TEMA 4

MÓDULO:
TÉCNICAS AVANZADAS DE PREDICCIÓN

TÉCNICAS PARA MEJORAR UN MODELO

Institut de Formació Contínua-IL3
UNIVERSITAT DE BARCELONA

ÍNDICE

OBJETIVOS ESPECÍFICOS

- Conocer cuándo recurrir a técnicas de validación cruzada o de reducción de dimensionalidad.
- Distinguir entre técnicas stepwise y técnicas de regularización para optimizar el modelo.
- Aprender en qué medida nos ayudan las técnicas de remuestreo a darle validez a nuestra modelización.

1. TÉCNICAS PARA MEJORAR UN MODELO

Empezamos de la misma forma en que acabamos el tema anterior ¿Debemos tener en cuenta más consideraciones para poder decir que nuestro modelo es suficientemente robusto y entonces podemos confiar en él para ponerlo en producción?

Cuando hemos generado nuestro modelo, hemos calculado las betas con una base de datos y hemos visto el grado de ajuste con esa misma base de datos. ¿No podríamos coger otra base de datos para comprobar el ajuste?

1.1 MÉTODOS DE REMUESTREO

El remuestreo, en regresión, es una técnica que consiste en:

- Sacar muestras de nuestra base de datos original.
- Estimar un modelo (igual que lo hemos hecho hasta ahora) en base a esa submuestra.
- Ver diferencias entre los parámetros estimados de cada una de las submuestras.
- Probar nuestras estimaciones en submuestras con las que no ha sido estimado el modelo, para valorar el poder de predicción evitando sobreajustes en los modelos.

Esta técnica no supone nada adicional al conocimiento teórico que traemos sobre estimación de modelos lineales generales. Sino que es una técnica que se basa en remuestreo (bootstrap), con la que conseguiremos obtener conclusiones adicionales a las obtenidas con la teoría estudiada hasta el momento. Con una base de datos con la que hasta ahora construíamos un único modelo, ahora vamos a construir múltiples modelos basados en submuestras de la base de datos principal. Con ellos, vamos a dotar de robustez, independencia y profundidad a nuestro análisis.

IMPORTANTE

La estimación y suma residual no van a provenir de los mismos datos, sino de muestras independientes y, aleatoriamente, escogidas.

Si eres el responsable de los resultados de un modelo, quizás quieras tomar todas las precauciones posibles antes de poner en funcionamiento este tipo de métodos.

PIENSA UN MINUTO

¿Probarías solamente tu modelo con los datos con los que, a su vez, estás calibrándolo?

SABÍAS QUE...

Se llamó "boostrap" a este tipo de técnicas dado su significado imaginario en el que, estando en una situación difícil, (atrapados) vamos a poder salir del contratiempo, elevándonos en el aire al tirar muy fuerte de las cuerdas de nuestros zapatos.

La filosofía es que, con pocos recursos, vamos a poder salir de problema con lo poco que llevamos encima.

¿Pero para qué quiero complicarme más la vida?

Para dar mayor robustez al proceso y, por lo tanto, mayor credibilidad. Estas técnicas nos van a ayudar a ver variabilidad en las betas y, también, nos van a ayudar a conocer el verdadero poder de predicción.

Training - Test

La primera estrategia que vamos a seguir es la más sencilla. Nuestra base de datos original, vamos a dividirla en dos partes.

La primera parte va a ir destinada a entrenar nuestro modelo, de tal forma que los coeficientes estimados van a estar calculados en base a este subset de nuestra base de datos de origen. Vamos a validar estos coeficientes estimados, con la otra parte de la base de datos que nos habíamos reservado en un primer momento.

Un ejemplo lo podemos desarrollar con la base de datos de sensores que habíamos estado utilizando hasta ahora.

Recordemos que teníamos una base de datos con el detalle del acelerómetro por cada milisegundo en el que se almacenaba la información de una acción cotidiana.

Nuestra misión era la de identificar aquellos patrones en los que los datos venían de una persona que se había caído frente al resto de patrones de acciones cotidianas.

