Selección de predictores y mejor modelo lineal múltiple: subset selection, ridge regression, lasso regression y dimension reduction

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Diciembre 2016

Índice

Introducción

Lecturas previas recomendadas:

- Introducción a la Regresión Lineal Múltiple
- Validación de modelos de regresión: *Cross-validation, OneLeaveOut, Bootstrap*

Definiciones:

El *bias* o sesgo de un modelo estadístico es el error que se introduce al intentar explicar la relación existente entre variables del mundo real, que suele ser extremadamente complicada, mediante un modelo matemático mucho más simple. El *bias* de un modelo es independiente de los datos empleados para crearlo, depende únicamente del tipo de modelo utilizado para intentar explicar la relación entre variables.

La varianza de un modelo estadístico se define como la variación que tiene un modelo dependiendo de las observaciones que se empleen para ajustarlo (*training data*).

Los modelos de regresión lineal múltiples suelen, a modo general, ajustarse mediante regresión por mínimos cuadrados. Existen dos aspectos principales a tener en cuenta a la hora de emplear este método de ajuste:

Precisión de predicción: Si la relación existente entre los predictores y la variable dependiente es aproximadamente lineal, el método de mínimos cuadrados tendrá un *bias* pequeño. Si además, el número de observaciones con las que se ajusta el modelo es mucho mayor que el número de predictores ($n>>p$), las estimaciones obtenidas por mínimos cuadrados tienden a tener poca varianza. Cumpliéndose ambas condiciones, un modelo lineal múltiple generado mediante mínimos cuadrados tendrá una buena capacidad de predicción (un *test error rate* bajo). A medida que el número de observaciones deja de ser mucho mayor que el número de predictores, la varianza aumenta, hasta llegar al punto en que, si $p>n$, la varianza es infinita y por lo tanto el método de mínimos cuadrados no debe utilizarse.

Interpretabilidad del modelo: Cuantos más predictores se introduzcan en un modelo, más compleja se hace su interpretación. Por esta razón, es conveniente limitar el modelo a aquellos predictores que tengan una influencia importante sobre la variable respuesta estudiada, excluyendo aquellos que son irrelevantes y que añaden complejidad innecesaria. El método de regresión por mínimos cuadrados difícilmente obtendrá estimaciones de coeficientes que sean exactamente 0, por lo que tenderá a considerar útiles todos los predictores.

Así pues, cuando se dispone de pocas observaciones o muchos predictores, es conveniente emplear un método de regresión que permita excluir predictores irrelevantes o, mejor dicho, identificar los más relevantes. A este proceso se le conoce como *feature selection* o *variable selection* y 3 de los métodos más empleados son: *subset selection, shrinkage/regularization* y *dimension reduction*. Cabe destacar que todos estos métodos reducen la varianza de las estimaciones pero siguen siendo modelos lineales.

Estimaciones del *test error*

Cuando se quiere elegir el mejor de entre un conjunto de modelos, es necesaria una medida que permita compararlos. Algunas medidas de bondad de ajuste de un modelo son:

- La Suma de Cuadrados Residuales (*Residual Sum of Squares RSS*) y R^2: ambos para modelos por mínimos cuadrados.
- *Deviance*: para modelos que no emplean mínimos cuadrados, como por ejemplo la regresión logística.

Estos estadísticos miden el *training error* y por lo tanto solo se pueden emplear para comparar modelos que tienen el mismo número de predictores, ya que por definición, cuantos más predictores se introducen en un modelo, menor es su *training error*. Sin embargo, lo que realmente permite cuantificar cómo de útil es un modelo no es el *training error* si no el *test error*, por lo que es esta última medida en la que hay que basarse para elegir entre modelos con diferente número de predictores. Generalmente, al hablar de *test error* se hace referencia al *test mean square error (test-MSE),* que equivale al *test Residual Sum of Squares* dividido por el número de observaciones $MSE = \frac{RSS}{n}$.

Existen dos tipos de aproximaciones para estimar el *test error*:

- Estimación indirecta del *test error* a partir de un ajuste hecho sobre el *training error* que compense el *bias* por *overfitting*: C_p, AIC, BIC y $R^2_{ajustado}$.
- Estimación directa del *test error* mediante *validation set* o *cross-validation*.

Mallow's (Cp)

El estadístico C_p introduce una penalización (en sentido creciente) de $2d\,\hat{\hat{\sigma}}^2$ al *training-RSS* con el objetivo de compensar el hecho de que, a medida que se introducen más predictores, el *training-RSS* estima a la baja el *test-RSS*. Cuanto menor es el valor de C_p, mejor el modelo.

$$C_p = \frac{1}{n}(RSS + 2d\,\hat{\hat{\sigma}}^2)$$

Siendo d el número de predictores y $\hat{\sigma}^2$ la estimación de la varianza del error ϵ.

Dado que requiere estimar $\hat{\sigma}^2$, este método no es aplicable si el número de predictores es mayor que el número de observaciones. Tampoco es recomendable si el número de predictores se aproxima al de observaciones.

AIC

El estadístico *Akaike Information Criterion (AIC)* se puede aplicar a una gran cantidad de modelos ajustados mediante *maximum likelihood* (mínimos cuadrados es un caso particular de *maximum likelihood*). Para regresión lineal por mínimos cuadrados, el valor de AIC es proporcional al de C_p por lo que ambos llevan a la selección del mismo modelo.

$$AIC = \frac{1}{n\,\hat{\sigma}^2}(RSS + 2d\,\hat{\sigma}^2)$$

BIC

El método *Bayesian Point of View (BIC)* para modelos de mínimos cuadrados se define como:

$$C_p = \frac{1}{n}(RSS + log(n)d\,\hat{\sigma}^2)$$

A diferencia de C_p, en BIC la penalización está determinada por $log(n)$, siendo n el número de observaciones. Esto implica que, cuando $n > 7$, el método BIC introduce mayores penalizaciones, tendiendo a seleccionar modelos con menos predictores que los seleccionados por C_p (y también que AIC).

