

Tema 2: Aprendizaje Supervisado

Regresión Lineal Regularizada II

Prof. Wladimir Rodríguez

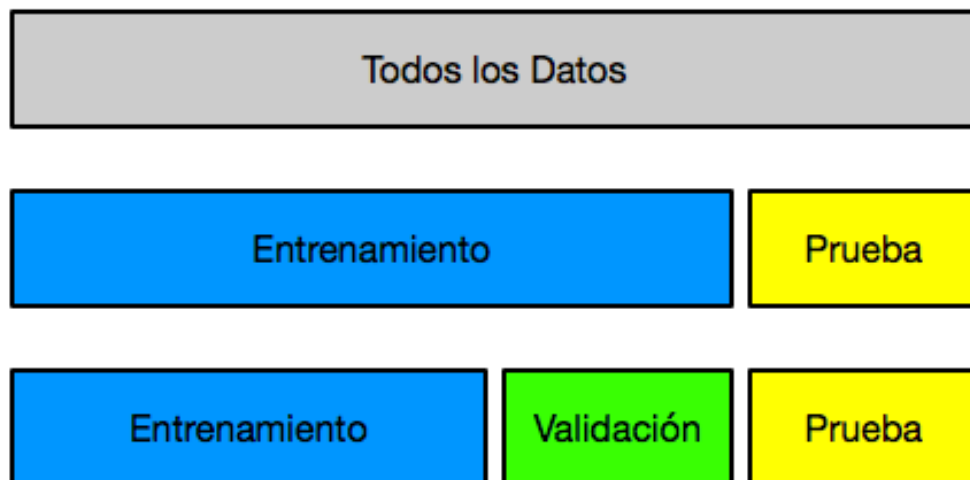
wladimir@ula.ve

Departamento de Computación

Selección del parámetro de penalización λ_2 (alfa) en la Regresión Lineal Ridge

Validación Cruzada y Selección de Atributos

Al igual que el grado polinomial, la penalización L_2 es un parámetro "mágico" que debemos seleccionar. Podríamos utilizar el enfoque de validación como lo hicimos en anteriormente, pero ese enfoque tiene una gran desventaja: deja menos observaciones disponibles para el entrenamiento. La validación cruzada busca superar este problema utilizando todo el conjunto de entrenamiento de una manera inteligente.



Implementaremos una especie de validación cruzada llamada validación cruzada "k-fold". El método obtiene su nombre porque implica dividir el conjunto de entrenamiento en k segmentos de tamaño prácticamente igual. Similar al método del conjunto de validación, medimos el error de validación con uno de los segmentos designados como el conjunto de validación.

La diferencia principal es que repetimos el proceso k veces de la siguiente manera:

- Seleccione el segmento 0 como el conjunto de validación y ajuste un modelo en el resto de los datos y evalúelo con este conjunto de validación
- Seleccione el segmento 1 como el conjunto de validación y ajuste un modelo en el resto de los datos y evalúelo con este conjunto de validación
- ...

- Seleccione el segmento k-1 como el conjunto de validación y ajuste un modelo en el resto de los datos y evalúelo con este conjunto de validación

	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4
Entrenamiento 1	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4
Entrenamiento 2	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4
Entrenamiento 3	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4
Entrenamiento 4	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4
Entrenamiento 5	Segmento 0	Segmento 1	Segmento 2	Segmento 3	Segmento 4

Después de este proceso, calculamos el promedio de los k errores de validación, y lo usamos como una estimación del error de generalización. Observe que todas las observaciones se usan tanto para el entrenamiento como para la validación, a medida que iteramos sobre segmentos de datos.

