \rightarrow \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

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

La función de coste, que devolvería el coste y unos valores del gradiente, quedaría tal y como se puede observar en la figura 1:

```
function [J, grad] = costeRegularizado(Theta, X, y, lambda)
  m = rows(X);
  numCols = rows(Theta);

  J =((1/(2*m)) * (sum((hipotesis(X, Theta) - y) .^ 2))) + ((lambda/(2*m)) *
sum(Theta(2:end).^2));


  grad = (1/m) * sum((hipotesis(X, Theta) - y) .* X);

  if (m > 1)
    grad(:,2:end) += ((lambda/m) * Theta(2:end)');
  endif

  grad = grad';
endfunction
```

Figura 1

Para ver a lo que nos estamos enfrentando, aplicaremos la función de coste con los datos con los que vamos a trabajar, tal y como podemos ver en la figura 2:

```
>> [J, grad] = costeRegularizado(theta,[ones(m, 1) X], y, 1)

J =  303.99
grad =

  -15.303
  598.251
```

Figura 2

Obtenidos el siguiente valor de coste y el par de valores del gradiente, dibujaremos una recta para ver cómo se ajusta a los datos de entrenamiento. Para ello, se ha implementado la función *pintarEntrenamientoRegLin* que se encarga de plotear la imgen. Su implementación la podemos ver en la figura 3:

```
function pintarEntrenamientoRegLin(X, y)
  [Theta] = entrenameFmincg(X, y, 0);
  m = rows(X);
  plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);
  xlabel('Change in water level (x)');
  ylabel('Water flowing out of the dam (y)');
  hold on;
  plot(X, hipotesis([ones(m, 1) X], Theta), 'LineWidth', 2)
endfunction
```

Figura 3

Tras ejecutar la siguiente función con los siguientes comandos, obtenemos la siguiente imagen, como se puede observar en la figura 4:

```
>> pintarEntrenamientoRegLin(X, y);
```

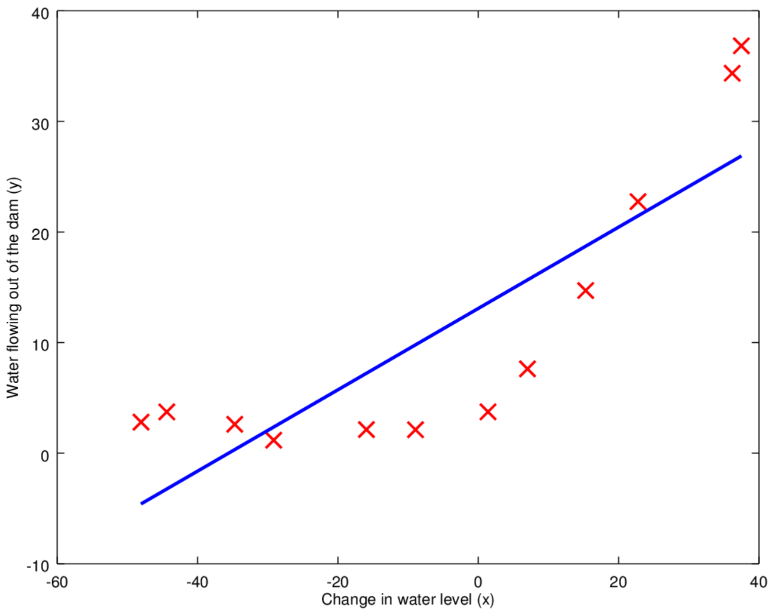


Figura 4

Para poder entrenar, se ha utilizado una función auxiliar en la función para mostrar la recta, de la figura 3, cuya implementación podemos ver en la figura 5:

```
function [Theta] = entrenameFmincg(X, y, lambda)
  options = optimset('MaxIter', 100, 'GradObj', 'on');
  X = [ones(rows(X), 1) X];
  Theta_ini = zeros(columns(X), 1);

  costFunction = @(t) costeRegularizado(t, X, y, lambda);
  Theta = fmincg(costFunction, Theta_ini, options);
endfunction
```