R^2-ajustado

El valor de R^2 se define como el porcentaje de varianza de la variable respuesta explicada por el modelo respecto del total de varianza observada. En los modelos múltiples, cuantos más predictores se incluyan en el modelo, mayor es el valor de R^2, ya que, por poco que sea, cada predictor va a explicar una parte de la varianza observada. La idea detrás de $R^2_{ajustado}$ es que, una vez que los predictores correctos se han incluido en el modelo, la varianza extra que se consigue explicar añadiendo más predictores no compensa la penalización.

$$R^2_{ajustado} = 1 - \frac{RSS}{TSS} x \frac{n-1}{n-k-1} = R^2 - (1-R^2)\frac{n-1}{n-k-1} = 1 - \frac{RSS/df_e}{TSS/df_t}$$

Al contrario que C_p, AIC y BIC, cuanto mayor sea el valor de $R^2_{ajustado}$, mejor el modelo. $R^2_{ajustado}$ no requiere de la estimación de $\hat{\sigma}^2$ por lo que se puede emplear cuando el número de predictores supera al número de observaciones. $R^2_{ajustado}$ no es generalizable a modelos no lineales.

Ninguno de los métodos mencionados anteriormente, a excepción de $R^2_{ajustado}$, se debe utilizar cuando el número de predictores se aproxima o supera al número de observaciones, ya que requieren de la estimación de $\hat{\sigma}^2$ y está no es precisa es este tipo de situaciones.

Validation and Cross-Validation

Mediante *Simple Validation* y *Cross-Validation* se puede estimar el *test error* de cada modelo y seleccionar aquel para el que sea menor. La ventaja de este método frente a los anteriormente descritos es que se trata de una estimación directa que requiere de menos asunciones. Al no ser necesaria una estimación de la varianza residual, ni conocer el número de predictores, se puede aplicar en un rango mayor de modelos. Dado que en la actualidad no suponen un problema computacional, en el libro *ISLR* consideran este método superior a los anteriores.

One standard error rule

La estimación de *test error* de cada modelo tiene asociado un error estándar. La norma de *one-standar-error* recomienda elegir como mejor modelo aquel que contenga menos predictores y cuya estimación de *test error* no se aleje más de 1 error estándar del modelo con menor *test error*. La idea es que si varios modelos tienen semejantes valores de *test error*, es decir, son aproximadamente igual de buenos, se debe escoger el más simple de ellos (principio de parsimonia).

Subset Selection

Los métodos conocidos como *subset selection* tienen la finalidad de identificar y seleccionar, de entre todos los predictores disponibles, aquellos que están más relacionados con la variable respuesta y así crear el mejor modelo. Dentro de este grupo se diferencian: *best subset selection* y *stepwise selection (forward, backward* e *hybrid)*. Para un mismo conjunto de datos, no todos tienen por qué converger en un mismo modelo final.

El esquema general de los métodos de *subset selection* consiste en:

1. Crear un conjunto de modelos, todos los posibles (*best subset*) o bien un conjunto considerable de ellos (*stepwise*), mediante diferentes combinaciones de los predictores disponibles.
2. Para cada posible tamaño de modelo (1 predictor, 2 predictores...) se selecciona el mejor basándose en el *training error*, normalmente el *RSS*.
3. Los modelos finalistas de cada tamaño se comparan entre ellos para identificar el mejor basándose en la estimación del *test error* (*cross-validation error*, C_p, AIC, BIC o $R^2_{ajustado}$) .

Best Subset Selection

El proceso de *Best Subset Selection* consiste en evaluar todos los posibles modelos que se pueden crear por combinación de los predictores disponibles. El algoritmo a seguir es el siguiente:

1. Se genera lo que se conoce como modelo nulo (M_0), que es el modelo sin ningún predictor.
2. Se generan todos los posibles modelos que contienen un único predictor y se selecciona el que tiene menor *Residual Sum of Squares (RSS)* o mayor R^2. Al modelo seleccionado se denomina (M_1).
3. Se repite el paso anterior para modelos con dos predictores y así sucesivamente hasta llegar al modelo con todos los predictores (M_k).
4. De entre los mejores modelos seleccionados para cada número de predictores (M_0, M_1, M_2,...,M_k) se identifica el mejor modelo, esta vez empleando *cross-validation error*, C_p, AIC, BIC o $R^2_{ajustado}$.

A pesar de que este método explora todas las posibilidades, tiene dos limitaciones fundamentales:

- Requerimientos computacionales: Se requiere calcular 2^p modelos distintos, lo que lo hace inviable para más de 40 predictores.
- Problemas de *overfitting* y varianza: Al generarse tantos modelos, por simple azar se pueden encontrar buenos resultados. En los vídeos de *ISLR* no lo recomiendan si hay más de 10 predictores.

Forward Stepwise Selection

Es una alternativa computacionalmente más eficiente que *Best Subset Selection* en la que no se evalúan todas las posibles combinaciones de predictores sino un subconjunto de las mismas. El proceso se inicia generando el modelo nulo (M_0) sin predictores. A continuación se generan todos los posibles modelos que se pueden crear introduciendo un predictor más al modelo nulo. De entre todos estos modelos con 1 predictor se selecciona el mejor basándose en *Residual Sum of Squares (RSS)* o R^2, al modelo elegido se denomina M_1. Se repite el paso anterior pero esta vez partiendo del último modelo seleccionado y así sucesivamente hasta llegar al modelo con todos los predictores. De entre los mejores modelos seleccionados para cada número de predictores (M_0, M_1, M_2,...,M_k) se identifica el mejor, esta vez empleando *cross-validation error*, C_p, AIC, BIC o $R^2_{ajustado}$.

Al crear modelos anidados, en los que el modelo k se construye a partir del modelo $k-1$, el método *forward stepwise selection* no garantiza que se seleccione el mejor modelo de entre todos los posibles ya que no se evalúan todas las posibles combinaciones. Sin embargo, suele llegar a modelos muy optimizados consiguiendo un buen rendimiento computacional y evitando el *overfitting*. *Forward stepwise selection* puede emplearse incluso cuando el número de predictores es mayor que el de observaciones.

