Selección de modelo Pt 3

Fernando Anorve

3/31/2020

Recordemos que tenemos una respuesta $\mathbf{Y}$ y potenciales predictores $X_1, X_2, \ldots, X_p$, donde p es considerablemente grande.

Hemos estado revisando métodos que nos ayuden a escoger mejor las variables, de modo que el modelo que utilicemos sea lo más simple posible, sin que éste quede sesgado de forma importante.

Métodos de regularización o “encogimiento”

En los algoritmos anteriores, hemos tratado de encontrar un subconjunto conveniente de los parámetros para simplificar nuestro modelo lineal.

Otra técnica, en cambio, incluye incluir los p predictores “restringir” o “regularizar” las estimaciones de los coeficientes $\hat{\beta}_j$ (dicho de otra forma, “encogerlas” hacia cero), para reducir su varianza.

Dichas técnicas incluyen:

- Regresión Ridge
- Regresión Lasso

Regresión Ridge

En el contexto de la regresión lineal en su versión de mínimos cuadrados, queremos minimizar la distancia entre valores estimados y valores observados.

$$RSS = \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

De hecho, estrictamente hablando, entre más predictores agreguemos, esa distancia siempre tenderá a disminuir (ya sea mucho o poco), y jamás aumentará. Como habíamos mencionado antes, esa no es la idea…

En este nuevo contexto de “regularización”, además de reducir la distancia entre valores y sus estimaciones, también queremos restingir la magnitud de los coeficientes β_j. Dicho de otra forma, “encogerlos” hacia cero.

Para limitar la magnitud de los coeficientes β_j, se agrega un término de penalización que depende de un parámetro $0 < \lambda < \infty$

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

Ahora hay que minimizar

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

¿Eso qué implica cuando $0 < \lambda < \infty$?

- Si todos (o muchos) los β_j son sustancialmente distintos a cero, $\lambda \sum_{j=1}^{p} \beta_j^2$ crece… Y no queremos eso
- Si todos (o muchos) los β_j son aproximadamente 0, el modelo se vuelve naíf, y el RSS aumenta… tampoco queremos eso

Formalmente,

$$\hat{\beta}_{ridge} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

Pero… ¿qué es λ?

Hasta ahora no hemos dicho quién es λ. Primero hay que notar un par de cosas:

- Cuando $\lambda \to 0$, ocurre que

  $$RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \approx RSS,$$

  por lo que $\hat{\beta}_{ridge} \approx \hat{\beta}_{LSE}$ (modelo inicial, de mínimos cuadrados)

- Cuando $\lambda \to \infty$, ocurre que

  $$RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \to \infty,$$

  salvo que $\sum_{j=1}^{p} \beta_j^2 = 0$, por lo que $\hat{\beta}_{ridge} = 0$ (modelo naíf)

Hay que hallar λ

1. Suficientemente grande para obtener un modelo más simple que el original (mínimos cuadrados)
2. Suficientemente pequeña como para obtener un modelo suficientemente diferente al naíf

¡Validación Cruzada!

Regresión Lasso

Al igual que en la regresión Ridge, también queremos restingir la magnitud de los coeficientes β_j, al mismo tiempo que queremos restringir la distancia entre valores y sus estimaciones.

Para ello se agrega un término de penalización que depende de un parámetro $0 < \lambda < \infty$

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

Entonces hay que minimizar

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

¿Eso qué implica cuando $0 < \lambda < \infty$?

- Si todos (o muchos) los β_j son sustancialmente distintos a cero, $\lambda \sum_{j=1}^{p} |\beta_j|$ crece
- Si todos (o muchos) los β_j son aproximadamente 0, el modelo se vuelve naíf, y el RSS aumenta

Formalmente,

$$\hat{\beta}_{lasso} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Recordemos que al igual que en el caso anterior:

- Cuando $\lambda \to 0$, $\hat{\beta}_{lasso} \approx \hat{\beta}_{LSE}$
- Cuando $\lambda \to \infty$, $\hat{\beta}_{lasso} = 0$ (modelo naíf)

Hay que hallar λ con los mismos criterios que antes:

1. Suficientemente grande para obtener un modelo más simple que el original
2. Suficientemente pequeña como para obtener un modelo suficientemente diferente al naíf

Igual que antes, podemos usar **validación cruzada** para hallar el valor ideal de λ

Ejemplos

LASSO

Consideremos de nuevo el conjunto Hitters de la librería ISLR, que contiene datos de las ligas mayores de baseball de las temporadas de 1986 y 1987.

```
data(Hitters)
Hitters = na.omit(Hitters)
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
##  [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
## [11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
## [16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
```

Nuestro objetivo: Hallar un modelo menos complejo que el que usa los 19 predictores, que tenga un ajuste “decente”.

