Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Aprendizaje automático

Practica 2: Regularización y selección de modelos

Autor:
Nerea Gallego Sánchez (801950)

7 de diciembre de 2022

Índice

1. Introducción y objetivos

Esta práctica consiste en aplicar y comparar las técnicas de regularización y validación cruzada para selección de modelos en una aplicación de regresión lineal en un caso real.

El caso de estudio es la predicción del precio de un coche en función de tres atributos: la antigüedad, los kilómetros y la potencia.
El objetivo es encontrar el mejor modelo polinómico para predecir el precio.

Los distintos casos que se van a probar son: el grado del polinomio para la antigüedad del coche, el grado del polinomio para los kilómetros, el grado del polinomio para la potencia y la elección del parámetro landa para realizar el ajuste.

2. Estudio previo

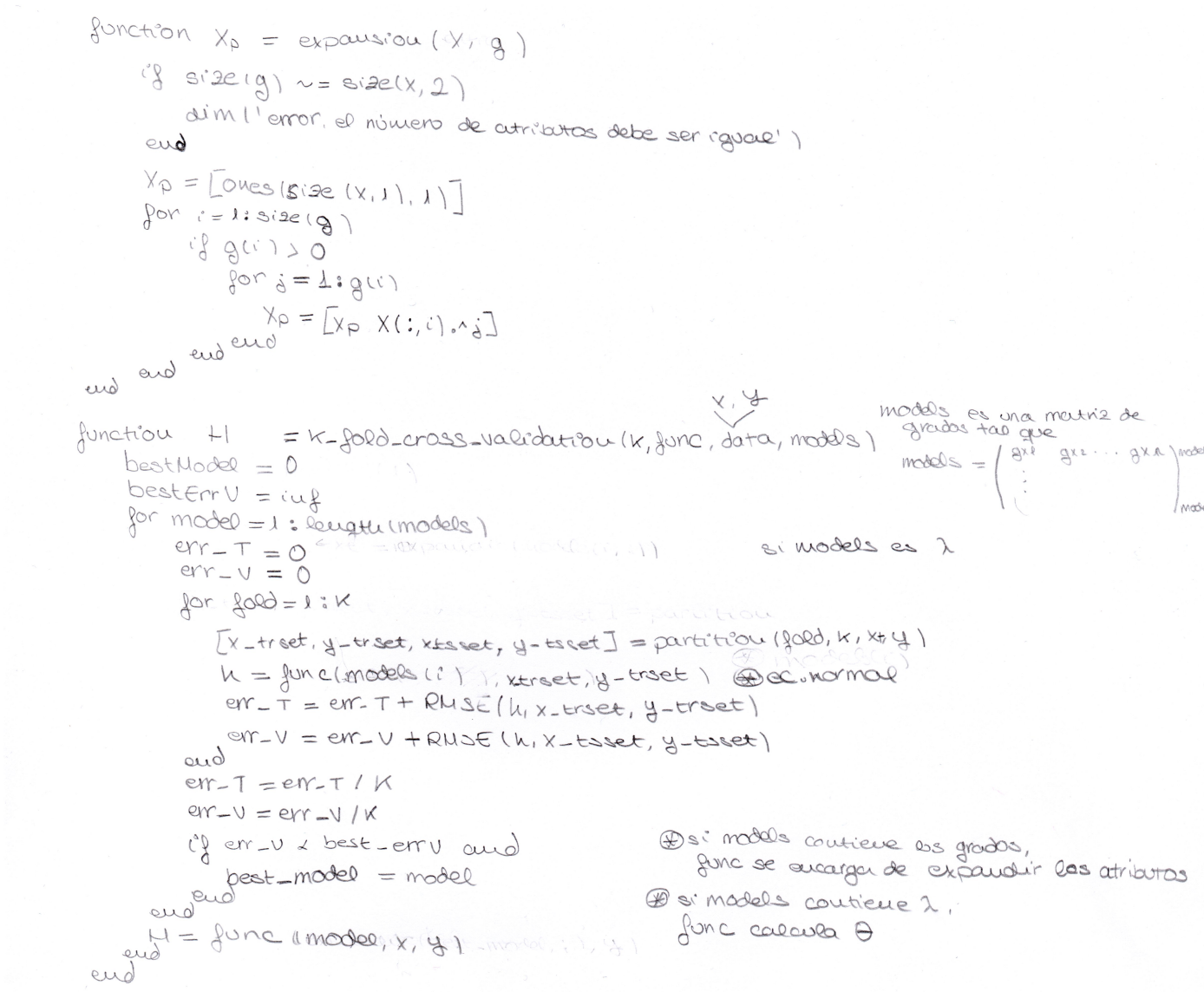

Figura 1: Estudio previo a la práctica.