Backward Stepwise Selection

El concepto es equivalente al de *forward stepwise selection* pero en este caso iniciando el proceso a partir del modelo que contiene todos los posibles predictores, *full model*. En cada iteración se generan todos los modelos que se pueden crear eliminando un único predictor a la vez y se selecciona el que tiene menor *Residual Sum of Squares (RSS)* o mayor R^2. El proceso se repite hasta llegar al modelo nulo sin predictores. De entre los mejores modelos seleccionados para cada número de predictores (M_0, M_1, M_2,...,M_k) se identifica el mejor, esta vez empleando *cross-validation error*, C_p, AIC, BIC o $R^2_{ajustado}$. *Backward stepwise selection* permite evaluar cada variable en presencia de las otras, lo que es una ventaja frente a *forward stepwise selection*. Sin embargo, dado que el método se inicia con el modelo que contiene todos los predictores, si la regresión es por mínimos cuadrados, no se puede aplicar *backward stepwise selection* cuando el número de predictores es mayor que el número de observaciones.

Hibrid (double) Stepwise Selection

Este método se inicia al igual que el *forward* pero, tras cada nueva incorporación, se realiza un test de extracción de predictores no útiles (como en el *backward*). Este método se aproxima más al *Best Subset Selection* pero sin caer en limitaciones computacionales.

Shrinkage/regularization methods

Los métodos de *subset selection* descritos anteriormente emplean mínimos cuadrados para ajustar un modelo lineal que contiene únicamente un subconjunto de predictores. Otra alternativa, conocida como *shrinkage* o *regularization*, consiste en ajustar el modelo incluyendo todos los predictores pero empleando un método que fuerce a que las estimaciones de los coeficientes de regresión tiendan a cero, es decir, que tienda a minimizar la influencia de los predictores menos importantes. Dos de los métodos más empleados son:

- *Ridge regression*: aproxima a cero los coeficientes de los predictores pero sin llegar a excluir ninguno.
- *Lasso*: aproxima a cero los coeficientes, llegando a excluir predictores.

Ambos métodos están especialmente indicados para situaciones en las que hay un mayor número de predictores que de observaciones.

La magnitud de los coeficientes de correlación de los predictores de un modelo lineal depende de la escala en que se mida cada predictor. El *data set* `diamonds` del paquete `ggplot2` contiene información sobre el precio (*price*), peso (*carat*) y tamaño (*depth*) de diamantes. Supóngase que se quiere generar un modelo lineal múltiple que prediga el precio en función del peso y tamaño, siendo las unidades dólares, gramos y milímetros respectivamente.

```
require(ggplot2)
data("diamonds")
coef(lm(price ~ carat + depth, data = diamonds))
```

```
## (Intercept)       carat       depth
##   4045.3332   7765.1407   -102.1653
```

Acorde al modelo, por cada unidad que se incrementa el peso de un diamante, manteniéndose constante el tamaño, el precio aumenta en promedio 7765.1407 dólares. Si en lugar de medir el peso en gramos se hace en miligramos, el resultado obtenido es el siguiente:

```
diamonds$carat <- diamonds$carat*1000
coef(lm(price ~ carat + depth, data = diamonds))
```

```
## (Intercept)       carat       depth
## 4045.333183    7.765141 -102.165322
```

Esta vez, por cada unidad que se incrementa el peso de un diamante, manteniéndose constante el tamaño, el precio aumenta en promedio 7.7651407 dólares. Esto pone de manifiesto que la magnitud de los coeficientes de correlación no puede emplearse para comparar la importancia que tienen los predictores en el modelo si estos se miden en distinta escala. Para poder considerar que, cuanto más próximo a cero es el coeficiente de regresión de un predictor, menor su influencia sobre la variable respuesta en comparación al resto, es necesario estandarizar todos los predictores antes de ajustar el modelo. La función `scale(center = TRUE, scale = TRUE)` permite hacerlo. **Los métodos de *ridge regression* y *Lasso* requieren una estandarización previa de los predictores**.

Ridge regression

Ridge regression es similar al ajuste por mínimos cuadrados en cuanto que ambos tratan de minimizar el *Residual Sum of Squares (RSS)*. La diferencia reside en que *ridge regression* incorpora un término llamado *shrinkage penalty* que fuerza a que los coeficientes de los predictores tiendan a cero. El efecto de esta penalización está controlada por el parámetro λ. Cuando $\lambda = 0$ la penalización es nula y los resultados son equivalentes a los obtenidos por mínimos cuadrados, cuando $\lambda = \infty$ todos los coeficientes son cero, lo que equivale al modelo sin ningún predictor (modelo nulo).

La principal ventaja del ajuste por *ridge regression* frente al ajuste por mínimos cuadrados es la reducción de varianza. Por lo general, en situaciones en las que la relación entre la variable respuesta y los predictores es aproximadamente lineal, las estimaciones por mínimos cuadrados tienen poco *bias* pero aún pueden sufrir alta varianza (pequeños cambios en los datos de entrenamiento tienen mucho impacto en el modelo resultante). Este problema se acentúa conforme el número de predictores introducido en el modelo se aproxima al número de observaciones de entrenamiento, llegando al punto en que, si $p>n$, no es posible ajustar por mínimos cuadrados. Empleando un valor adecuado de λ, identificado mediante *cross-validation*, el método de *ridge regression* es capaz de reducir varianza sin apenas aumentar el *bias*, consiguiendo así un menor error total.

La limitación del método de ajuste por *ridge regression* en comparación a los métodos de *subset selection* es que el modelo final va a incluir todos los predictores. Esto es así porque, si bien la penalización empleada fuerza a que los coeficientes tiendan a cero, nunca llegan a ser exactamente cero (solo si $\lambda = \infty$). Este método consigue minimizar la influencia sobre el modelo de los predictores menos relacionados con la variable respuesta, pero en el modelo final van a seguir apareciendo. Aunque esto no supone un problema para la precisión del modelo, sí lo es para su interpretación.