Figura 5

Como podemos observar tras el resultado de la ejecución de la figura 4, la hipótesis genera valores sesgados a la recta. El número de ejemplos a utilizar tiene un efecto directo sobre el error que se obtiene. Para poder visualizar una gráfica que nos ayude a ver estos casos se ha implementado la función *pintarEntrenamientoRegLin*, a la cual la podemos observar en la figura 6:

```
function pintarCurvaAprendizajeRegLineal(X, y, Xval, yval, lambda)
  m = rows(X);
  X = [ones(rows(X), 1) X];
  Xval = [ones(rows(Xval), 1) Xval];
  Theta_ini = zeros(2, 1);
  options = optimset('MaxIter', 100, 'GradObj', 'on');

  for i = 1 : m
    costFunction = @(t) costeRegularizado(t, X(1:i, :), y(1:i), 0);
    theta = fmincg(costFunction, Theta_ini, options);
    J(i) = costeRegularizado(theta, X(1:i, :), y(1:i), 0);
    Jval(i) = costeRegularizado(theta, Xval, yval, 0);
  endfor

  plot(1:m, J, 1:m, Jval);
  title('Curva de aprendizaje para la regresion lineal')
  legend('Entrenamiento', 'Validacion')
  xlabel('Numero de ejemplos de entrenamiento')
  ylabel('Error')
  axis([0 12 0 150])
endfunction
```

Figura 6

En función de la figura 6, solamente se realizan subconjuntos de los datos ‘X’, no de ‘Xval’. Tras la ejecución de la función anterior, se obtiene la gráfica que podemos ver en la figura 7:

```
>> pintarCurvaAprendizajeRegLineal(X, y, Xval, yval, 0);
```

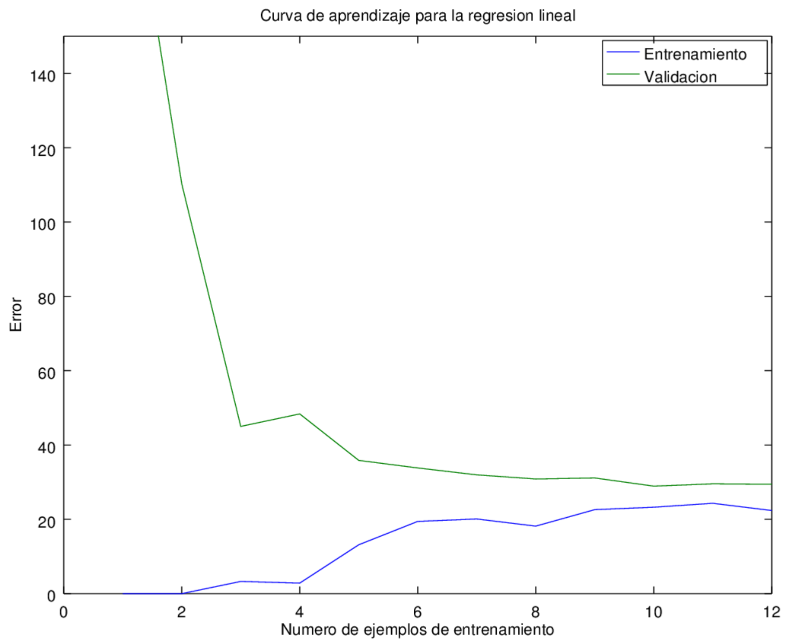


Figura 7

El método de la curva de aprendizaje, nos ayuda pues, a saber si el aprendizaje está sesgado. Como se puede observar en la figura 7, estamos ante un caso de sesgo, ya que al aumentar el número de ejemplos de entrenamiento, la curva de error se acerca a la propia curva del número de ejemplos de entrenamiento.

En esta práctica, para poder solucionar esto, se hará uso de una hipótesis polinomial, la cual hay que construir. Para ello, se ha implementado la función *pintarRegPolinómica*, la cual podemos observar en la figura 8:

```
function pintarRegPolinomica(X, y, p)
  options = optimset('MaxIter', 100, 'GradObj', 'on');
  m = rows(X);

  Xnorm = adaptaRegresionPoli(X, p);
  [Xnorm, mu, sigma] = featureNormalize(Xnorm);
  Xnorm = [ones(m, 1), Xnorm];


  Theta_ini= zeros(columns(Xnorm), 1);
  costFunction = @(t) costeRegularizado(t, Xnorm, y, 0);
  theta = fmincg(costFunction, Theta_ini, options);

  plotFit(-60, max(X), mu, sigma, theta, p)
  hold on
  plot(X, y, "xr",'MarkerSize', 7, 'LineWidth', 2)
  title('Regresion polinomica (lambda = 0)')
  xlabel('Cambio en el nivel del agua (x)')
  ylabel('Agua que derrama la presa (y)')
endfunction
```