Para estimar bien el error de generalización, es crucial mezclar los datos de entrenamiento antes de dividirlos en segmentos. Reservamos el 10% de los datos como conjunto de prueba y barajamos el resto. (Asegúrese de usar `random state = 0` para obtener una respuesta consistente.)

```
In [1]: import math
import random
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Lectura de los datos

```
In [2]: ventas = pd.read_csv('../datos/kc_house_data.csv')
ventas = shuffle(ventas)
atributos = list(ventas)
atributos.remove('price')
atributos.remove('date')
atributos.remove('id')
```

Dividir los datos en dos conjuntos uno de entrenamiento y validación y el otro de prueba

```
In [3]: ventas_entre_val, ventas_prueba = train_test_split(ventas, test_size=0.1, random_state=0)
ventas_entre_val.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	v
17055	5561000920	20140502T000000	630000.0	4	2.75	2710	37277	2.0		0
15393	7708000010	20141107T000000	421000.0	3	2.00	1420	12655	1.0		0
16851	5490210670	20140822T000000	449950.0	4	2.50	2070	7312	1.0		0
17176	3763300005	20140520T000000	325000.0	4	2.25	1870	9680	1.0		0
4791	2210500019	20150324T000000	937500.0	3	1.00	1320	8500	1.0		0

5 rows × 21 columns

Crear los segmentos del conjunto de entrenamiento y validación

Una vez que los datos son barajados, lo dividimos en segmentos iguales. Cada segmento debe recibir n/k elementos, donde n es el número de observaciones en el conjunto de entrenamiento y k es el número de segmentos. Puesto que el segmento 0 comienza en el índice 0 y contiene n/k elementos, termina en el índice $(n/k) - 1$. El segmento 1 comienza donde el segmento 0 se detuvo, en el índice (n/k) . Con n/k elementos, el segmento 1 termina en el índice $(n*2/k) - 1$. Continuando de esta manera, deducimos que el segmento i comienza en el índice $(n*i/k)$ y termina en $(n*(i+1)/k) - 1$. Con este patrón en mente, escribimos un lazo corto que imprime los índices inicial y final de cada segmento, sólo para asegurarse de que está haciendo las divisiones correctas.

```
In [4]: n = len(ventas_entre_val)
k = 10 # validación cruzada 10-fold
```

```
for i in range(k):
    inicio = (n*i)//k
    fin = (n*(i+1))//k-1
    print(i, (inicio, fin))
```

```
0 (0, 1944)
1 (1945, 3889)
2 (3890, 5834)
3 (5835, 7779)
4 (7780, 9724)
5 (9725, 11669)
6 (11670, 13614)
7 (13615, 15559)
8 (15560, 17504)
9 (17505, 19450)
```

Definiciones de algunas funciones de utilidad

```
In [5]: def imprimir_coeficientes(modelo):
# Obtener el grado del polinomio
grado = len(modelo.coef_)

# Obtener Los parámetros aprendidos como una lista
w = [modelo.intercept_]
w += (modelo.coef_).tolist()
# Numpy tiene una función para imprimir polinomios de manera elegante
# (La usaremos, pero necesita los parámetros en orden inverso)
```

```
print ('Polinomio de grado ' + str(grado) + ':')
w.reverse()
print (np.poly1d(w))
```

Ahora implementaremos la validación cruzada k-fold. Definiremos una función que calcule los k errores de validación designando cada uno de los k segmentos como el conjunto de validación. Acepta como parámetros (i) k , (ii) penalidad_l2 , (iii) los datos de entrenamiento, (iv) la salida (v) lista de atributos. La función devuelve el error de validación medio usando k segmentos como conjuntos de validación.

- Para cada i en $[0, 1, \dots, k-1]$:
 - Calcular los índices de inicio y fin del segmento i y llamar 'inicio' y 'fin'
 - Crear el conjunto de validación tomando una porción (inicio: fin + 1) de los datos.
 - Crear el conjunto entrenamiento fijado añadiendo la rebanada (fin + 1: n) al final de la rebanada (0: inicio) .
 - Entrenar un modelo lineal usando un conjunto de entrenamiento recién formado, con una penalidad_l2 dada
 - Calcule el error de validación usando el conjunto de validación recién creado

```
In [6]: def validacion_cruzada_k_fold(algoritmo, k, penalidad_l2, data, salida, lista_atributos):
n = len(data)
rss = 0
mse = 0
modelo = algoritmo(alpha=penalidad_l2)
X = data[lista_atributos]
y = data[salida]
for i in range(k):
    inicio = (n*i)//k
    fin = (n*(i+1))//k-1
    X_validacion = X[inicio: fin+1]
    y_validacion = y[inicio: fin+1]
    X_entrenamiento = X[0:inicio]
    y_entrenamiento = y[0:inicio]
    ultimo_X = X[fin+1:n]
    ultimo_y = y[fin+1:n]
    X_entrenamiento = X_entrenamiento.append(ultimo_X)
    y_entrenamiento = y_entrenamiento.append(ultimo_y)
    modelo.fit(X_entrenamiento, y_entrenamiento)
    mse += mean_squared_error(y_validacion, modelo.predict(X_validacion))
mse = mse / k
return mse
```