Lasso

El método *lasso* es una alternativa al ajuste por *ridge regression* que permite superar su principal desventaja, la incapacidad de excluir predictores del modelo. El método *lasso*, al igual que *ridge regression*, fuerza a que las estimaciones de los coeficientes de los predictores tiendan a cero. La diferencia es que *lasso* sí es capaz de fijar algunos de ellos exactamente a cero, lo que permite además de reducir la varianza, realizar selección de predictores. Como resultado, el método *lasso* tiende a generar modelos más fáciles de interpretar que los obtenidos mediante *ridge regression*. A esto se le conoce como *sparse modeling*.

Comparación entre Lasso y Ridge regression

La superioridad de un método sobre el otro depende del escenario en el que se apliquen, siendo en ocasiones muy similares. Por lo general, cuando solo un pequeño número de predictores de entre todos los incluidos tienen coeficientes estandarizados sustanciales y el resto tienen valores muy pequeños o iguales a cero, *lasso* genera mejores modelos. Si por el contrario, todos los predictores incluidos tienen coeficientes diferentes a cero y aproximadamente de la misma magnitud, *ridge regression* tiende a funcionar mejor.

Selección del *tunning parameter* λ

La precisión del modelo obtenido mediante *lasso* o *ridge regression* depende de la elección del valor λ empleado, ya que determina el grado de penalización. Mediante *Cross-Validation* es posible identificar cual es el valor de λ más adecuado. Para ello se selecciona un rango de valores de λ y se estima el *cross-validation error* resultante para cada uno, finalmente se selecciona el valor de λ para el que el error es menor y se ajusta de nuevo el modelo, esta vez empleando todas las observaciones.

Dimension reduction methods

Los métodos anteriormente descritos controlan la varianza, bien empleando únicamente un subconjunto de predictores o bien haciendo que los coeficientes de regresión tiendan a cero. En ambos casos se emplean las variables originales sin ser modificadas o como máximo habiendo sido estandarizadas.

Las técnicas conocidas como *dimension reduction* crean un número reducido de nuevas variables (componentes) a partir de combinaciones lineales de las variables originales y con ellas se ajusta el modelo. De este modo se consigue generar modelos con menor número de predictores pero que abarcan casi la misma información que la que aportan todas las variables originales. Existen diferentes aproximaciones para lograr este fin, dos de las más utilizadas son: *Principal Components (PCA)* y *Partial Least Square (PLS)*.

Por lo general, ambos métodos tienen buenos resultados en aquellos casos en los que los predictores están altamente correlacionados. Cuando esto ocurre, con pocas componentes principales se puede capturar la mayor parte de la varianza de los predictores así como la relación con la variable respuesta. Es importante tener en cuenta que, aunque permiten generar modelos que contienen un número menor de predictores, no se trata de un método de selección de variables, ya que las nuevas variables (componentes) son combinaciones lineales de todos los predictores originales.

Dado que no dejan de ser un ajuste lineale por mínimos cuadrados que emplean componentes principales como predictores, para que sean válidos se tienen que cumplir las condiciones requeridas para la regresión por mínimos cuadrados.

A continuación se describen de forma muy superficial los métodos *PCR* y *PLS*. Para información más detallada sobre reducción de dimensionalidad consultar Análisis de Componentes Principales (Principal Component Analysis, PCA).

Principal Components Regression PCR

Principal Components Regression consiste en ajustar un modelo de regresión lineal mediante mínimos cuadrados empleando como predictores las componentes generadas a partir de un *Principal Component Analysis (PCA)*. De esta forma, con un número reducido de componentes se puede explicar la mayor parte de la varianza.

Algunas definiciones que permiten entender mejor PCA y PCR:

- Las *principal components* se crean como combinación lineal de las variables originales.
- El vector de la primera componente principal (*first principal component*) define la línea más próxima a todos los puntos.
- La dirección del vector de la primera componente es aquella en la que las observaciones tienen mayor varianza.
- Proyectar un punto en una línea o un plano consiste en encontrar la localización de dicha línea o plano más próxima al punto.
- La segunda componente principal es la combinación lineal de las variables que explica mayor varianza después de la primera componente y que no está correlacionada con esta última. La condición de no correlación entre componentes principales equivale a decir que las direcciones de las componentes son perpendiculares u ortogonales.
- Cuando el número de componentes es igual al número de predictores originales, el resultado de *Principal Components Regression* es equivalente al de regresión por mínimos cuadrados.
- El número óptimo de componentes principales se puede elegir mediante *cross-validation*.
- Cuando se realiza *PCR* hay que estandarizar los predictores antes de calcular las *PCA*, de lo contrario, las variables que se miden en una escala mayor o las que presenten mayor varianza tendrán más peso. Si todos los predictores se miden con la misma escala y presentan la misma varianza, entonces no es necesaria la estandarización.

Partial Least Square PLS

El método *Partial Least Squares (PLS)* es muy similar al *PCR* en cuanto que ambos emplean las componentes principales resultantes de un análisis *PCA* como predictores. La diferencia reside en que, mientras *PCR* ignora la variable respuesta *Y* para determinar las combinaciones lineales, *PLS* busca aquellas que, además de explicar la varianza observada, predicen *Y* lo mejor posible. Puede considerarse como una versión *supervised* de *PCR*. En el libro *ISL* no mencionan ninguna ventaja específica de PLS frente a PCA o viceversa.

Ejemplos con R

Best Subset Selection

En este ejemplo se emplea el set de datos `Hitters` *del paquete* `ISLR` *que contiene 19 variables con información técnica sobre jugadores de béisbol. El objetivo es encontrar el modelo que permita predecir con mayor precisión el salario de un jugador.*

```
library(ISLR)
data("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"
```

```
str(Hitters)
```

```
## 'data.frame':    322 obs. of  20 variables:
##  $ AtBat    : int  293 315 479 496 321 594 185 298 323 401 ...
##  $ Hits     : int  66 81 130 141 87 169 37 73 81 92 ...
##  $ HmRun    : int  1 7 18 20 10 4 1 0 6 17 ...
##  $ Runs     : int  30 24 66 65 39 74 23 24 26 49 ...
##  $ RBI      : int  29 38 72 78 42 51 8 24 32 66 ...
##  $ Walks    : int  14 39 76 37 30 35 21 7 8 65 ...
##  $ Years    : int  1 14 3 11 2 11 2 3 2 13 ...
##  $ CAtBat   : int  293 3449 1624 5628 396 4408 214 509 341 5206 ...
##  $ CHits    : int  66 835 457 1575 101 1133 42 108 86 1332 ...
##  $ CHmRun   : int  1 69 63 225 12 19 1 0 6 253 ...
##  $ CRuns    : int  30 321 224 828 48 501 30 41 32 784 ...
##  $ CRBI     : int  29 414 266 838 46 336 9 37 34 890 ...
##  $ CWalks   : int  14 375 263 354 33 194 24 12 8 866 ...
##  $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
##  $ PutOuts  : int  446 632 880 200 805 282 76 121 143 0 ...
##  $ Assists  : int  33 43 82 11 40 421 127 283 290 0 ...
##  $ Errors   : int  20 10 14 3 4 25 7 9 19 0 ...
##  $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