Lo primero que vamos a hacer es realizar una submuestra y realizar nuestro modelo de entrenamiento (training):

```
```{r warning=FALSE,message=FALSE}
muestra<-0.7
sample<-sample(x=c(1:nrow(df)),size = muestra*nrow(df))
modelo_probit<-glm(response~.,data=df[sample,],family=binomial(link="probit"))
```
```

A continuación, vamos a comprobar el ajuste AUC que veíamos, anteriormente, sobre la base de datos de entrenamiento y sobre la base de datos de test (que es la complementaria a la de entrenamiento):

```
```{r results='asis', size="small"}
suppressMessages(auc(df$response[sample],predict(modelo_probit,df[sample,],type="response")))
suppressMessages(auc(df$response[-sample],predict(modelo_probit,df[-sample,],type="response")))
```
```

```
Area under the curve: 0.8678
Area under the curve: 0.5246
```

Podemos comprobar la diferencia importantísima entre ambos resultados. **¿Qué significa esto?** que la cifra que nos había dado, anteriormente, de grado de ajuste estaba muy viciada por los datos que está-

bamos utilizando. Nuestro modelo realizado se ajustará a los datos reales mejor, como indica la medida sobre la base de datos de test, que la de entrenamiento.

¿Con esto nos aseguramos que las muestras son independientes y que el error de la parte test sea más realista?

Con esto lo mejoramos bastante, sin embargo, como ya podéis imaginar, dependiendo de la muestra aleatoriamente escogida tendremos unos resultados u otros. Por lo tanto, aunque es una técnica que se utiliza muchísimo en la práctica, es cierto que se puede mejorar si queremos ser aún más precisos.

K Fold Validation

Lo primero que se nos ocurre, y es algo que no es muy difícil de programar, es hacer múltiples particiones en lugar de una partición. La base de datos la vamos a dividir en K partes. Estas K partes las va a decidir el analista dependiendo del consumo computacional, recursos e incertidumbre sobre los datos que tenga.

A partir de aquí, lo que vamos a hacer es realizar K iteraciones sobre modelización y comprobación de la modelización. En la primera iteración, estimaremos el modelo con K-1 partes y validaremos con la parte restante que no hemos utilizado para calcular el modelo. Así lo iremos haciendo sucesivamente hasta completar las K partes decididas inicialmente. Al finalizar el proceso, podremos ver tanto la variabilidad del ajuste como la media del mismo. Con este proceso, nos aseguramos mayor poder y seguridad sobre las estimaciones que estamos realizando.

Vamos a hacer esto mismo con la base de datos de los sensores:

```r
```{r warning=FALSE,message=FALSE}
division<-4
df$cluster<-sample(x = c(1:division),size = nrow(df),replace = T)

auc<-c(0)
for (i in 1:division){
  modelo_probit<-glm(response~.,data=df[df$cluster!=i,],family=binomial(link="probit"))
  auc[i]<-suppressMessages(auc(df$response[df$cluster==i],predict(modelo_probit,df[df$cluster==i,],type="response")
))
}
print(auc)
mean(auc)
```
```

```
[1] 0.5336842 0.7505952 0.7205882 0.5143411
[1] 0.6298022
```

En este caso, he decidido partir en cuatro partes la base de datos y sacar el indicador del AUC.

Como podemos ver, cada una de las particiones nos lleva a resultados bien distintos, lo cual hace que tenga sentido poder utilizar la media de los mismos para poder hacer una correcta estimación del poder de predicción del modelo.

Si este proceso lo repetimos 20 veces, veremos una nueva distribución del poder de predicción mucho más centrada y acertada:

```r
```{r warning=FALSE,message=FALSE}

division<-4
veces<-20

medias<-c(0)
for (x in 1:veces){
df$cluster<-sample(x = c(1:division),size = nrow(df),replace = T)
auc<-c(0)
for (i in 1:division){
  modelo_probit<-glm(response~.,data=df[df$cluster!=i,],family=binomial(link="probit"))
  auc[i]<-suppressMessages(auc(df$response[df$cluster==i],predict(modelo_probit,df[df$cluster==i,],
type="response")))
}