3. Selección del grado del polinomio para la antigüedad del coche

En este apartado se pide programar el algoritmo de *k-fold cross-validation* para elegir el grado del polinomio de antigüedad (entre 1 y 10) dejando fijos los kilómetros y la potencia con grado 1.

Para ello, primero se ha creado una función de *k-fold cross-validation* que dados unos datos de entrada, unos datos de salida, un algoritmo de entrenamiento, modelos de entrenamiento, una función para calcular el error y la cantidad de folds, devuelve el índice del mejor modelo obtenido de entre los modelos pasados para evaluar.
La función final es muy similar a la realizada en el apartado previo.

```
function [H, errT, errV] = k_fold_cross_validation(k, func, X, y, models, funcErr)
    bestModel = 0;
    bestErrV = inf;
    size(models,1)
    errT = [];
    errV = [];
    for model = 1:size(models,1)
        err_T = 0;
        err_V = 0;
        for fold = 1:k
            [x_tsset, y_tsset, x_trset, y_trset] = particion(fold,k,X,y);
            h = func(models(model,:), x_trset, y_trset);
            err_T = err_T + funcErr(h, x_trset, y_trset, models(model,:));
            err_V = err_V + funcErr(h, x_tsset, y_tsset, models(model,:));
        end
        err_T = err_T / k;
        err_V = err_V / k;
        errT = [errT err_T];
        errV = [errV err_V];
        if err_V < bestErrV
            bestModel = model;
            bestErrV = err_V;
        end
    end
    H = bestModel;
end
```

La función evalúa para cada modelo, los datos de entrenamiento y calcula una media del error utilizando como datos de entrada cada fold. Finalmente, mira si el error obtenido es el mínimo hasta ahora. Una vez evaluados todos los modelos de esta manera, devuelve el mejor modelo obtenido.

Para realizar esta estimación, se han cargado los datos del set de datos de entrenamiento proporcionado. A continuación, se ha creado una matriz con las distintas combinaciones de los grados de los atributos que se quieren probar. Esta matriz tiene en sus filas el grado para cada uno de los atributos en una interacción.

Se ha creado una función para entrenar el modelo. Esta función es *entrenaExpansion*, ya que su función principal es entrenar el modelo realizando la expansión de atributos.

```
function H = entrenaExpansion(G,X,y)
    Xp = expandir(X,G);
    [Xp,mu,sigma] = normalizar(Xp);
    H = ecuacionNormal(Xp,y);
    H = desnormalizar(H,mu,sigma);
end
```

La función tiene como parámetros de entrada el modelo con el que se quiere entrenar, el conjunto de datos de entrenamiento y el conjunto de datos de salida. Como se puede observar en el código, la función expande los datos de entrada, a continuación los normaliza. Luego calcula el vector de pesos utilizando

