Ridge

¡Comencemos!

Competencias

- Conocer la mecánica de regularización en los métodos Ridge.
- Utilizar los métodos de regularización para resolver problemas de dimensionalidad y mejora de desempeño predictivo.
- Conocer las normas L2.

Introducción

En esta sección conocerás la mecánica de regularización en los métodos Ridge. Además

¡Vamos con todo!

¿Por qué debo regularizar?

Antes de estudiar las principales normas de regularización, es bueno definir una serie de justificaciones sobre el uso de regularizadores en los modelos:

- En ciertas situaciones, puede darse que existan parámetros que tengan un peso exagerado en el proceso de entrenamiento.
- Este peso exagerado de parámetros conlleva a mayores chances de overfit en el conjunto de entrenamiento, dado que el modelo asignará una importancia desmedida a un atributo en específico.
- También hay razones computacionales: En la medida que agregamos más parámetros, hacemos más costosa de estimar nuestra ecuación. Cabe destacar que la regularización puede solucionar el problema de la dimensionalidad mediante la eliminación de atributos, pero no en la reducción de éstos. De esta manera, también es un buen proxy para evaluar la estabilidad e idoneidad de éstos en el proceso de predicción.
- Regularizar permite realizar una evaluación diagnóstica de los parámetros inferidos, dependiendo de elementos estrictamente ajenos a los producidos por el modelo.

Ridge: Norma L2

La primera forma de crear modelos de regresión sesgados es agregando una penalización a la suma de los cuadrados, Ridge penaliza esta suma agregando el término

$$\lambda \sum_{j=1}^{p} \beta_j^2$$

Luego, nuestra función de minimización queda de la siguiente forma:

$$\beta_{\text{Ridge}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cuando nuestro modelo lineal sobre-ajusta los datos de entrenamiento algunos de los coeficientes de la regresión se inflan, provocando que tengan valores altos en comparación con los demás. Al penalizar la función objetivo de la forma anterior, el aumentar el valor de alguno de los coeficientes se traduce en una penalización a la función objetivo, luego, solo se permitirá el incremento de un coeficiente cuando se produce una reducción proporcional en la suma de los cuadrados.

- Ridge también se conoce como **regularización en norma L2** porque el término que se agrega en la función objetivo en realidad es la norma L2 (o norma euclídea) de un vector cuyo largo es la magnitud del coeficiente correspondiente.

¿Y de dónde sacamos ese λ que aparece multiplicando a la suma?

Para entender cómo encontrar este valor debemos entender primero cuál es su rol en el término penalizador. Si nos fijamos, λ multiplica cada coeficiente por igual, por lo tanto, dice cuánto se está penalizando el modelo frente a un determinado cambio en un coeficiente. Mientras más alto sea λ el modelo recibirá un mayor "castigo" cuando intente aumentar un coeficiente. Este es un valor que tendremos que escoger nosotros al momento de definir el modelo con regularización.

Digresión: Cotas asintóticas en funciones objetivo:

Una observación interesante que debemos tener en cuenta es que si $\lambda = 0$ en la función objetivo de Ridge, lo que obtenemos como resultado es una regresión lineal ordinaria, pues el término completo se hace igual a **0**. De esta observación se infiere que la función objetivo de Ridge es una cota superior a la función objetivo de la regresión mediante mínimos cuadrados. Lo que buscamos mediante Ridge es la minimización de una cota de la función de OLS. Este comportamiento se repite en varios modelos de máquinas de aprendizaje, donde la minimización/maximización de una cota de la función objetivo resulta mejor que buscar la solución para la función completa.

Un comportamiento interesante de la regularización en norma L2 es que luego de cierta cantidad de penalización añadida en λ los valores de algunos parámetros se acercan a 0, suprimiendolos del modelo final. De esta forma, la elección de λ es un elemento a considerar.

Ejemplo: Prediciendo valores de inmuebles en Boston

Veamos en la práctica en que se traduce esta regularización analizando los valores de los coeficientes. Para esto, utilizaremos un dataset que contiene distintos atributos de casas en Boston. Nuestro objetivo será predecir el valor de la casa de acuerdo a aspectos como el número de baños que posee, la cantidad de pisos, etc. Nuestro vector objetivo es la variable price.

Comencemos por la inclusión de las librerías clásicas (`numpy`, `pandas` y `matplotlib.pyplot`) en nuestro ambiente de trabajo. Importaremos el archivo `kc_house_archive`. Posteriormente eliminaremos columnas que hacen referencia a información referencial como el zipcode, el identificador de la observación (`id`) y la fecha (`date`).

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
plt.rcParams['figure.figsize'] = (10, 6)
import cv_error as gfx
```

La base de datos se compone de 21.613 observaciones con 18 atributos. Mediante `df.info()` podemos extraer información sobre la naturaleza del dato, así como la existencia de valores perdidos. Observamos que los datos no presentan cadenas, lo que facilita el preprocesamiento en `sklearn`.