Figura 8

Se ha utilizado en la figura 8 una función auxiliar, *adaptaRegresionPoli*, que se encarga de aplicar potencias para obtener una nueva matriz, con la que obtener la hipótesis polinomial, tal y como se puede observar en la figura 9:

```
function D = adaptaRegresionPoli(X, p)
  %X -> m x 1
  m = rows(X);

  D = zeros(m, p - 1); %p - 1 porque posteriormente ingreso la primer columna como
X

  D = [X D];
  for i = 2 : p
    D(:, i) = (D(:, 1) .^ i);
  endfor
end
```

Figura 9

Si ejecutamos la función de la figura 8 obtenemos la gráfica de la figura 10:

```
>> pintarRegPolinomica(X, y, 8)
```

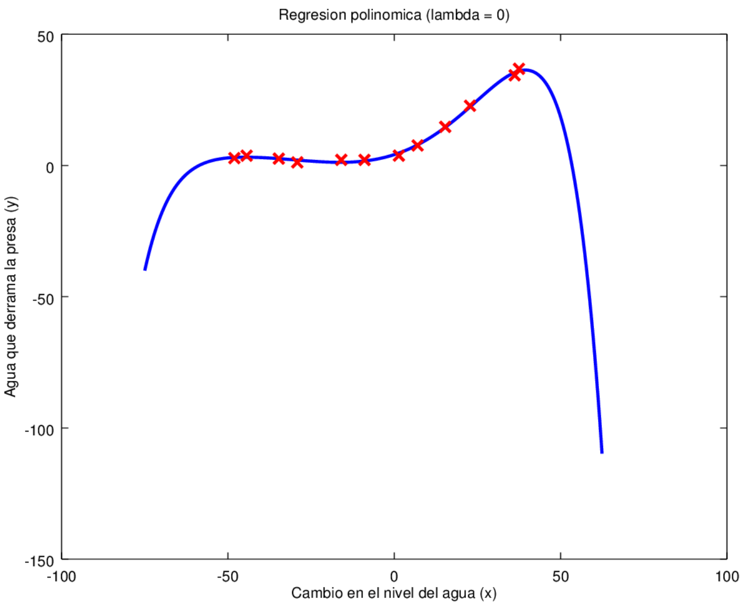


Figura 10

Teniendo en cuenta la idea base de la implementación de la figura 8, vamos a realizar la técnica anterior para obtener curvas de aprendizaje, variando el número de ejemplos de entrenamiento para ver el efecto de sesgo. Para ello, se ha implementado la función *pintarCurvaAprendizajeRegresionPolinomial*, cuya implementación podemos observar en la figura 11.

```
function pintarCurvaAprendizajeRegresionPolinomial(X, y, Xval, yval, lambda,
p)
  options = optimset('MaxIter', 100, 'GradObj', 'on');
  m = rows(X);

  Xnorm = adaptaRegresionPoli(X, p);
  [Xnorm, mu, sigma] = featureNormalize(Xnorm);
  Xnorm = [ones(m, 1), Xnorm];

  Xnorm_val = adaptaRegresionPoli(Xval, p);
  Xnorm_val = bsxfun(@minus, Xnorm_val, mu);
  Xnorm_val = bsxfun(@rdivide, Xnorm_val, sigma);
  Xnorm_val = [ones(size(Xnorm_val, 1), 1), Xnorm_val];

  Theta_ini= zeros(columns(Xnorm), 1);
  costFunction = @(t) costeRegularizado(t, Xnorm, y, 0);
  theta = fmincg(costFunction, Theta_ini, options);



  for i = 1 : m
    costFunction = @(t) costeRegularizado(t, Xnorm(1:i, :), y(1:i), lambda);
    theta = fmincg(costFunction, Theta_ini, options);
    J(i) = calcularError(theta, Xnorm(1:i, :), y(1:i));
    Jval(i) = calcularError(theta, Xnorm_val, yval);
  endfor

  plot(1:m, J, 'LineWidth', 3, (1:m), Jval, 'LineWidth', 2)
  axis([0 13 0 100])
  title('Curva de aprendizaaje para la regresion polinomica (lambda = 0)')
  legend('Entrenamiento', 'Validacion')
  xlabel('Numero de ejemplos de entrenamiento')
  ylabel('Error')
end
```

Figura 11

Se ha utilizado una función auxiliar para calcular solamente el error, *calcularError*, cuya implementación podemos observar en la figura 12:

```
function e = calcularError(Theta, X, y)
  m = rows(X);
  numCols = rows(Theta);

  e = (1/(2*m)) * (sum((hipotesis(X, Theta) - y) .^ 2));
end
```

Figura 12

Como el valor de lambda afecta a las formas de las curvas de aprendizaje, realizaremos varias llamadas a la función *pintarCurvaAprendizajeRegresionPolinomial* con diversos valores de lambda.