Para llevar a cabo esta técnica podemos utilizar la librería glmnet, con la función cv.glmnet. A diferencia de otras técnicas que hemos visto antes, en vez que utilizar como parámetro una fórmula con formato “y ~ X”, se necesita una matriz X y un vector de respuesta y. La matriz X puede obtenerse del data.frame usando la función model

matrix.

- Nótese que lo que antes eran factores de n niveles, ahora son $n-1$ variables dummies (i.e. valen 0 ó 1)

Por ejemplo: si antes la variable x_1 podía valer “perro”, “gato” o “pollo.” Ahora se convierte en $es.gato \in \{0,1\}$ y $es.pollo \in \{0,1\}$, donde si ambas son 0, es porque $x_1 = $“perro”.

Llevamos a cabo lasso como sigue:

- Primero escogemos un subconjunto de entrenamiento

```
library(glmnet)

X = model.matrix( Salary ~ . ,data = Hitters)[,-1]
y = Hitters$Salary

n = length(y)

set.seed(1024)
train = sample(n,round(n/2))

X_train = X[train,]
y_train = y[train]
```

- Y sobre el conjunto de entrenamiento se utiliza la función cv.glmnet

```
cvfit = cv.glmnet(X_train, y_train,nfolds = 10)
```

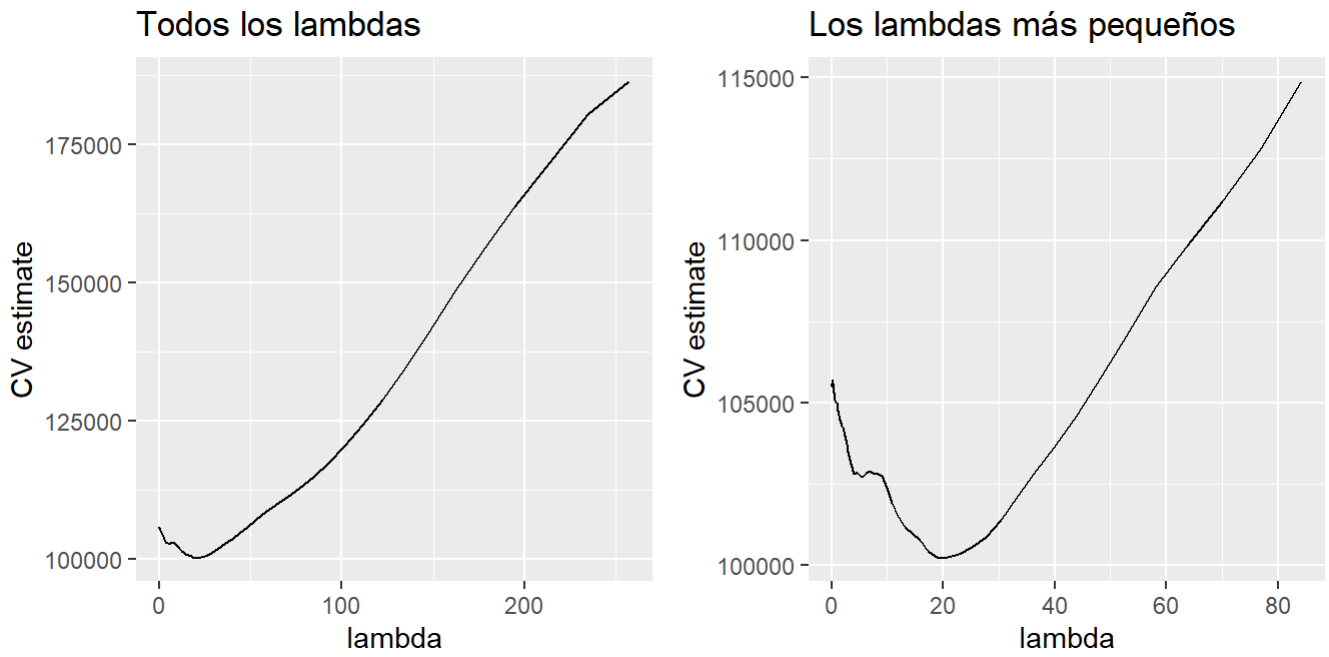
Recordemos que Lasso depende del un parámetro λ que hay que escoger estratégicamente. ¿Cómo lo escogemos?