medias[x]<-mean(auc)
}

quantile(medias)
```
```
       0%       25%       50%       75%      100% 
0.5781677 0.6196201 0.6354345 0.6546742 0.6788394 
```

En el histograma podemos ver la realidad del poder de predicción de nuestro algoritmo. Podemos asumir que la esperanza matemática de la ROC va a ser el percentil 50 de los cuantiles reflejados arriba y que, en ningún caso, vamos a esperar resultados mejores que percentiles 90-100.

¡Vaya! con lo bien que nos había quedado la modelización de las caídas.

En realidad, este ajuste tan mediocre lo hemos obtenido porque solamente estamos utilizando un tipo de sensor (acelerómetro) y una localización de dicho sensor (muñeca). Si ampliamos el número de sensores utilizados, el ajuste aumenta significativamente.

¿Podemos hacer algo más con el remuestreo?

Realmente, tenemos múltiples posibilidades:

- Comprobar el ajuste real de nuestro modelo frente a muestras independientes.
- Elección de variables o grado del splines dado el error medio del remuestreo k-fold.
- Otra opción que tenemos es la de testar las betas de nuestro modelo.

Contamos con una cuarta base de datos con la que vamos a testar las betas de un modelo de precio del alquiler de viviendas.

ACTIVIDAD

PRECIO ALQUILER

Contamos con los datos procedentes de Airbnb.

La descarga son de viviendas disponibles en Madrid (Bº Retiro) en 10/2020:

- Longitud y latitud del piso.
```

- Precio del piso por día y logaritmo del precio.
- Room_Type: tipo de habitación. Todos los tipos de vivienda en esta base de datos son únicos. Son viviendas familiares.
- Minimum_nights: mínimo número de noches requerido.
- Number_of_reviews: revisiones que tiene el piso y scoring de las revisiones.
- Bedrooms: nº de habitaciones con las que cuenta.
- Reviews_per_month: nº de revisiones por mes.
- Availability: disponibilidad los últimos meses y si tiene disponibilidad inmediata.
- Beds: nº de camas.
- Accommodates: nº de acomodaciones permitidas.
- Tv_ports : puertos de cable para conectar la televisión.
- Ph_ports: puertos de cable para conectar el teléfono.
- Nº de vecinos y piso de la vivienda.
- Ventanas: número de ventanas en la casa.

```{r results='asis', size="small"}
pisos<-read.csv("./Data/table_5.05.csv",sep=",")[,-1]
colnames(pisos)
```

```
 [1] "longitude"                  "latitude"                   "price"
 [4] "room_type"                  "minimum_nights"             "number_of_reviews"
 [7] "review_scores_value"        "calculated_host_listings_count" "bedrooms"
[10] "reviews_per_month"          "beds"                       "accommodates"
[13] "availability_30"            "availability_60"            "availability_90"
[16] "instant_bookable"           "Distancia_Centro"           "Distancia_Norte"
[19] "Distancia_Sur"              "logprice"                   "tv_ports"
[22] "phone_ports"                "Vecinos"                    "Piso"
[25] "ventanas"
```

Objetivo: El objetivo que nos marcan es el de realizar una modelización para poder asesorar el precio que se puede/debería cobrar por cada alojamiento, dadas unas variables definidas explicativas del precio.