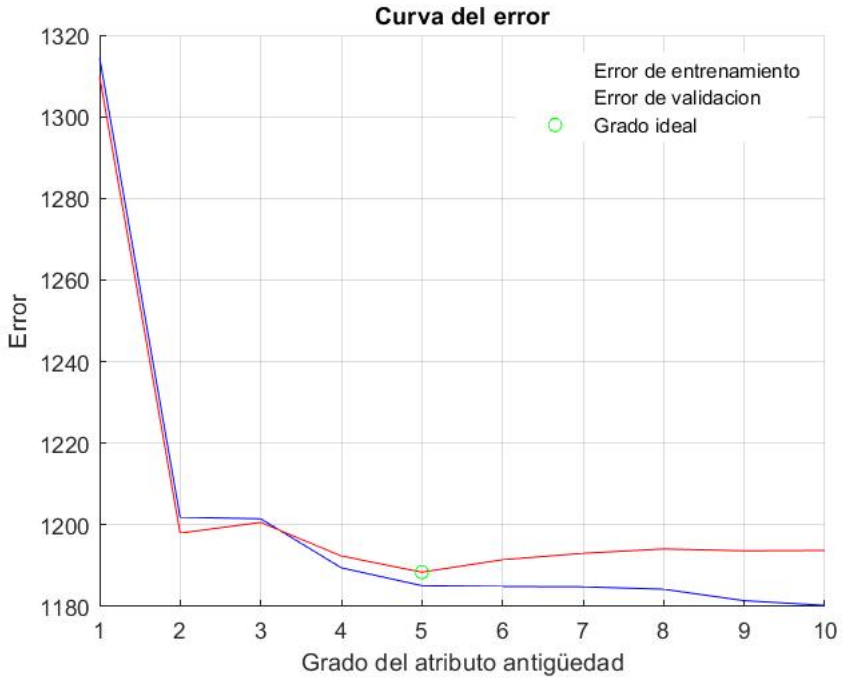

Figura 2: Curva de evolución de los errores RMSE de entrenamiento y validación

la ecuación normal y, por último, desnormaliza el vector de pesos obtenido.
En este caso se ha decidido utilizar la ecuación normal para calcular el vector de pesos ya que se realizan muchas iteraciones con el algoritmo de k-fold y esta función es más rápida que el algoritmo de descenso de gradiente.

Como función de error, se ha ultizado $RMSE$, pero como se ha decidido realizar la expansión de atributos dentro de la función que calcula el vector de pesos, es necesario crear también una función que calcule el error, ya que para calcular dicho error, será necesario realizar también la expansión de atributos de los datos de entrada.
La función es la siguiente:

```
function h = errorExpansion(h,X,y,model)
    xp = expandir(X,model);
    h = RMSE(h,xp,y);
end
```

Se ha invocado a la función $k - fold\ cross - validation$ con las funciones indicadas anteriormente (función 3 para el entrenamiento y función 3 para el error), 10 folds, los distintos modelos (en este caso, los grados de los atributos a expandir) y los datos del set de entrenamiento con la salida esperada.

El resultado obtenido como grado ideal para el atributo de antigüedad es 5.

En la gráfica 2 se puede observar como el menor error obtenido con los datos de validación se encuentra cuando el grado del polinomio es 5. A partir de ese momento se produce sobreajuste.
La linea azul de la gráfica corresponde con el error obtenido con los datos de entrenamiento y la linea roja muestra el error obtenido con los datos de validación.

4. Selección del grado del polinomio para los kilómetros

Se pide obtener el mejor grado de polinomio para el atributo de los kilómetros, dejando fijo el mejor grado obtenido en el apartado anterior para el atributo de antigüedad y utilizando grado 1 para el atributo de potencia.

Se utiliza el algoritmo de $k - fold\ cross - validation$ (función 3) para elegir el grado del polinomio