```
1 - (sum(complete.cases(Hitters))/nrow(Hitters))
```

```
## [1] 0.1832298
```

El *dataset* contiene aproximadamente un 18% de observaciones a las que les falta al menos una variable (*missing value*). Existen técnicas para manejar *missing values* pero en este caso simplemente se van a excluir los registros que no estén completos.

```
datos <- na.omit(Hitters)
dim(datos)
```

```
## [1] 263  20
```

El número de observaciones supera en más de 10 veces al número de predictores. Esto es importante a la hora de elegir el método con el que se va a ajustar el modelo. Como se ha descrito previamente, si el número de predictores se aproxima o supera al de observaciones, la regresión por mínimos cuadrados no es adecuada.

La función `regsubsets()` del paquete `leaps` realiza *best subset selection* de modelos ajustados mediante regresión lineal por mínimos cuadrados. En primer lugar identifica el mejor modelo de cada tamaño (mejor modelo con 1 predictor, mejor modelo con 2 predictores...), entendiendo por mejor modelo aquel que tiene menor *RSS*.

```
require(leaps)
mejores_modelos <- regsubsets(Salary ~ ., data = datos, nvmax = 19)
# El argumento nvmax determina el tamaño máximo de los modelos a
# inspeccionar.  Si se quiere realizar best subset selection evaluando todos
# los posibles modelos, nvmax tiene que ser igual al número de variables
# disponibles
```

La función `summary()` aplicada a un objeto `regsubsets()` devuelve una tabla en la que se muestra, para cada posible tamaño de modelo, cual es el mejor. El nombre de la fila indica el tamaño del modelo (el número de predictores que contiene). El "∗" indica qué variables se han incluido. Por ejemplo, el mejor modelo formado por dos predictores incorpora las variables *Hits* y *CRBI*.

```
summary(mejores_modelos)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = datos, nvmax = 19)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
## 9  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
```

```
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"
##           CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  "*"  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  "*"  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  "*"  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  "*"  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 )  "*"  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  "*"  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  " "  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  " "  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

Una vez identificado el mejor modelo de cada tamaño, se tiene que escoger el mejor de entre todos ellos. La función `regsubsets()` también devuelve los valores de R^2, RSS, $R^2_{ajustado}$, Cp y BIC.

```
names(summary(mejores_modelos))
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
summary(mejores_modelos)$adjr2
```

```
##  [1] 0.3188503 0.4208024 0.4450753 0.4672734 0.4808971 0.4972001 0.5007849
##  [8] 0.5137083 0.5180572 0.5222606 0.5225706 0.5217245 0.5206736 0.5195431
## [15] 0.5178661 0.5162219 0.5144464 0.5126097 0.5106270
```

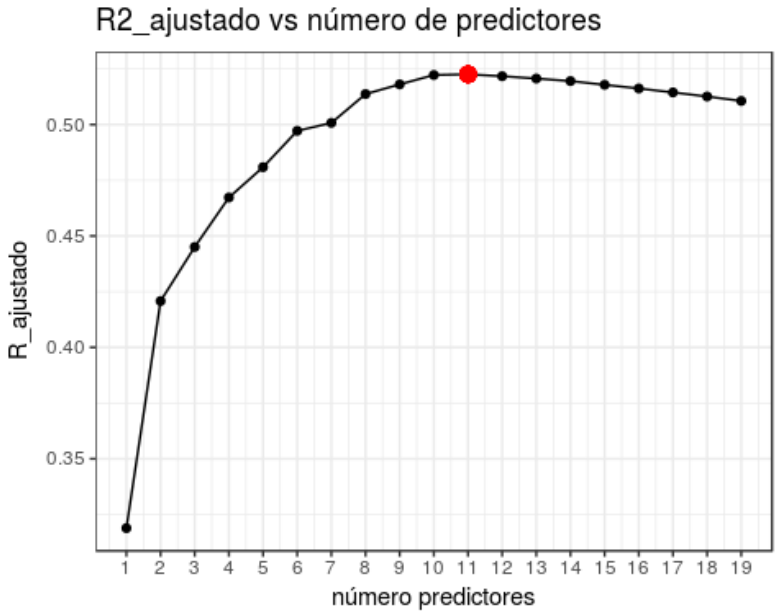
```
# se identifica que modelo tiene el valor máximo de R ajustado
which.max(summary(mejores_modelos)$adjr2)
```

```
## [1] 11
```

El modelo que ocupa la posición 11 es el que mayor $R^2_{ajustado}$ alcanza. Dado que las posiciones en la tabla creada por `regsubsets()` se corresponden con el número de predictores, el mejor modelo es el que contiene 11 predictores.