- Echemos un vistazo a los valores de validación cruzada que ya se incluyen en el producto de cv.glmnet

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) + geom_line() + xlab("la
mbda") + ylab("CV estimate") + ggtitle("Todos los lambdas")

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvm ,80))) + geom
_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños")

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```



Notamos que probablemente necesitamos un lambda no mayor a 30… ¿pero cuál?

Hay dos técnicas: lambda.min y lambda.1se

- lambda.min consiste básicamente en tomar el lambda que resulte en la mínima estimación de validación cruzada.
- lambda.1se consiste en tomar el **modelo más simple** cuya estimación de CV esté a menos de un error estándar de la mínima estimación de CV. Recordemos que entre mayor sea λ, más simple es el modelo

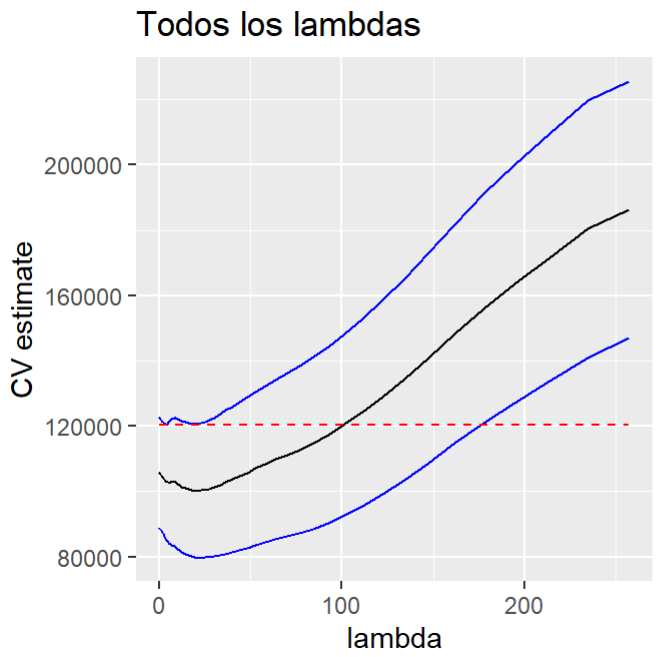
Recordemos que cada estimación del error de validación cruzada es un promedio, por lo que existe un error estándar.

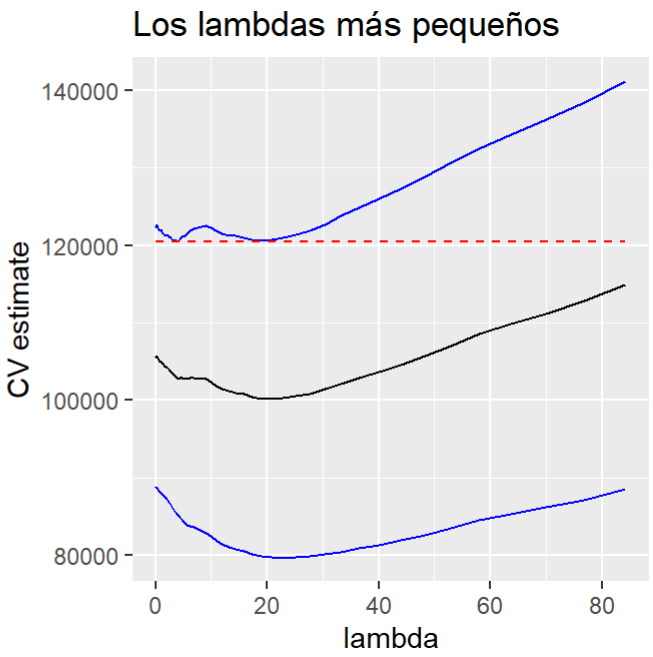
Este segundo criterio se basa en que la estimación del error de validación cruzada no es una ciencia exacta. Más bien nos interesa una estimación relativamente baja (i.e. el mínimo +/- un error estándar), y en ese intervalo, tomar el modelo más simple.