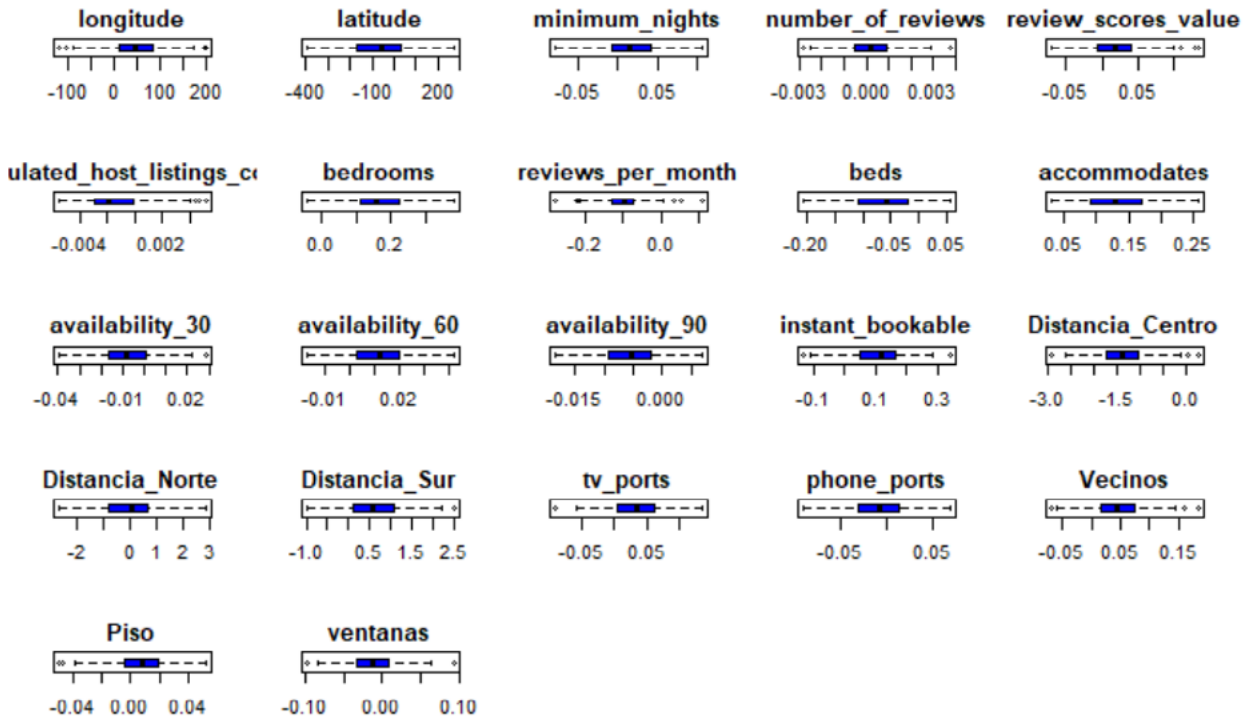
Con esta base de datos, vamos a realizar una estimación de la distribución de las betas de cada variable explicativa en base a submuestras de nuestra base de datos central:

```{r results='asis', warning=FALSE,message=FALSE}
pisos<-dplyr::select(pisos,-price,-room_type)
matriz<-as.data.frame(matrix(0,nrow=200,ncol=ncol(pisos)))

for (i in 1:200){
  rand<-sample(c(1:nrow(pisos)),size=0.5*nrow(pisos),replace = FALSE)
  modelopisos<-glm(logprice~.,data=pisos[rand,])
  matriz[i,]<-as.vector(modelopisos$coefficients)
}

colname<-c("intercep",colnames(pisos))
colname <- colname[!colname %in% "logprice"]
colnames(matriz)<-colname
matriz<-matriz[,-1]

p<-suppressMessages(diag_cajas(matriz,filas = floor(ncol(pisos)/5)+1,columnas=5))
```



La idea que perseguimos con el remuestreo, en este caso, es la de observar la variabilidad de las betas ante sets de datos diferentes aleatoriamente extraídos.

Las betas que tengan poca dispersión serán aquellas más insesgadas al muestreo establecido. Cuidado con las betas que tengan valores positivos y negativos dependiendo de la muestra, esto quiere decir que nos debemos fiar poco de dicha variable explicativa o que hay efectos no lineales u opuestos dentro de ella.

Una vez que hemos conocido las técnicas de remuestreo... **Cuando hemos realizado el modelo, hemos tomado unas variables como dadas ¿le podemos meter al modelo algún tipo de parámetro que penalice cada variable que metemos de más?**

1.2 MÉTODOS DE REGULARIZACIÓN

Los métodos de regularización son extensiones del modelo lineal que lo que pretenden, básicamente, es una reducción de la dimensionalidad en la base de datos, utilizando técnicas que permitan establecer una penalización sobre cada variable añadida en la misma fórmula que vamos a estimar.

En el apartado dedicado a la regresión lineal vimos:

$$Y = \beta_1 X_1 + \cdots + \beta_p X_p + e$$

$$\hat{\beta} \sim N(\beta, \sigma(X^t X)^{-1})$$

El error medio de cada beta lo podemos definir como:

$$ErrorMedioBeta = E((\hat{\beta} - \beta)^2) = (E(\hat{\beta}) - \beta)^2 + Var(\hat{\beta})$$

El primer componente es el sesgo al cuadrado y la segunda parte es la varianza.