Una vez que tenemos una función para calcular el error de validación promedio de un modelo, podemos escribir un lazo para encontrar el modelo que minimiza el error de validación promedio. Escriba un lazo que haga lo siguiente: Estaremos nuevamente ajustando un modelo polinomial de grado 15 usando la entrada sqft_living Con la penalidad_l2 en $[10^1, 10^{1.5}, 10^2, 10^{2.5}, \dots, 10^7]$ (para obtener esto en Python, puede utilizar esta función Numpy: `np.logspace(1, 7, Num = 13)`) Ejecutar validación cruzada de 10-fold con penalidad_l2 Informe qué penalización de L2 produjo el error de validación promedio más bajo.

```
In [7]: penalidad_l2 = np.logspace(0, 0.5, num=20)
mse = np.empty(len(penalidad_l2))
data = ventas_entre_val[atributos]
mis_atributos = list(data)
data['price'] = ventas_entre_val['price']
for i in range(len(penalidad_l2)):
```

```

        mse[i] = validacion_cruzada_k_fold(Ridge, 10, penalidad_l2[i], data, 'price', atributos)
print (mse)
print (min(mse))

```

```

[4.00385533e+10 4.00385683e+10 4.00385864e+10 4.00386080e+10
 4.00386337e+10 4.00386640e+10 4.00386996e+10 4.00387413e+10
 4.00387899e+10 4.00388464e+10 4.00389120e+10 4.00389877e+10
 4.00390750e+10 4.00391756e+10 4.00392911e+10 4.00394235e+10
 4.00395752e+10 4.00397485e+10 4.00399464e+10 4.00401720e+10]
40038553289.54385

```

Graficar el error de validación con respecto al valor de la penalización

```

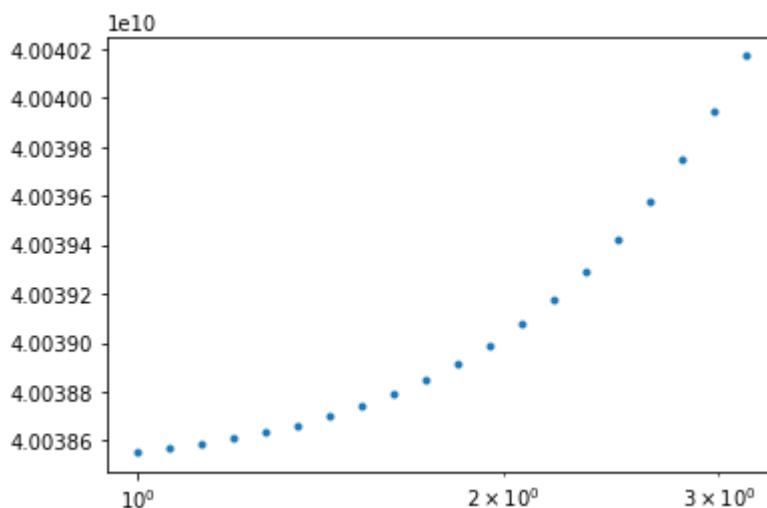
In [8]: # Graficar los valores de la penalizacion_l2 en el eje xy el error de validación cruzada en
# El uso de plt.xscale ('log') hará que su trazado sea más intuitivo.
alfa = penalidad_l2[np.argmin(mse)]
print('alfa = ', alfa )
print('MSE =', mse[np.argmin(mse)])
plt.plot(penalidad_l2, mse, '.')
plt.xscale('log')