Con lambda igual a 0, obtenemos la gráfica de la figura 13:

```
>> pintarCurvaAprendizajeRegresionPolinomial(X, y, Xval, yval, 0, 8)
```

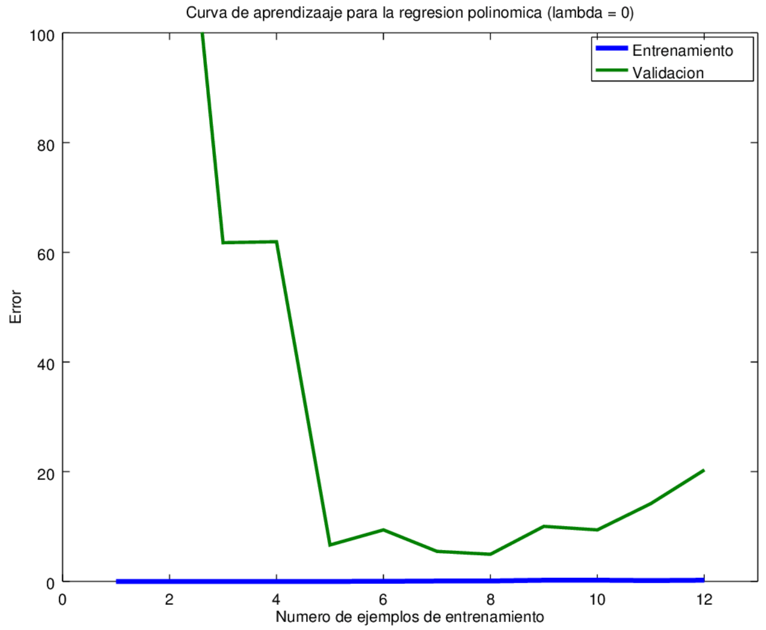
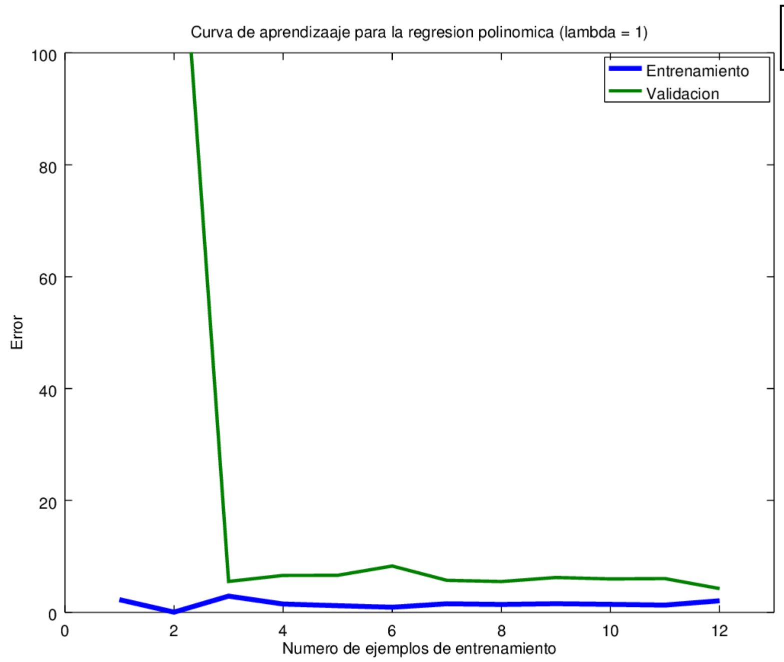


Figura 13

Con lambda igual a 1, obtenemos la gráfica de la figura 14:

```
>> pintarCurvaAprendizajeRegresionPolinomial(X, y, Xval, yval, 1, 8)
```

Figura 14



Con lambda igual a 100, obtenemos la gráfica de la figura 15:

>> pintarCurvaAprendizajeRegresionPolinomial(X, y, Xval, yval, 100, 8)