Los métodos que propone la regularización se basan en tratar de incrementar el sesgo reduciendo significativamente la varianza. Al final, el error disminuye si la reducción de la varianza es mayor que el incremento del sesgo. La manera en que lo hace es simplemente, incluyendo en la suma residual, una penalización, la cual como hemos visto en apartados previos, la vamos a optimizar:

$$Ridge === \mathrm{SR}(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{SR}(\beta) + \lambda \parallel \beta_{-1} \parallel_2^2$$

$$Lasso === \mathrm{SR}(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| = \mathrm{SR}(\beta) + \lambda \parallel \beta_{-1} \parallel_1$$

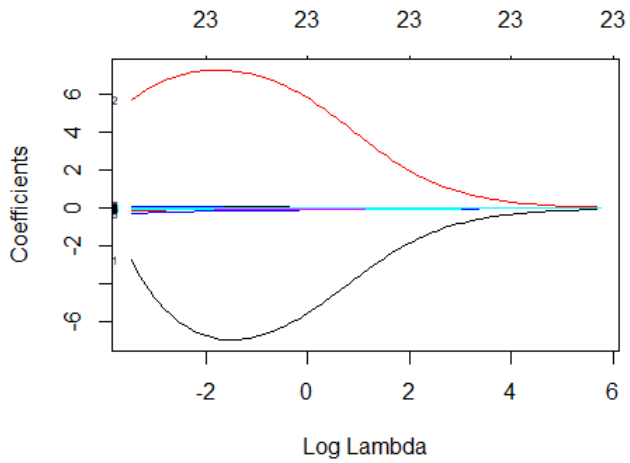
REGRESIÓN RIDGE

Estamos introduciendo el parámetro lambda en la regresión. Cuando lambda vale cero, entonces la fórmula quedaría similar a la regresión lineal vista en apartados anteriores. Teóricamente, vamos a incrementar el sesgo de los estimadores, sin embargo, vamos a reducir la varianza. El problema, por lo tanto, que surge es el de ver qué valor lambda disminuye y minimiza el error.

Nos vamos a adentrar en el paquete de R "glmnet" que nos va a hacer la estimación Ridge para cada valor posible de lambda. Vamos a seguir utilizando la base de datos que hemos utilizado, anteriormente, del precio del piso de alquiler. Es una base de datos óptima para este tipo de ejercicios, puesto que tenemos relativamente pocos datos y bastantes variables explicativas. Para evitar el overfitting, deberíamos introducir algún tipo de ajuste que penalizase la introducción de variables en el modelo:

```
```{r results='asis', size="small",warning=FALSE,message=FALSE}
model <- model.matrix(~.-1-logprice, data = pisos)
response<-pisos$logprice
Ridge <- glmnet(x = model, y = response, alpha = 0)

plot(Ridge, label = TRUE, xvar = "lambda")
```
```



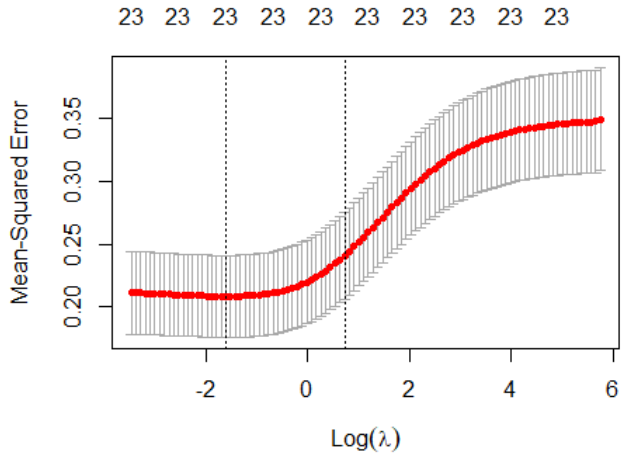
En esta figura, tenemos el valor de las betas por cada valor que va tomando Lambda. Como hemos dicho, para lambda=0, entonces el valor de las betas se corresponde con el valor medio que habíamos calculado en el ejercicio anterior de remuestreo. A medida que lambda empieza a aumentar, la beta va reduciéndose y tendiendo a cero, puesto que la penalización es tan grande que la introducción de las variables carece de sentido cuando lambda es muy grande.