```

```

alfa = 1.0
MSE = 40038553289.54385

```



Una vez que encuentre el mejor valor para la penalización L2 mediante la validación cruzada, es importante reentrenar un modelo final con todos los datos de entrenamiento usando este valor de penalizacion_l2 . De esta manera, su modelo final será entrenado con el conjunto de datos completo.

```

In [9]: modeloRidge = Ridge(alpha=alfa)
X = ventas_entre_val[atributos]
y = ventas_entre_val['price']
X_prueba = ventas_prueba[atributos]
y_prueba = ventas_prueba['price']
modeloRidge.fit(X, y)
mse = mean_squared_error(y_prueba, modeloRidge.predict(X_prueba))
print('MSE =', mse )
imprimir_coeficientes(modeloRidge)

```

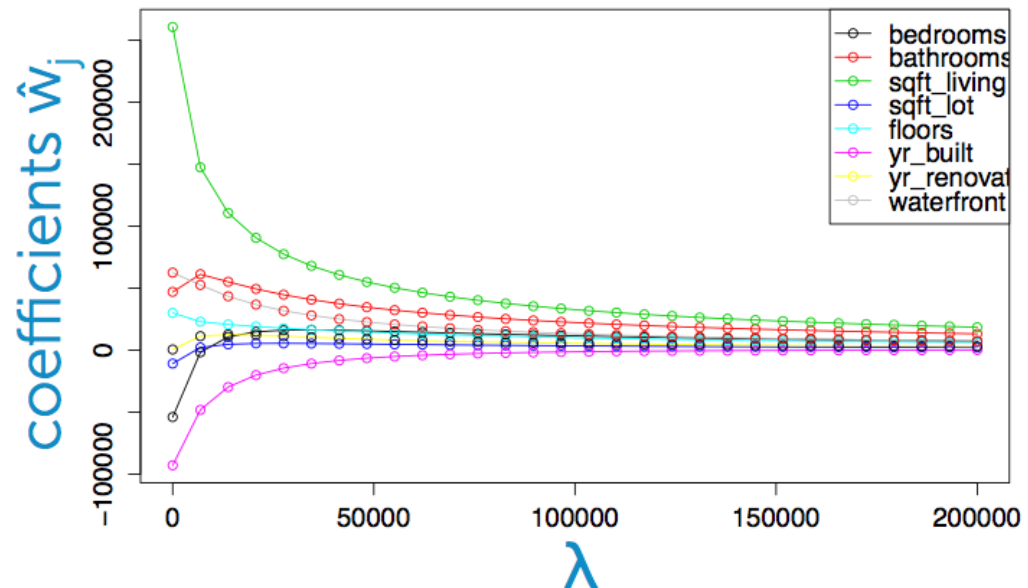
MSE = 46439082793.87048

Polinomio de grado 18:

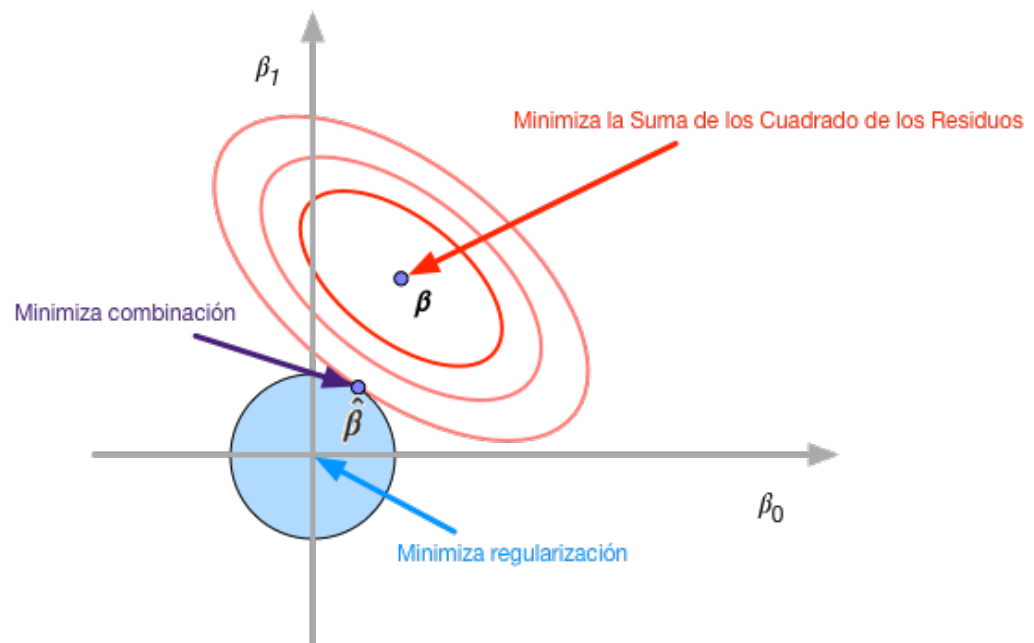
$$\begin{aligned} & -0.347 x^{18} + 22.35 x^{17} - 2.07e+05 x^{16} + 5.941e+05 x^{15} - 583.2 x^{14} + 18.6 x^{13} \\ & - 2668 x^{12} + 43.39 x^{11} + 66.72 x^{10} + 9.721e+04 x^9 + 2.535e+04 x^8 \end{aligned}$$