Una representación gráfica del estadístico escogido para comparar los modelos, en este caso $R^2_{ajustado}$, frente al número de predictores permite evaluar la evolución de la precisión del modelo en función del tamaño y si la mejora es sustancial.

```
library(ggplot2)
p <- ggplot(data = data.frame(n_predictores = 1:19,
                              R_ajustado = summary(mejores_modelos)$adjr2),
            aes(x = n_predictores, y = R_ajustado)) +
     geom_line() +
     geom_point()

# Se identifica en rojo el máximo
p <- p + geom_point(aes(x=n_predictores[which.max(summary(mejores_modelos)$adjr2)],
                        y=R_ajustado[which.max(summary(mejores_modelos)$adjr2)]),
                        colour = "red", size = 3)
p <- p + scale_x_continuous(breaks = c(0:19)) +
     theme_bw() +
     labs(title = "R2_ajustado vs número de predictores", x = "número predictores")
p
```



Para conocer cuáles son los coeficientes del mejor modelo y su estimación:

```
coef(object = mejores_modelos, id = 11)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat
##  135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914
##        CRuns         CRBI       CWalks      LeagueN    DivisionW
##    1.4553310    0.7852528   -0.8228559   43.1116152 -111.1460252
##      PutOuts      Assists
##    0.2894087    0.2688277
```

Si bien el modelo con mayor $R^2_{ajustado}$ es el formado por 11 predictores, la mejora conseguida a partir de 6 predictores es mínima:

```
summary(mejores_modelos)$adjr2[11]
```

```
## [1] 0.5225706
```

```
summary(mejores_modelos)$adjr2[6]
```

```
## [1] 0.4972001
```

Acorde al principio de parsimonia, el modelo que se debe seleccionar como adecuado es el formado por entre 6 y 8 predictores.

Forward and Backward Stepwise Selection

Del mismo modo que se realiza *best subset selection*, con la función `regsubsets()` se puede realizar *stepwise selection* en ambos sentidos (*forward* o *backward*) indicándolo en el argumento `method`.

Backward

```
require(leaps)
mejores_modelos_backward <- regsubsets(Salary ~ ., data = datos, nvmax = 19,
                                       method = "backward")
# se identifica el valor máximo de R ajustado
which.max(summary(mejores_modelos_backward)$adjr2)
```

```
## [1] 11
```

```
coef(object = mejores_modelos_backward, 11)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat
##  135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914
##        CRuns         CRBI       CWalks      LeagueN    DivisionW
##    1.4553310    0.7852528   -0.8228559   43.1116152 -111.1460252
##      PutOuts      Assists
##    0.2894087    0.2688277
```

Forward

```
mejores_modelos_forward <- regsubsets(Salary ~ ., data = datos, nvmax = 19,
    method = "forward")
# se identifica el valor máximo de R ajustado
which.max(summary(mejores_modelos_forward)$adjr2)
```

```
## [1] 11
```

```
coef(object = mejores_modelos_forward, 11)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat
##  135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914
##        CRuns         CRBI       CWalks      LeagueN    DivisionW
##    1.4553310    0.7852528   -0.8228559   43.1116152 -111.1460252
##      PutOuts      Assists
##    0.2894087    0.2688277
```

Ambos métodos (*backward* y *forward*) identifican como mejor modelo el formado por los mismos 11 predictores.

Validation set y K-Cross-Validation

Simple validation set

El primer paso del método *validation set* requiere dividir aleatoriamente las observaciones disponibles en *training set* y *test set*. En R se pueden conseguir una división aleatoria empleando índices aleatorios.

```
set.seed(1)
datos <- na.omit(Hitters)
# Se emplean como training aproximadamente 2/3 de las observaciones. Se seleccionan
# índices aleatorios que forman el training dataset
train <- sample(x = 1:263, size = 180, replace = FALSE)
# Los restantes forman el test dataset
```

Empleando el *training dataset* se identifica el mejor modelo para cada posible tamaño. En este paso pueden estudiarse todas las posibles combinaciones de predictores (*best subset selection*) o solo parte de ellos (*stepwise selection*).

```
library(leaps)
# En este caso se emplea forward stepwise selection, pero podrían emplearse
# los otros métodos.  Dado que hay 19 predictores disponibles y se quieren
# estudiar todos los posibles tamaños de modelo, nvmax=19
mejores_modelos <- regsubsets(Salary ~ ., data = datos[train, ], nvmax = 19,
                              method = "forward")
# Mejores_modelos almacena los 19 modelos seleccionados
```

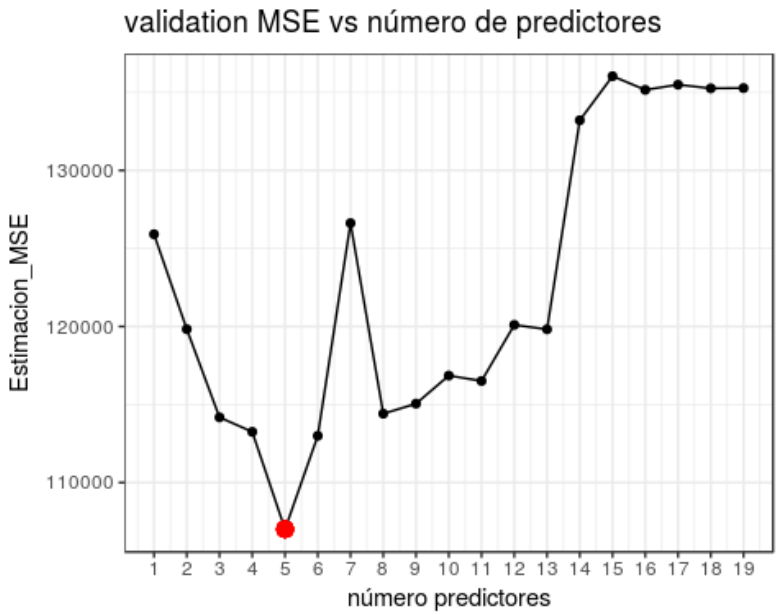
Como resultado del proceso de evaluación de `regsubsets()` se han seleccionado 19 modelos, el mejor para cada tamaño. Para poder compararlos se procede a estimar el *validation test error* empleando las observaciones que se han excluido del *training* y que se han designado como *test*.

```
# Se genera un vector que almacenará el test-error de cada modelo
validation_error <- rep(NA, 19)

# No existe una función predefinida para obtener el validation-test-error,
# se tiene que programar.  Para facilitar el proceso es conveniente trabajar
# con matrices. Los valores de los predictores de cada observación se
# almacenan en forma de matriz. La función model.matrix() devuelve una
# matriz formada con los predictores indicados en la fórmula e introduce
# para todas las observaciones un intercept con valor 1, así al multiplicar
# por los coeficientes se obtiene el valor de la predicción (producto matricial).
```

```
test_matrix <- model.matrix(Salary ~ ., data = datos[-train, ])

# Para cada uno de los modelos almacenados en la variable mejores modelos:
for (i in 1:19) {

    # Se extraen los coeficientes del modelo
    coeficientes <- coef(object = mejores_modelos, id = i)

    # Se identifican los predictores que forman el modelo y se extraen de la
    # matriz modelo
    predictores <- test_matrix[, names(coeficientes)]

    # Se obtienen las predicciones mediante el producto matricial de los
    # predictores extraídos y los coeficientes del modelo
    predicciones <- predictores %*% coeficientes

    # Finalmente se calcula la estimación del test error como el promedio de los
    # residuos al cuadrado (MSE)
    validation_error[i] <- mean((datos$Salary[-train] - predicciones)^2)
}

which.min(validation_error)
```

```
## [1] 5
```

```
# Gráfico
p <- ggplot(data = data.frame(n_predictores = 1:19,
                              Estimacion_MSE = validation_error),
                    aes(x = n_predictores, y = Estimacion_MSE)) +
    geom_line() +
    geom_point()

# Se identifica en rojo el mínimo
p <- p + geom_point(aes(x = n_predictores[which.min(validation_error)],
                        y = validation_error[which.min(validation_error)]),
                        colour = "red", size = 3)
p <- p + scale_x_continuous(breaks = c(0:19)) +
        labs(title = "validation MSE vs número de predictores",
             x = "número predictores") +
        theme_bw()
p
```