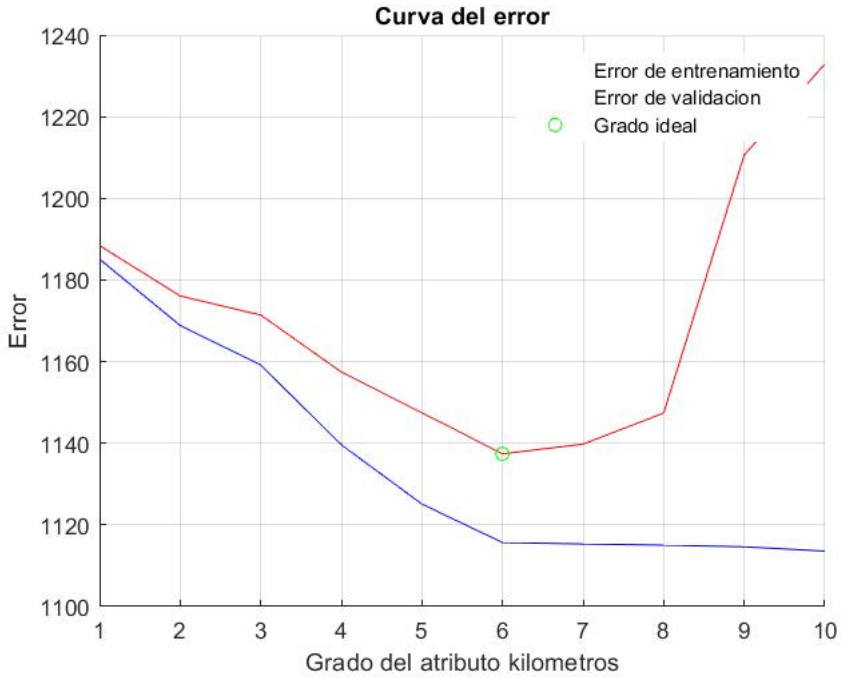

Figura 3: Curva de evolución de los errores RMSE de entrenamiento y validación

de kilómetros (entre 1 y 10) dejando fijos los grados de antigüedad y potencia. Para el polinomio de antigüedad se elige el mejor grado obtenido en el apartado anterior. Es decir, 5. Para el atributo de potencia se utiliza grado 1.

Igual que ocurria en el apartado anterior, se ha creado una matriz con las distintas combinaciones de los grados de los atributos que se quieren probar. En este caso, se ha mantenido fijo el primer y tercer atributo, asignándoles valores de 5 y 1 respectivamente. En el caso del segundo atributo se le han asignado valores desde 1 hasta 10.
Se ha utilizado la función *entrenaExpansion* (función 3) para entrenar el modelo utilizando expansión de atributos. También se ha utilizado la función *errorExpansion* (función 3) para calcular el error con cada modelo.

Para realizar la evaluación se ha invocado a la función $k - fold\ cross - validation$ con las funciones indicadas anteriormente, 10 folds, los grados de los atributos que se quieren expandir y los datos del set de entrenamiento con su salida esperada.

El resultado obtenido es que el grado ideal para el atributo de kilómetros es 6.

En la gráfica 3 se puede observar como el menor error obtenido con los datos de validación se encuentra cuando el grado del polinomio es 6. A partir de ese momento, al aumentar el grado del polinomio se produce sobreajuste. De la misma manera, se puede ver como hay subajuste cuando el polinomio tiene un grado menor que 6.
La linea azul de la gráfica corresponde con el error obtenido con los datos de entrenamiento y la linea roja muestra el error obtenido con los datos de validación.

5. Selección del grado del polinomio para la potencia

En este apartado de pide elegir mediante $k - fold\ cross - validation$ el grado del polinomio de la potencia, dejando fijos los grados encontrados en los apartados anteriores.

Se utiliza el algoritmo de $k - fold\ cross - validation$ (funcion 3) para elegir el grado del polinomio potencia (entre 1 y 10) dejando fijos los grados de antigüedad y kilómetros.
Tanto para el atributo de antigüedad como para el atributo de potencia se elige el mejor grado obtenido

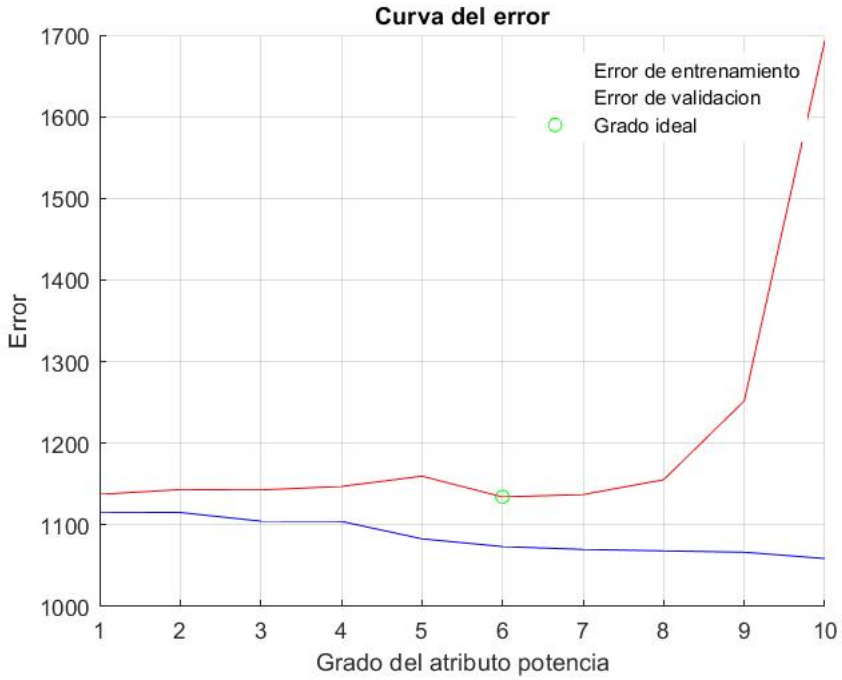

Figura 4: Curva de evolución de los errores RMSE de entrenamiento y validación

en los apartados anteriores. Es decir, para el atributo de antigüedad se utiliza el grado 5 y para el atributo de potencia se utiliza el grado 6.