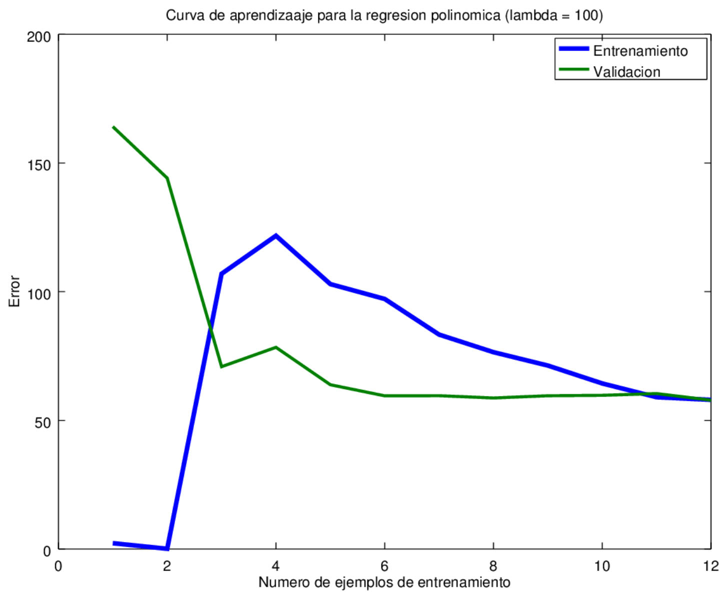


Figura 13

Para saber que lambda nos conviene más, se ha implementado la función *pintarError*, que nos muestra una gráfica donde podemos observar con que lambda es menor el error. Su implementación la podemos observar en la figura 14:

```
function pintarErrorLambda(X, y, Xval, yval, p)
  lambda = [0, 0.001, 0.003, 0.01, 0.03, 1, 3, 10];

  options = optimset('MaxIter', 100, 'GradObj', 'on');
  m = rows(X);
  Xnorm = adaptaRegresionPoli(X, p);
  [Xnorm, mu, sigma] = featureNormalize(Xnorm);
  Xnorm = [ones(m, 1), Xnorm];

  Xnorm_val = adaptaRegresionPoli(Xval, p);
  Xnorm_val = bsxfun(@minus, Xnorm_val, mu);
  Xnorm_val = bsxfun(@rdivide, Xnorm_val, sigma);
  Xnorm_val = [ones(rows(Xnorm_val), 1), Xnorm_val];

  Theta_ini= zeros(columns(Xnorm), 1);

  for i = 1:columns(lambda)
    costFunction = @(t) costeRegularizado(t, Xnorm, y, lambda(i));
    theta = fmincg(costFunction, Theta_ini, options);
    J(i) = calcularError(theta, Xnorm, y);
    Jval(i) = calcularError(theta, Xnorm_val, yval);
  endfor

  plot(lambda, J, 'LineWidth', 3, lambda, Jval, 'LineWidth', 3)

  axis([0 10 0 20])
  legend('Entrenamiento', 'Validacion')
  xlabel('Lambda')
  ylabel('Error')
end
```

Figura 14

Una vez encontrado la mejor lambda, calcularemos su error de hipótesis. Para ello usaremos unos ejemplos de entrenamientos diferentes. La función que se encargará de tal cometido es *errorHipotesis*, cuya implementación podemos observar en la figura 15:

```
function e = errorHipotesis(X, y, Xtest, ytest, p)
  options = optimset('MaxIter', 100, 'GradObj', 'on');
  m = rows(X);
  Xnorm = adaptaRegresionPoli(X, p);
  [Xnorm, mu, sigma] = featureNormalize(Xnorm);
  Xnorm = [ones(m, 1), Xnorm];

  Xnorm_test  = adaptaRegresionPoli(Xtest, p);
  Xnorm_test  = bsxfun(@minus, Xnorm_test, mu);
  Xnorm_test  = bsxfun(@rdivide, Xnorm_test, sigma);
  Xnorm_test = [ones(rows(Xtest), 1) Xnorm_test];

  theta_ini_norm = zeros(columns(Xnorm), 1);

  costFunction = @(t) costeRegularizado(t, Xnorm, y, 3);
  theta = fmincg(costFunction, theta_ini_norm, options);

  e = costeRegularizado(theta, Xnorm_test, ytest, 0);
endfunction
```

Figura 15

Ejecutando la función de la figura 15, obtenemos el resultado que podemos observar en la figura 16:

```
>> e = errorHipotesis(X, y, Xtest, ytest, 8)
e =  3.8599
```

Figura 16