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Todos los lambdas") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvup), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvlo), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = min(cvfit$cvup)), color = "red", linetype = 2)

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvm ,80))) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvup ,80)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvlo ,80)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = min(cvfit$cvup)), color = "red", linetype = 2)

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```





¡Sorpresa! lambda.1se es mucho más alto que 30. Veamos cuál es la respuesta de R

```
cvfit$lambda.min
```

```
## [1] 19.01743
```

```
cvfit$lambda.1se
```

```
## [1] 101.4903
```

Hay que tener cuidado porque pueden ser muy distintos

¿Qué coeficientes resultan de ambos criterios?

Mínimo lambda

```
coef(cvfit, s = "lambda.min")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -45.6565384
## AtBat         .
## Hits          1.7002883
## HmRun         .
## Runs          .
## RBI           .
## Walks         3.5690164
## Years         .
## CAtBat        .
## CHits         0.3091307
## CHmRun        .
## CRuns         .
## CRBI          .
## CWalks        .
## LeagueN       .
## DivisionW   -85.1301259
## PutOuts       0.1687340
## Assists       .
## Errors        .
## NewLeagueN    .
```

Lambda 1se

```
coef(cvfit, s = "lambda.1se")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) 165.7603185
## AtBat         .
## Hits          1.0200247
## HmRun         .
## Runs          .
## RBI           .
## Walks         2.2292626
## Years         .
## CAtBat        .
## CHits         0.2023314
## CHmRun        .
## CRuns         .
## CRBI          .
## CWalks        .
## LeagueN       .
## DivisionW     .
## PutOuts       .
## Assists       .
## Errors        .
## NewLeagueN    .
```

Notamosque aquí algunos coeficientes de la regresión son iguales a cero. Es decir, se basa en un subconjunto de las variables.

Error in-sample

```
yhat=predict(cvfit,X_train,s="lambda.1se")
sqrt(mean((y_train-yhat)^2))
```

```
## [1] 330.2579
```

Se espera que al ser un modelo más complejo (i.e. menor lambda), lambda.min dé un menor error in-sample

```
yhat2=predict(cvfit,X_train,s="lambda.min")
sqrt(mean((y_train-yhat2)^2))
```

```
## [1] 292.171
```

Hay que recordar que el error in-sample no nos dice tanto como quisiéramos sobe la capacidad predictiva de un modelo.

Conjunto de validación:

Tomamos el resto de las observaciones como conjunto de validación

```
X_val = X[-train,]
y_val = y[-train]
```

Y ahora revisamos el error medio del conjunto de validación para ambos criterios

```
yhat=predict(cvfit,X_val,s="lambda.1se")
sqrt(mean((y_val-yhat)^2))
```

```
## [1] 392.5964
```

```
yhat2=predict(cvfit,X_val,s="lambda.min")
sqrt(mean((y_val-yhat2)^2))
```

```
## [1] 363.1194
```

Un resultado interesante es que el criterio lambda.1se tiene mejor capacidad de predicción que el lambda.min. Usualmente esto ocurre porque el primero es un modelo más simple, pero con estimador de error de validación cruzada relativamente bajo.

Ridge

Nuestro objetivo (de nuevo): Hallar un modelo menos complejo que el que usa las 19 covariables para predecir el salario, que tenga un ajuste “decente”.

La regresión ridge también se puede hallar en la librería glmnet, añadiendo un parámetro extra a la función cv.glmnet

```
set.seed(2020)
cvfit = cv.glmnet(X_train, y_train,nfolds = 10, alpha = 0)
```