Igual que ocurría en el apartado anterior, se ha creado una matriz con las distintas combinaciones de los grados de los atributos que se quieren probar. En este caso, se ha mantenido fijo el primer y segundo atributo, asignándoles valores de 5 y 6 respectivamente. En el caso del tercer atributo se le han asignado valores desde 1 hasta 10.
Se ha utilizado la función *entrenaExpansion* (función 3) para entrenar el modelo utilizando la expansión de atributos. También se ha utilizado la función *errorExpansion* (función 3) para calcular el error con cada modelo.

Para realizar la evalución se ha invocado a la función $k - fold\ cross - validation$ con las funciones indicadas anteriormente, 10 folds, los modelos a probar y los datos del set de entrenamiento y su salida esperada.

El resultado obtenido es que el grado ideal para el atributo de potencia es 6.
En la gráfica 4 se puede observar como el menor error obtenido con los datos de validación se encuentra cuando el grado del polinomio es 6. A partir de eso momento, al aumentar el grado del polinomio se produce sobreajuste. De la misma manera, se puede ver como hay subajuste cuando el polinomio tiene un grado menor que 6.
La linea azul de la gráfica corresponde con el error obtenido con los datos de entrenamiento y la linea roja muestra el error obtenido con los datos de validación.

A continuación, una vez encontrado el mejor grado de polinomio para cada uno de los atributos, se pide entrenar el modelo con todos los datos y calcular el RMSE con los datos de test.
Para ello, se han cargado de nuevo los datos de entrenamiento, se han expandido los atributos correspondientes y se ha calculado el vector de pesos. Primero se han normalizado los datos, luego se ha aplicado la ecuación normal, y por último se ha desnormalizado el vector de pesos.
Con estos pasos se obtiene el vector de pesos final del modelo.
Ahora se han cargado los datos de test, se han expandido los atributos correspondientes con el mismo grado que para los datos de entrenamiento.

Para calcular el RMSE con los datos de test, se ha pasado a la función $RMSE$ proporcionada por

los profesores el vector de pesos calculado, los datos con los atributos expandidos y la salida esperada. La función devuelve el error obtenido.
El error que se obtiene con los datos de test es $RMSEtest = 1{,}0093e+03$.

6. Regularización

En este apartado se pide realizar el ajuste para un polinomio de grado 10 para la antigüedad, los kilómetros y la potencia utilizando regularización. La tarea consiste en elegir el parámetro landa mediante $k - fold\ cross - validation$.

Para realizar este ha partado ha sido necesario crear una nueva función para calcular el vector de pesos, ya que ahora se debe utilizar regularización.
La función creada para la regularización es la siguiente:

```
function theta = entrenaRegularizacion(lambda, X, y)
    [Xp,mu,sigma] = normalizar(X);
    H = Xp'*Xp + lambda*diag([0 ones(1,size(Xp,2)-1)]);
    theta = H \ (Xp'*y);
    theta = desnormalizar(theta,mu,sigma);
end
```

Se le pasan como parámetros el parámetro landa, los datos de entrada y los datos de salida.
La función normaliza los datos, a continuación aplica la regularización para calcular el vector de pesos. Por último se desnormalizan los datos y devuelve el vector de pesos.

También ha sido necesario crear una nueva función que calcule el error dado el modelo, los datos de entrada, los datos de salida esperados y el vector de pesos calculado.
La función es la siguiente:

```
function err = errorRegularizacion(t,X,y,model)
    err = RMSE(t,X,y);
end
```

Se utiliza el algoritmo de $k - fold\ cross - validation$ (función 3) para elegir el parámetro de landa ideal.