Nuestro problema es ¿qué lambda escogemos?

La elección de lambda se suele hacer con la validación cruzada que acabamos de ver. Vamos a ir midiendo, por cada set de datos y lambdas, el error medio que obtenemos y nos vamos a quedar con el valor de lambda que minimiza dicho error medio:

```
```{r results='asis', size="small",warning=FALSE,message=FALSE}
Ridge2 <- cv.glmnet(x = model, y = response, alpha = 0, nfolds = 100)

plot(Ridge2)
```
```



Podemos ver el valor óptimo de la lambda en el cual minimizaríamos el error medio en el modelo. Nuestro modelo final lo podemos calcular con la lambda mín. ima obtenido:

```
```{r results='asis', size="small",warning=FALSE,message=FALSE}
pander(as.data.frame(as.matrix(predict(Ridge, type = "coefficients", s = Ridge2$lambda.min))),
split.cell = 80, split.table = Inf)
```
```

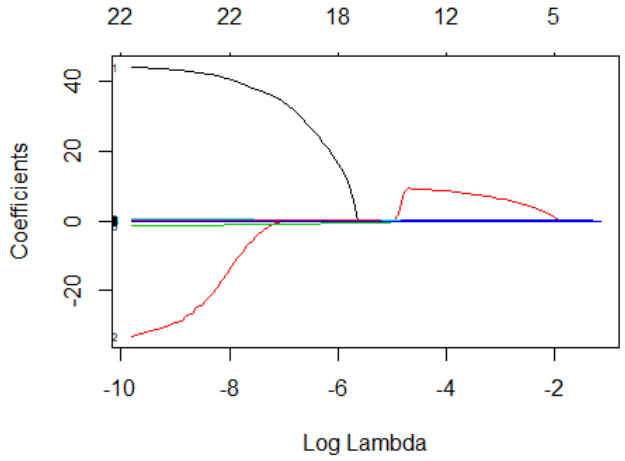
(Intercept)	-315.5
longitude	-6.925
latitude	7.292
minimum_nights	0.0156
number_of_reviews	-0.0007129
review_scores_value	0.0116
calculated_host_listings_count	-0.0004492
bedrooms	0.09521
reviews_per_month	-0.05968
beds	0.01497
accommodates	0.06535
availability_30	-0.0003108
availability_60	-8.023e-06
availability_90	0.0004597
instant_bookablef	-0.04021
instant_bookablet	0.04017
Distancia_Centro	-0.1274
Distancia_Norte	-0.0593
Distancia_Sur	-0.01049
tv_ports	0.02193
phone_ports	0.005408
Vecinos	0.02149
Piso	0.006463
ventanas	0.01816

Estas serían nuestras betas finales optimizadas para este problema de regularización.

REGRESIÓN LASSO

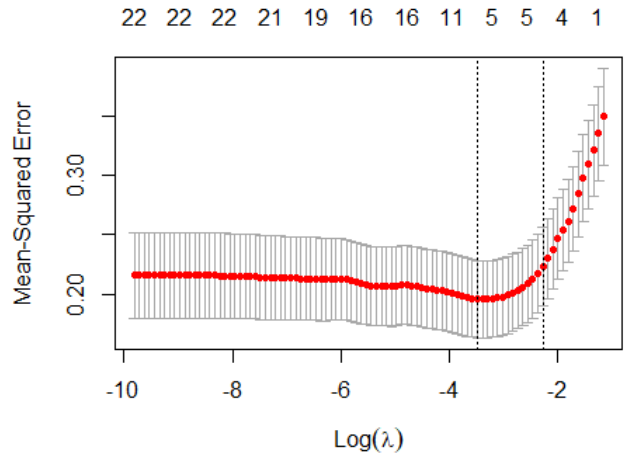
La principal diferencia con Lasso es el tipo de penalización utilizado. Para resolver el problema, lo que hacemos es llevar los coeficientes a cero, por lo que se realiza una selección implícita de variables:

```
```{r results='asis', size="small",warning=FALSE,message=FALSE}
Lasso <- glmnet(x = model, y = response, alpha = 1)
plot(Lasso, xvar = "lambda", label = TRUE)
```
```