El modelo con menor *validation test error* es el que contiene 5 predictores. Se puede considerar que 5 es el número de predictores que debe contener el modelo para alcanzar la mayor precisión predictiva. Por último, una vez identificada la cantidad de predictores que debe contener el modelo y para obtener las estimaciones de los coeficientes de la forma más precisa posible, se vuelven a ajustar los posibles modelos con 5 predictores empleando todas las observaciones (*training* + *test*). Es importante volver a seleccionar el mejor de entre los modelos con 5 predictores ya que al incluir nuevas observaciones puede que esos 5 predictores no sean los mismos que los seleccionados previamente con el *training dataset*.

```
mejores_modelos <- regsubsets(Salary ~ ., data = datos, nvmax = 19,
                              method = "forward")
coef(object = mejores_modelos, id = 5)
```

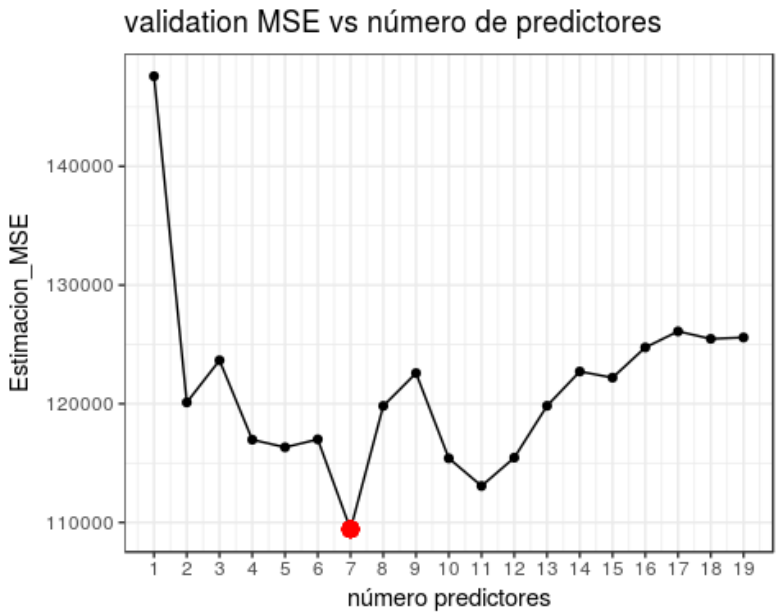
```
##  (Intercept)        AtBat         Hits         CRBI    DivisionW
##   97.7684116   -1.4401428    7.1753197    0.6882079 -129.7319386
##      PutOuts
##    0.2905164
```

A pesar de que la validación simple es un método muy sencillo, tiene el inconveniente de sufrir mucha varianza. Los resultados pueden variar mucho dependiendo de cómo se repartan las observaciones entre *tranining set* y *test set*. Por ejemplo, si se establece `set.seed(1388)`, el reparto generado da como resultado que el mejor modelo es el formado por 7 predictores.

```
set.seed(1388)
datos <- na.omit(Hitters)
train <- sample(x = 1:263, size = 180, replace = FALSE)
mejores_modelos <- regsubsets(Salary ~ ., data = datos[train, ], nvmax = 19,
                              method = "forward")
validation_error <- rep(NA, 19)
test_matrix <- model.matrix(Salary ~ ., data = datos[-train, ])
for (i in 1:19) {
    coeficientes <- coef(object = mejores_modelos, id = i)
    predictores <- test_matrix[, names(coeficientes)]
    predicciones <- predictores %*% coeficientes
    validation_error[i] <- mean((datos$Salary[-train] - predicciones)^2)
}
which.min(validation_error)
```

```
## [1] 7
```

```
p <- ggplot(data = data.frame(n_predictores = 1:19,
                              Estimacion_MSE = validation_error),
            aes(x = n_predictores, y = Estimacion_MSE)) +
     geom_line() +
     geom_point()
p <- p + geom_point(aes(x = n_predictores[which.min(validation_error)],
                        y = validation_error[which.min(validation_error)]),
                        colour = "red", size = 3)
p <- p + scale_x_continuous(breaks = c(0:19)) +
         theme_bw() +
         labs(title = "validation MSE vs número de predictores",
              x = "número predictores")
p
```



k-Cross-Validation

El primer paso en el proceso de *K-Cross-Validation* es dividir las observaciones de forma aleatoria en *k* grupos de aproximadamente el mismo tamaño. En este caso se va a emplear un valor de *k=10*.

```
library(ISLR)
library(leaps)
datos <- na.omit(Hitters)
set.seed(11)
# Sample() mezcla aleatoriamente las posiciones.
# Es importante que la asignación sea aleatoria.
grupo <- sample(rep(x = 1:10, length = nrow(datos)))
# Se comprueba que la distribución es aproximadamente equitativa
table(grupo)
```

```
## grupo
##  1  2  3  4  5  6  7  8  9 10
## 27 27 27 26 26 26 26 26 26 26
```