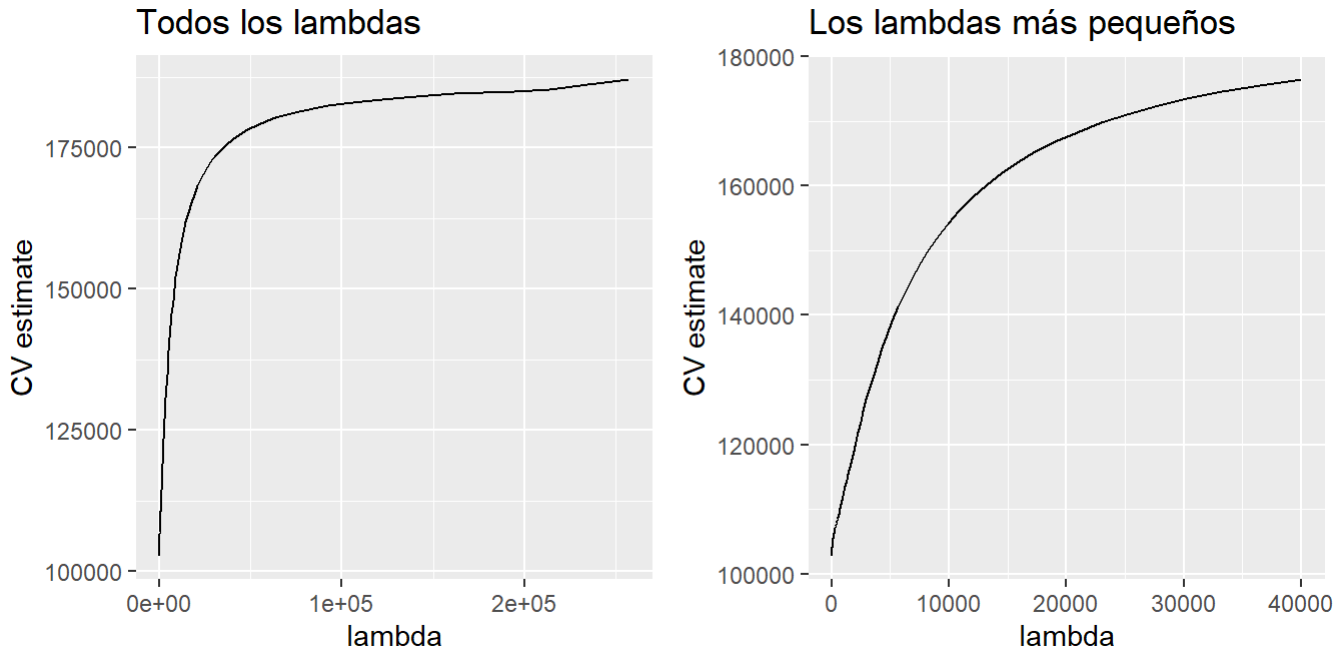
¿Cómo escoger λ?

- Echemos un vistazo a las estimaciones error de validación cruzada

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) + geom_line() + xlab("la
mbda") + ylab("CV estimate") + ggtitle("Todos los lambdas")

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvm ,80))) + geom
_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños")