El modelo va descartando variables de tal forma que obtengamos el mín. error posible en la estimación final:

```
```{r results='asis', size="small",warning=FALSE,message=FALSE}
Lasso2 <- cv.glmnet(x = model, y = response, alpha = 1, nfolds = 100)
plot(Lasso2)
pander(as.data.frame(as.matrix(predict(Lasso, type = "coefficients",s = Lasso2$lambda.min))), split.cell =
80, split.table = Inf)
```
```



(Intercept)	-302
longitude	0
latitude	7.588
minimum_nights	0
number_of_reviews	0
review_scores_value	0
calculated_host_listings_count	0
bedrooms	0.103
reviews_per_month	-0.08095
beds	0
accommodates	0.0918
availability_30	0
availability_60	0
availability_90	0
instant_bookablef	-0.01339
instant_bookablet	0
Distancia_Centro	-0.2334
Distancia_Norte	0
Distancia_Sur	0
tv_ports	0
phone_ports	0
Vecinos	0
Piso	0
ventanas	0

Como vemos, el modelo hace una selección de variables o lo que es lo mismo, establece el valor de beta en 0 para algunas de ellas, quedándose con el resto.

RECUERDA

El modelo final es aquel cuya lambda minimiza el error general del modelo.

¿Podemos comparar los modelos que hemos ido generando?

```
```{r results='asis', warning=FALSE,message=FALSE}
rand<-sample(c(1:nrow(pisos)),size=0.5*nrow(pisos),replace = FALSE)

#Entrenamiento
modelopisos<-glm(logprice~.,data=pisos)

#Nueva Base de Datos
newX <- model.matrix(~.-1-logprice,data=pisos[-rand,])

#Error
sum((exp(pisos[-rand,]$logprice)-exp(predict(modelopisos, newx = pisos[-rand,])))**2)
sum((exp(pisos[-rand,]$logprice)-exp(predict(Lasso, s = Lasso2$lambda.min,newx = newX)))**2)
sum((exp(pisos[-rand,]$logprice)-exp(predict(Ridge, s = Ridge2$lambda.min,newx = newX)))**2)
```
```

```
[1] 2531371
[1] 742126.7
[1] 734396.9
```

Vemos la optimización clara de los modelos Lasso y Ridge sobre un modelo en el que incluimos todas las variables. Comparando los errores cuadráticos de los modelos anteriores, vemos que el que mejor aproximación a la realidad tiene es el modelo Ridge, seguido por el Lasso y finalmente el GLM.

¿Se puede generalizar este tipo de modelos al igual que los modelos lineales?

Efectivamente, hay toda una literatura de modelos Ridge y Lasso en el mundo de los GLM (Modelos Lineales Generalizados).

Maximizábamos la función de verosimilitud y con ella obteníamos los parámetros. Ahora, vamos a hacer exactamente lo mism,o pero introduciendo el hiperparámetro lambda igual que lo hemos hecho con el modelo lineal simple:

$$\hat{\beta}_{\lambda,\alpha} := \min\left\{-L(\beta) + \lambda \sum_{j=1}^{p} (\alpha|\beta_j| + (1-\alpha)|\beta_j|^2)\right\}$$

Pasamos a la última sección del programa en la que vamos a ver más metodologías que nos van a ayudar a comprender los problemas que normalmente se tratan.

OTROS MÉTODOS: PLS

Existen otros métodos para la reducción de la dimensionalidad.

Entre los más populares, también se encuentra el de mín. s cuadrados parciales (PLS). Lo que vamos a hacer es construir "p" nuevas variables a partir de las variables originales de nuestro modelo para realizar nuestra regresión:

Variables Originales = (X1,,,Xp)
Variables Nuevas = (P1,,,Pp)
Los predictores PLS serán los siguientes:

$$P_1 = \sum a_{1j} X_j$$

El problema, en primera instancia, consiste en definir los valores de "a". Vamos a realizar el proceso en varios pasos:

Primer Paso

Definimos los valores de "a" para P1. Realizamos la covarianza entre cada una de las variables explicativas y la variable respuesta y la estandarizamos:

$$a_{1j} = \frac{cov(X_j, Y)}{var(X_j)}$$

Con esto ya tendríamos definido P1.