```
df = pd.read_csv('kc_house_data.csv')
# Vamos a eliminar ciertas columnas que son irrelevantes para nuestro
analisis
df.drop(['zipcode', 'id', 'date'], axis = 1, inplace = True)

print('Numero de filas: {0}'.format(df.shape[0]))
print('Numero de columnas: {0}'.format(df.shape[1]))
```

```
Numero de filas: 21613
Numero de columnas: 18
```

```
# solicitamos información sobre los atributos
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 18 columns):
price           21613 non-null float64
bedrooms         21613 non-null int64
bathrooms       21613 non-null float64
sqft_living     21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null int64
view             21613 non-null int64
condition       21613 non-null int64
grade           21613 non-null int64
sqft_above       21613 non-null int64
sqft_basement   21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
lat             21613 non-null float64
long             21613 non-null float64
sqft_living15   21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(5), int64(13)
memory usage: 3.0 MB
```

No hay datos nulos, por lo que podemos comenzar a pensar en aplicar nuestro modelo con un poco más de tranquilidad. Partamos por inspeccionar nuestro vector objetivo price. Para ello generaremos un histograma con `seaborn`. En el lado izquierdo se presenta el histograma de la variable sin modificar, se observa un fuerte sesgo hacia valores bajos que puede afectar el desempeño del modelo.

Para ello vamos a generar el siguiente flujo de preprocesamiento:

1. Dado que los métodos de regularización son sensibles a la escalas de las variables, generaremos una versión estandarizada de la base de datos con `StandardScaler`.

2. Dado que nuestro vector objetivo tiene un fuerte sesgo, lo transformaremos mediante el logaritmo. El gráfico de la derecha muestra que el comportamiento empírico de la variable se asemeja más a una distribución normal.

```
from sklearn.preprocessing import StandardScaler

# Normalizamos la variable precio
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
df_scaled['price'] = np.log(df['price'])
```

```
fig , ax = plt.subplots(1,2)
sns.distplot(df['price'], bins='fd',ax = ax[0])\
  .set_title('Histograma Precio', size = 14)
sns.distplot(df_scaled['price'], bins = 'fd', ax = ax[1])\
  .set_title('Histograma logaritmo del Precio', size = 14);
```

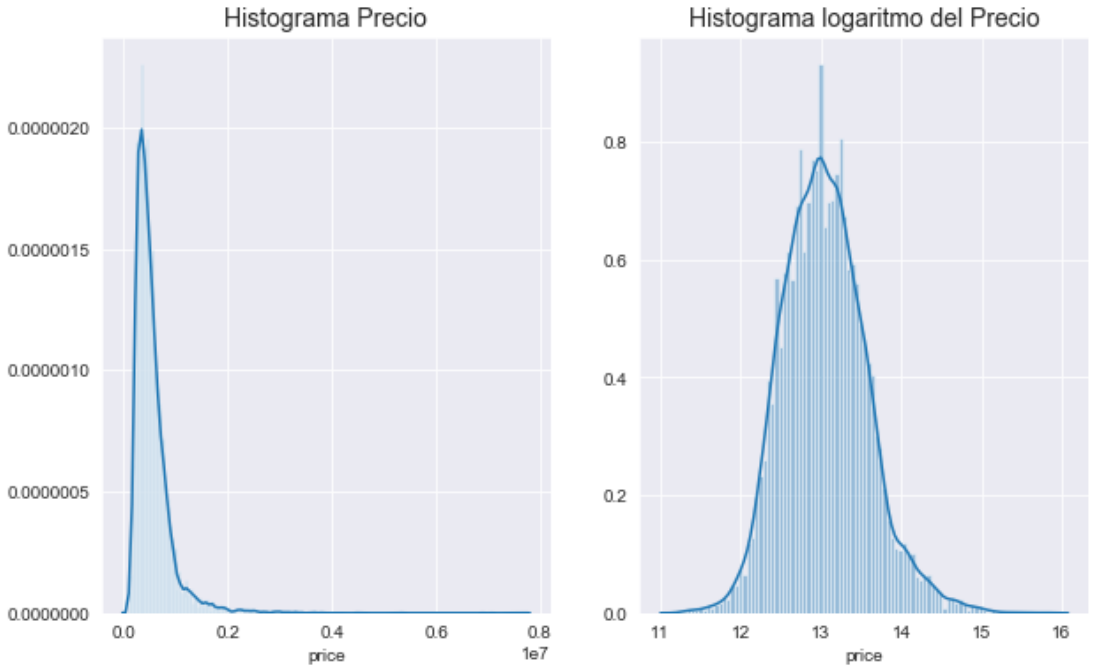

Imagen 1. Histogramas Precio y Log(Precio).
Fuente: Desafío Latam.

Normalización de datos

Escalar una variable permite ajustar su rango de valores a un rango acotado predefinido, este tipo de preprocesamiento se utiliza cuando el modelo que se tiene es sensible a magnitudes, en nuestro caso, la variable objetivo para la regresión es price, la cual presenta rangos de valores en magnitudes demasiado distintas en comparación a los atributos con los que alimentaremos el modelo, como el número de baños (rango 0-8), el número de habitaciones (rango 0-33), etc. Si no se escala la variable, se generará un modelo en el cual, al aumentar en un pequeño diferencial el atributo (feature), la variable objetivo de la predicción experimentará un aumento considerable.

Es interesante notar que algunos algoritmos pueden mejorar considerablemente su rapidez de convergencia con datos normalizados (máquinas de soporte vectorial, por ejemplo). En nuestro caso, puesto que se realizará una regresión lineal de mínimos cuadrados, la normalización no tiene este tipo de efecto pues no varía la correlación de los coeficientes al ser términos lineales. Podemos ver como la dispersión de los valores de la variable se reduce notablemente.