Valor de los coeficientes con respecto al valor de la penalización. Regresión Lineal Ridge

Los valores de los coeficientes tienden a cero a medida que se incrementa el valor de la penalización (alfa)



Contornos de la función de penalización y el error para la Regresión Lineal Ridge



```
In [10]: modeloLR = LinearRegression()
modeloLR.fit(X, y)
mse = mean_squared_error(y_prueba, modeloLR.predict(X_prueba))
print('MSE =', mse)
imprimir_coeficientes(modeloLR)
```

MSE = 46419376645.531944

Polinomio de grado 18:

$$\begin{aligned} & -0.3464 x^{18} + 22.38 x^{17} - 2.08e+05 x^{16} + 5.961e+05 x^{15} - 585.4 x^{14} \\ & + 18.59 x^{13} - 2664 x^{12} + 43.33 x^{11} + 66.78 x^{10} + 9.714e+04 x^9 \\ & + 2.527e+04 x^8 + 5.4e+04 x^7 + 5.94e+05 x^6 + 9.55e x^5 + 0.1074 x^4 + 110.1 x^3 \end{aligned}$$

Selección de Atributos Usando Regresión Lasso

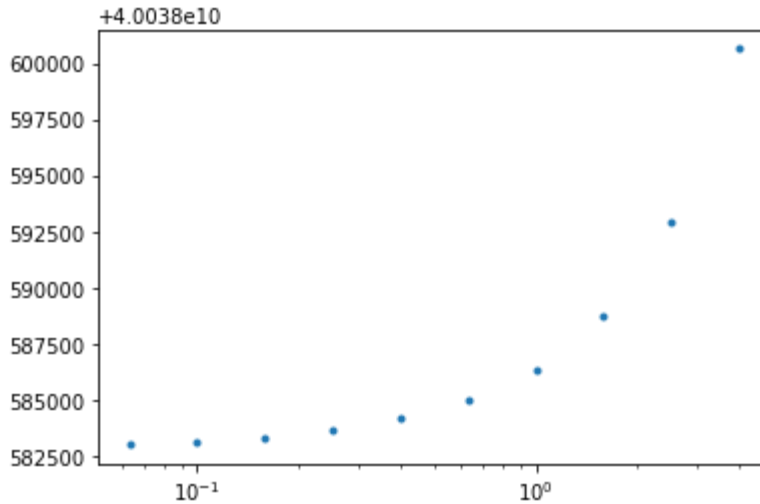
```
In [11]: def seleccion_atributos(k, penalidad_l1, data, salida, lista_atributos):
n = len(data)
mse = 0
modelo = Lasso(alpha=penalidad_l1)
X = data[lista_atributos]
y = data[salida]
for i in range(k):
    inicio = (n*i)//k
    fin = (n*(i+1))//k-1
    X_validacion = X[inicio: fin+1]
    y_validacion = y[inicio: fin+1]
    X_entrenamiento = X[0:inicio]
    y_entrenamiento = y[0:inicio]
    ultimo_X = X[fin+1:n]
    ultimo_y = y[fin+1:n]
    X_entrenamiento = X_entrenamiento.append(ultimo_X)
    y_entrenamiento = y_entrenamiento.append(ultimo_y)
    modelo.fit(X_entrenamiento, y_entrenamiento)
    mse += mean_squared_error(y_validacion, modelo.predict(X_validacion))
mse = mse / k
return mse
```

```
In [12]: penalidad_l1 = np.logspace(-1.2, 0.6, num=10)
mse = np.empty(len(penalidad_l1))
data = ventas_entre_val[atributos]
data['price'] = ventas_entre_val['price']
for i in range(len(penalidad_l1)):
    mse[i] = validacion_cruzada_k_fold(Lasso, 10, penalidad_l1[i], data, 'price', atributos)
print (mse)
```

```
[4.00385831e+10 4.00385832e+10 4.00385834e+10 4.00385837e+10
4.00385842e+10 4.00385850e+10 4.00385864e+10 4.00385887e+10
4.00385929e+10 4.00386006e+10]
```

```
In [13]: # Graficar los valores de la penalización_l1 en el eje xy el error de validación cruzada en el eje y
# El uso de plt.xscale('log') hará que su trazado sea más intuitivo.
alfa = penalidad_l1[np.argmax(mse)]
print('alfa = ', alfa)
print('MSE =', mse[np.argmin(mse)])
plt.plot(penalidad_l1, mse, '.')
plt.xscale('log')
```

```
alfa = 3.9810717055349722
MSE = 40038583065.284485
```



```
In [14]: modeloLasso = Lasso(alpha=5000000)
modeloLasso.fit(X, y)
mse = mean_squared_error(y_prueba, modeloLasso.predict(X_prueba))
print('MSE =', mse)
imprimir_coeficientes(modeloLasso)
```