Segundo Paso

Nuestro segundo paso va a ser definir P2, pero eliminando los efectos de P1 en las variables explicativas, para evitar correlaciones entre nuestros nuevos predictores.

Esto lo vamos a conseguir regresionando cada variable explicativa con P1 y, posteriormente, con los errores, vamos a formar las nuevas "a" que definan P2:

$$a_{2j} = \frac{cov(e_j, Y)}{var(e_j)}$$

Con esto tendríamos definido P2.

Tercer Paso

El mismo proceso que en el paso anterior lo vamos a repetir con tantas nuevas variables nos queden por estimar. De esta forma regresionamos las variables explicativas con Pj y sacamos los errores de dicha regresión para obtener coeficientes.

Cuarto Paso

Una vez hemos obtenido todas las variables regresoras nuevas (P), realizaremos la nueva regresión.

Todo esto es lo que hay detrás de la función plsr de R. Con esto vamos a ver:

```
```{r results='asis', warning=FALSE,message=FALSE}
mpls<-plsr(logprice~.,data=pisos)
summary(mpls)
```
```

```
Data:   X dimension: 198 22
        Y dimension: 198 1
Fit method: kernelpls
Number of components considered: 22
TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11
comps  12 comps
X          51.958   80.076    98.84    99.30    99.74    99.76    99.80    99.83    99.87     99.89
99.90     99.92
logprice    5.427    6.597     7.89    30.27    37.87    45.60    48.28    49.71    49.92     50.18
50.33     50.38
          13 comps  14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22
comps
X            99.93     99.95     99.96     99.96     99.97     99.98     99.99    100.00    100.00
100.00
logprice     50.42     50.47     50.75     51.49     51.74     51.85     51.87     51.87     52.14
52.18
```

La primera linea (X) incluye el porcentaje de varianza explicada de los predictores originales, por eso converge al 100% al utilizar todos los predictores.

La segunda línea (logprice) incluye el porcentaje de varianza explicada de la respuesta, es decir, el R cuadrado. Como vemos, a partir de 6 componentes, al R2 le cuesta incrementar por cada nuevo predictor que incorporamos.

Por lo que en el siguiente trozo de código, vamos a decirle que nuestro modelo retenga, solamente, los primeros seis componentes. Como habíamos dicho, esta es una técnica de reducción de dimensionalidad, con un enfoque distinto al que habíamos visto con Ridge y Lasso. En esta ocasión, si que contamos con todos los componentes y tendríamos el mismo resultado que con el glm al completo que ya habíamos visto. Esta técnica se utiliza para eliminar dimensionalidades al problema y, por lo tanto, hacerlo más tangible o asumible computacionalmente:

```
```{r results='asis', warning=FALSE,message=FALSE}
mpls<-plsr(logprice~.,data=pisos,ncomp=6)
summary(mpls)
```
```

```
Data:   X dimension: 198 22
        Y dimension: 198 1
Fit method: kernelpls
Number of components considered: 6
TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X          51.958   80.076    98.84    99.30    99.74    99.76
logprice    5.427    6.597     7.89    30.27    37.87    45.60
```

IDEAS CLAVE

- A la hora de validar nuestro modelo y medir su capacidad predictiva real, debemos recurrir a las técnicas de cross-validation con las que entrenar nuestro modelo con un set de datos independiente al set de datos con el que medir su bondad de ajuste.

- Además, las técnicas de remuestreo nos ayudan a comprender la volatilidad de nuestros estimadores dependiendo de la submuestra con la que trabajemos y, por lo tanto, es una medida del riesgo implícito (volatilidad) del modelo.

- Las técnicas de regularización del modelo lineal son una alternativa para reducción de dimensionalidad ante un determinado problema. Son muy útiles cuando hay un conjunto de variables alto y un número de observaciones limitado.