```
plt.subplot(1, 2, 1)
sns.boxplot(df['price'], orient='v').set_title('Boxplot Precios',
size=14);
plt.subplot(1, 2, 2)
sns.boxplot(df_scaled['price'], orient='v').set_title('Boxplot
log-Precios', size=14)
```

```
Text(0.5, 1.0, 'Boxplot log-Precios')
```

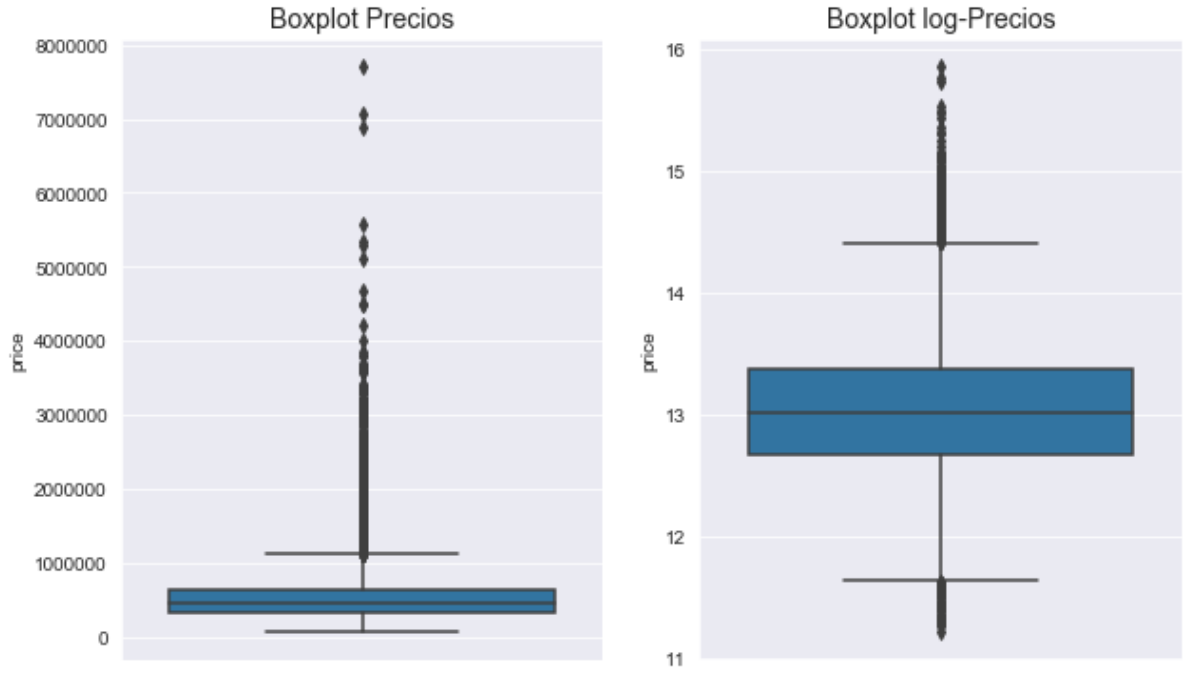

Imagen 2. Boxplot Precios y Log(Precios).
Fuente: Desafío Latam.

La normalización de variables en un dataset se utiliza, por lo general, cuando necesitamos aplicar un proceso que asume normalidad en la variable, o un escalamiento para acotar los valores de ciertas variables a un cierto rango tratable y comparable con los de otras variables, en nuestro caso, el modelo asume que el error al de cada estimador de los coeficientes con respecto a su valor paramétrico es normal. Podemos observar los histogramas de la variable precio antes y después de normalizarla:

Antes de entrenar cualquier modelo, necesitamos dividir el dataset en al menos dos conjuntos: **Entrenamiento** y **Validación**, esto lo podemos realizar fácilmente con la función `train_test_split()` de la librería `sklearn`, utilizaremos un `test-size` de 30% de la data, el resto será dedicada a entrenamiento:

```
from sklearn.model_selection import train_test_split

# X será nuestro conjunto de atributos. y será nuestra variable objetivo
X = df_scaled.iloc[:, 1:] # Tomamos todas las columnas menos la primera
(price)
N = X.shape[0] # guardamos el número de filas (datos de entrenamiento)
y = df_scaled['price'] # asignamos como target la variable 'price'
#Separamos los subsets de test y train
X_train, X_test, y_train, y_test, = train_test_split(X, y, test_size =
0.3, random_state = 63)
```