Para realizar este estudio, se han cargado los datos de entrenamiento y se han expandido a polinomios de grado 10 los tres atributos.
A continuación se ha creado un vector con posibles parámetros de landa para evaluar. Tras realizar sucesivas pruebas de parámetros, se ha decidido elegir valores entre 0,000000001 y 0,00001 de manera que cada valor es el doble que el anterior.
Para evaluar los modelos, se ha invocado la función $k - fold\ cross - validation$ con las funciones indicadas anteriormente (función 3 para calcular los pesos y función 3 para calcular el error), 10 folds, los parámetros de landa, los datos del set de entrenamiento y su salida esperada.

Tras ejecutar este algoritmo de evaluación se ha obtenido que el parámetro landa ideal es $lambda = 2{,}5600e-07$.
En la grafica 5 se puede observar como el menor error obtenido con los datos de validación se encuentra cuando el factor de regularización es $lambda = 2{,}5600e-07$. A partir de ese momento se comienza a producir subajuste. Se puede observar como, en la parte derecha de la gráfica, se produce sobreajuste del modelo, ya que el vector de lambda es muy pequeño.
La linea azul de la gráfica corresponde con el error obtenido con los datos de entrenamiento y la linea roja muestra el error obtenido con los datos de validación.

A continuación, una vez encontrado el mejor grado de polinomio para cada uno de los atributos, se pide entrenar el modelo con todos los datos y calcular el RMSE con los datos de test.
Para ello, se han cargado de nuevo los datos de entrenamiento, se han expandido los atributos correspondientes con grado de polinomio 10 y se ha calculado el vector de pesos mediante la función *entrenaRegularizacion* (funcion 6).

Una vez obtenido el vector de pesos, se han cargado los datos de test y se han expandido los atributos correspondientes con el mismo grado que los datos de entrenamiento.

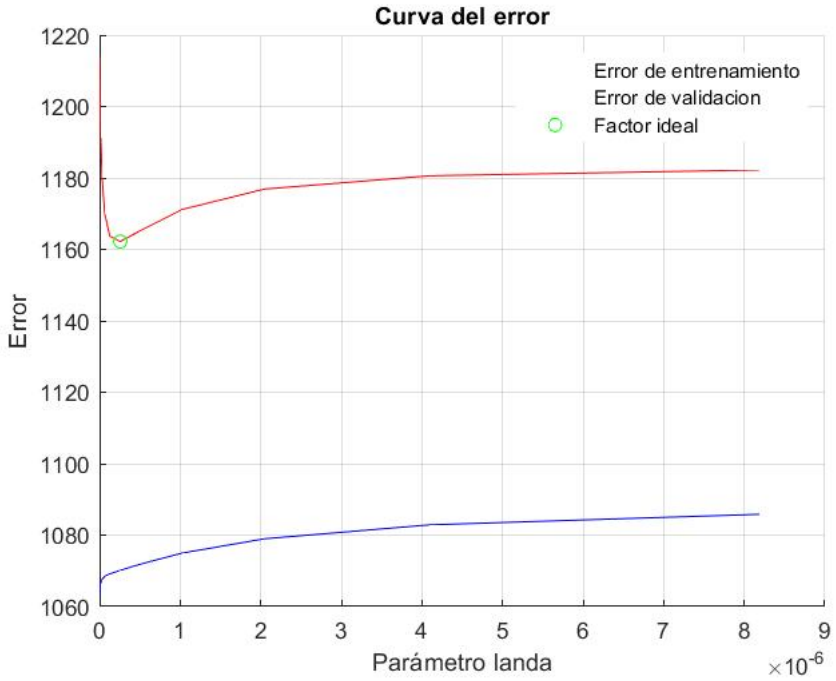

Figura 5: Curva de evolución de los errores RMSE de entrenamiento y validación

Para calcular el RMSE con los datos de test, se ha pasado a la función RMSE proporcionada por los profesores el vector de pesos calculado, los datos con los atributos expandidos y la salida esperada.
La función devuelve el error obtenido.
El error que se obtiene con los datos de test es $RMSEtest = 1{,}0097e+03$.

El error obtenido con este modelo es algo mayor que el error obtenido con el modelo anterior. Esto puede deberse a que en este caso se utiliza un modelo bastante más complejo y, a pesar de que la regularización penaliza a los modelos complejos, en este caso no es suficientemente bueno como para superar al modelo polinómico bien elegido.