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```

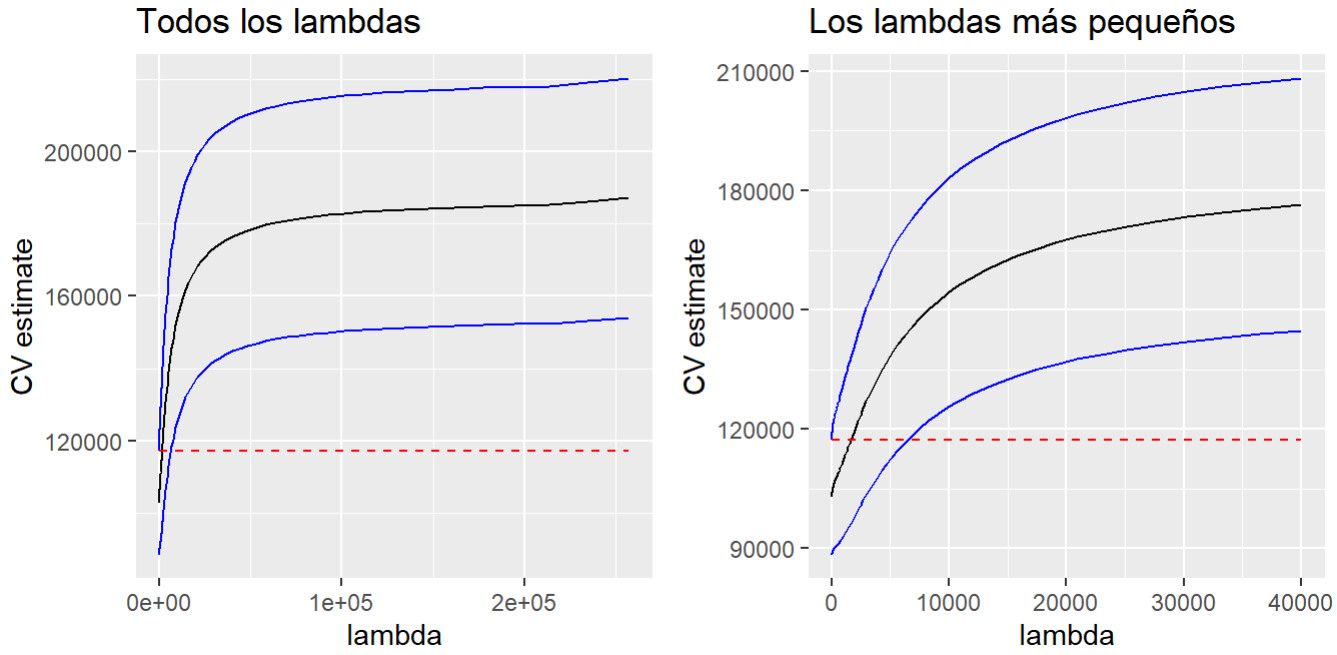


Recordemos las dos técnicas lambda.min y lambda.1se en las siguientes gráficas:

```
normal_plt = ggplot(data = NULL, aes(x = cvfit$lambda , y = cvfit$cvm)) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Todos los lambdas") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvup), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = cvfit$cvlo), color = "blue") +
  geom_line(aes(x = cvfit$lambda , y = min(cvfit$cvup)), color = "red", linetype = 2)

zoom_plt = ggplot(data = NULL, aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvm ,80))) +
  geom_line() + xlab("lambda") + ylab("CV estimate") + ggtitle("Los lambdas más pequeños") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvup ,80)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = tail(cvfit$cvlo ,80)), color = "blue") +
  geom_line(aes(x = tail(cvfit$lambda, 80) , y = min(cvfit$cvup)), color = "red", linetype = 2)

grid.arrange(normal_plt,zoom_plt,ncol = 2)
```



Interesantemente, estas gráficas son totalmente distintas a las de LASSO

```
cvfit$lambda.min
```

```
## [1] 25.7315
```

```
cvfit$lambda.1se
```

```
## [1] 1542.563
```

Al igual que antes, son muy distintos ambos valores

¿Qué coeficientes resultan de ambos criterios?

Mínimo lambda

```
coef(cvfit, s = "lambda.min")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  -19.81800827
## AtBat         -0.42238845
## Hits           2.91963399
## HmRun         -2.48455283
## Runs          -1.46056883
## RBI            1.31262589
## Walks          5.10974379
## Years        -10.39094234
## CAtBat         0.01084708
## CHits          0.23460216
## CHmRun        -1.02279878
## CRuns          0.35245822
## CRBI           0.21316561
## CWalks        -0.13300914
## LeagueN      149.96726951
## DivisionW   -100.94833285
## PutOuts        0.22494136
## Assists       -0.20275645
## Errors         2.50317769
## NewLeagueN  -163.88391208
```

Lambda 1se

```
coef(cvfit, s = "lambda.1se")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) 117.778836371
## AtBat         0.118154583
## Hits          0.531071677
## HmRun         0.926156695
## Runs          0.733574018
## RBI           0.691331174
## Walks         1.139769088
## Years         2.987246508
## CAtBat        0.009843520
## CHits         0.041009930
## CHmRun        0.131509928
## CRuns         0.074906043
## CRBI          0.062481392
## CWalks        0.073469783
## LeagueN       5.760402921
## DivisionW   -28.764389111
## PutOuts       0.050446215
## Assists       0.006373237
## Errors        0.543766363
## NewLeagueN   -7.401745760
```

Notamos una gran diferencia respecto a LASSO, donde algunos coeficientes de la regresión eran iguales a cero. Aquí en cambio todos son distintos de cero. Esta es una característica inherente a la regresión Ridge, y a su vez una desventaja frente a LASSO: no se usa Ridge para selección de variables propiamente hablando.