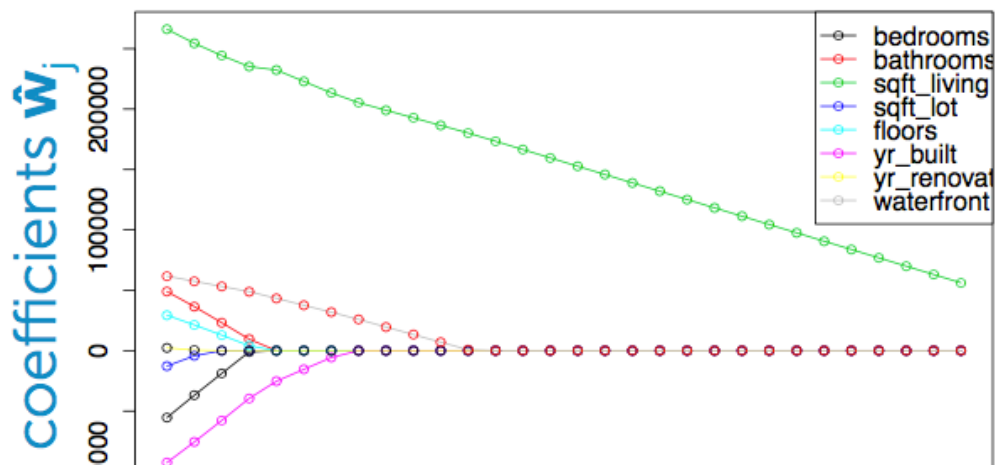
```
MSE = 75216765537.79742
```

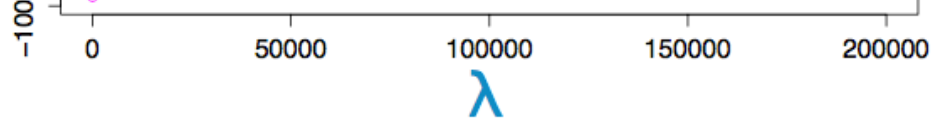
```
Polinomio de grado 18:
```

```
18      17      16      13      12      11      10
-0.7416 x + 66.24 x - 0 x + 53.57 x - 0 x + 11.8 x - 0 x
      4      3
+ 0.05056 x + 236.8 x - 0 x - 8.38e+04
```

Valor de los coeficientes con respecto al valor de la penalización. Regresión Lineal Lasso

Los valores de los coeficientes se van haciendo igual a cero a medida que se incrementa el valor de la penalización (alfa)





Contornos de la función de penalización y el error para la Regresión Lineal Lasso