Error in-sample

```
yhat=predict(cvfit,X_train,s="lambda.1se")
sqrt(mean((y_train-yhat)^2))
```

```
## [1] 327.6836
```

Se espera que al ser un modelo más complejo (i.e. menor lambda), lambda.min dé un menor error in-sample

```
yhat2=predict(cvfit,X_train,s="lambda.min")
sqrt(mean((y_train-yhat2)^2))
```

```
## [1] 272.5306
```

Conjunto de validación:

Tomamos el resto de las observaciones como conjunto de validación

```
X_val = X[-train,]
y_val = y[-train]
```

Y ahora revisamos el error medio del conjunto de validación para ambos criterios

```
yhat=predict(cvfit,X_val,s="lambda.1se")
sqrt(mean((y_val-yhat)^2))
```

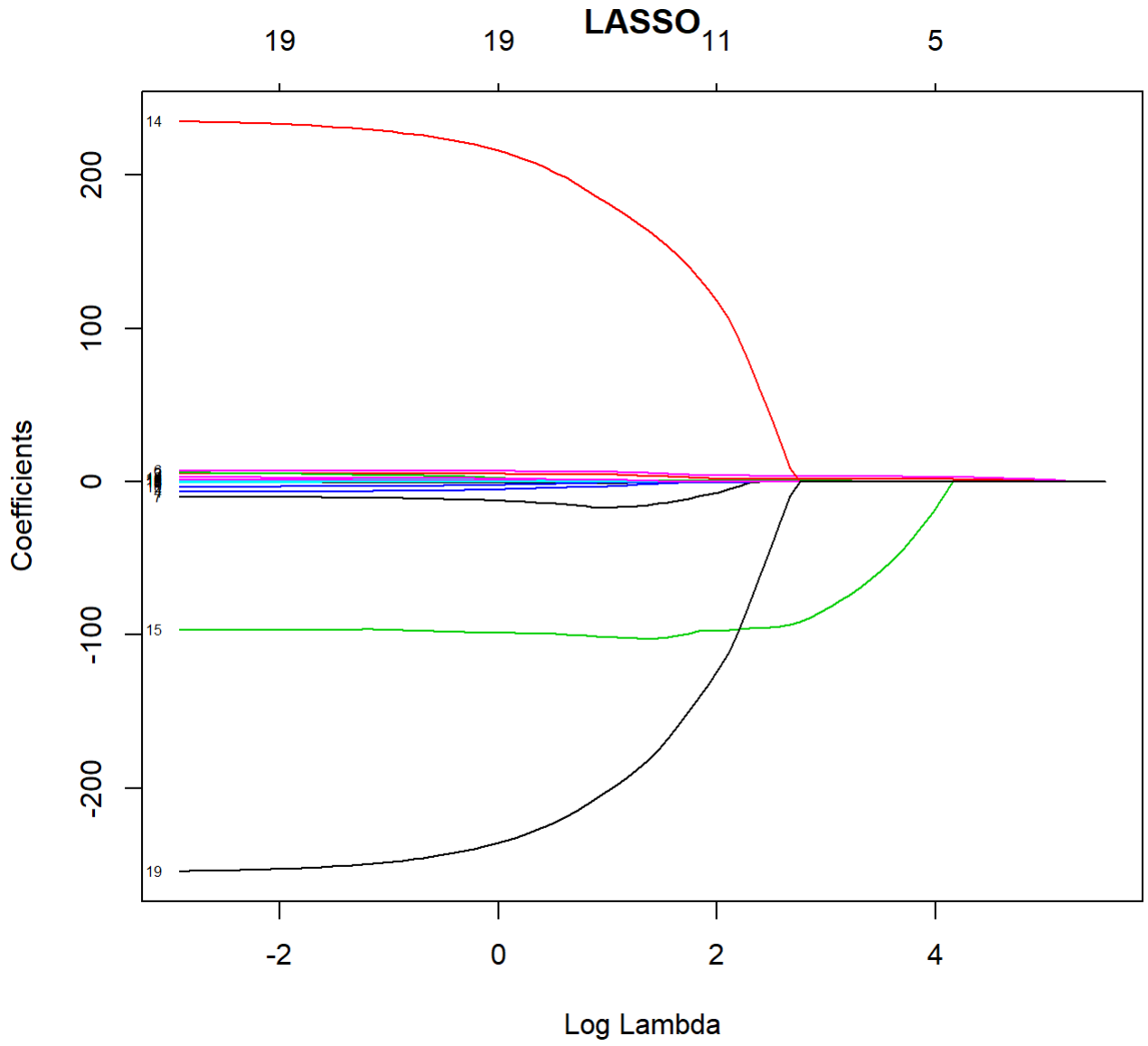
```
## [1] 379.2011
```

```
yhat2=predict(cvfit,X_val,s="lambda.min")
sqrt(mean((y_val-yhat2)^2))
```

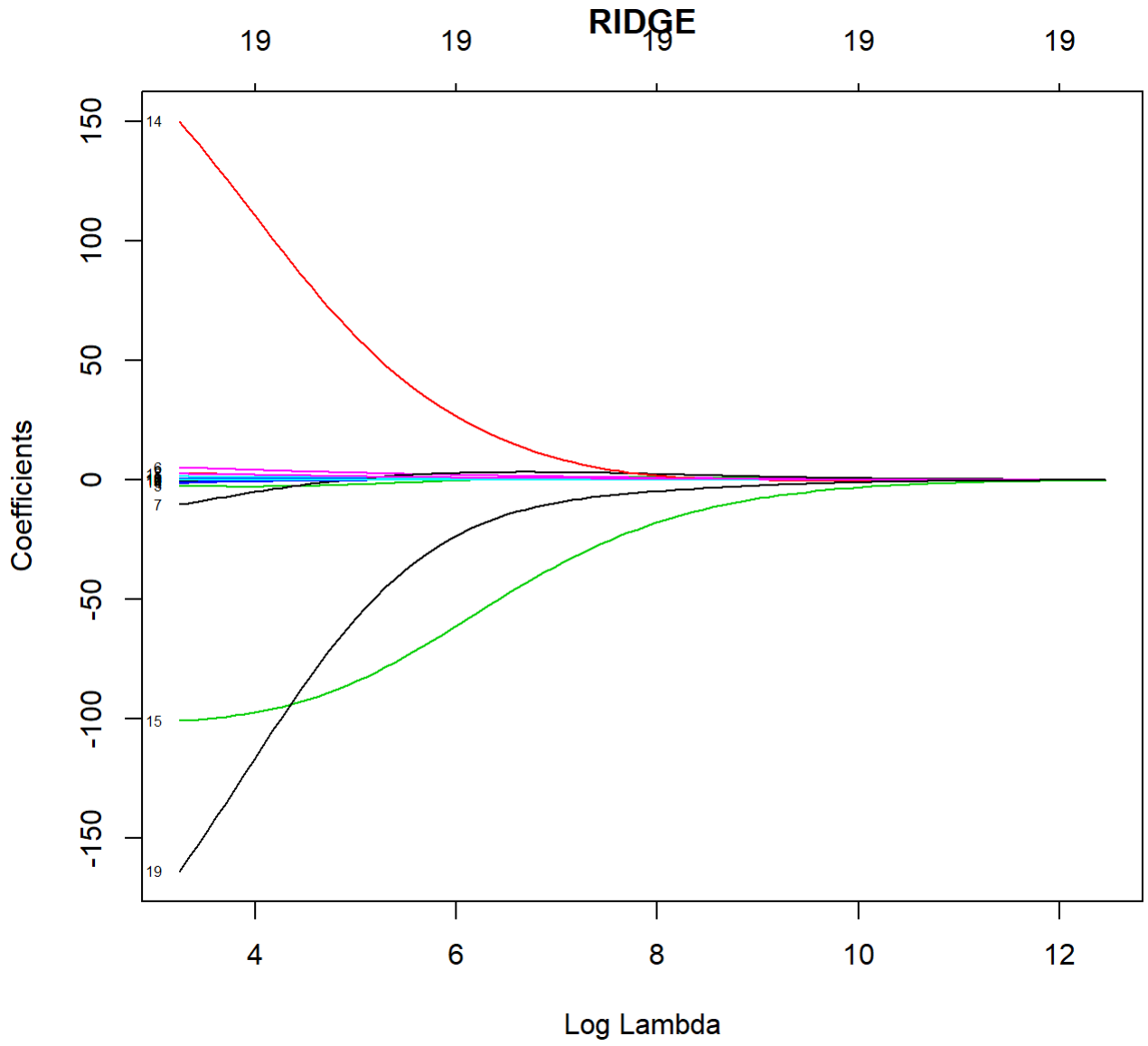
```
## [1] 377.2936
```

Comparación

```
glmfit=glmnet(X_train,y_train,alpha = 1)
plot(glmfit,xvar="lambda", main = "LASSO", label = T)
```




```
glmfit=glmnet(X_train,y_train,alpha = 0)
plot(glmfit,xvar="lambda", main = "RIDGE",label = T)
```


RIDGE
19
19
19
19
19
Coefficients
150
100
50
0
-50
-100
-150
14
15
19
7
4
6
8
10
12
Log Lambda