A continuación se inicia un proceso iterativo con *k* ciclos en el que:

1. Se identifica el mejor modelo para cada tamaño (1 predictor,..., 19 predictores), empleando como *training set* las observaciones de todos los *k* grupos excepto uno y evaluándolos acorde a menor *RSS*. Esto puede hacerse empleando la función `regsubsets()`.
2. Se estima y almacena el *test error (MSE)* para cada uno de los modelos seleccionados empleando las observaciones del grupo que se excluyó en el paso anterior.
3. Se repite el proceso *k* veces, excluyendo en cada iteración un grupo distinto del *training set*.
4. Se calcula el promedio de los *k test error (MSE)* para cada tamaño de modelo. A este valor promedio se le conoce como estimación del M*ean-Cross-Validation Test Error*. Cuanto menor sea, mejor precisión predictiva tiene el modelo.
5. Identificación del tamaño de modelo que consigue el menor *mean-Cross-validation test error*.
6. Reajuste e identificación del mejor modelo con el número de predictores obtenido en el paso 5, empleando todas las observaciones como *training*. Esto puede hacerse empleando la función `regsubsets()`.

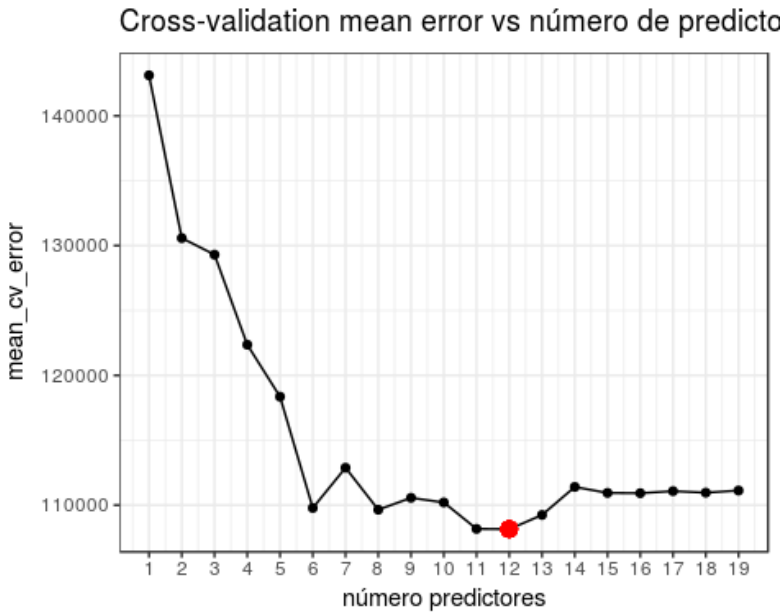
En este ejemplo, dado que hay 19 modelos y 10 grupos de validación (*k*), para cada uno de los 19 modelos se calculan 10 estimaciones de *test error* haciendo un total de 190 valores. La mejor forma de almacenarlos es en formato de matriz. El mejor modelo de todos ellos será aquel que tenga un promedio de error menor (*mean cv-test error*).

```
# Función descrita en libro ISLR p.250 Dado un objeto creado por la función
# regsubsets(), que es una lista de modelos, y un nuevo set de
# observaciones, la función predict.regsubsets() devuelve las predicciones
# para cada uno de los modelos.

predict.regsubsets <- function(object, newdata, id, ...) {
    # Extraer la fórmula del modelo (variable dependiente ~ predictores)
    form <- as.formula(object$call[[2]])
    # Generar una matriz modelo con los nuevos datos y la fórmula
    mat <- model.matrix(form, newdata)
    # Extraer los coeficientes del modelo
    coefi <- coef(object, id = id)
    # Almacenar el nombre de las variables predictoras del modelo
    xvars <- names(coefi)
    # Producto matricial entre los coeficientes del modelo y los valores de los
    # predictores de las nuevas observaciones para obtener las predicciones
    mat[, xvars] %*% coefi
}

# Matriz que almacena los test-error estimados. Cada columna representa un
# modelo. Cada fila representa uno de los 10 grupos en los que se han dividido las
# observaciones
```

```
error_matrix <- matrix(data = NA, nrow = 10, ncol = 19,
                       dimnames = list(NULL, c(1:19)))

# Loop en el que se excluye en cada iteración un grupo distinto
# ESTE LOOP ESTA HECHO PARA UN DATA FRAME CON 19 PREDICTORES
num_validaciones <- 10
num_predictores <- 19

for (k in 1:num_validaciones) {
    # Identificación de datos empleados como training
    train <- datos[grupo != k, ]
    # Selección de los mejores modelos para cada tamaño basándose en RSS
    mejores_modelos <- regsubsets(Salary ~ ., data = train, nvmax = 19,
                                  method = "forward")

    # Para cada uno de los modelos 'finalistas' se calcula el test-error con el
    # grupo excluido
    for (i in 1:num_predictores) {
        test <- datos[grupo == k, ]
        # Las predicciones del modelo i almacenado en el objeto regsubsets se
        # extraen mediante la función predict.regsubsets() definida arriba
        predicciones <- predict.regsubsets(object = mejores_modelos,
                                           newdata = test, id = i)
        # Cálculo y almacenamiento del MSE para el modelo i
        error_matrix[k, i] <- mean((test$Salary - predicciones)^2)
    }
}
# Cada columna de la matriz error_matrix contiene los 10 valores de error
# calculados por cv
mean_cv_error <- apply(X = error_matrix, MARGIN = 2, FUN = mean)
# plot(sqrt(mean_cv_error), type = 'b', pch = 19)
which.min(x = mean_cv_error)
```

```
## 12
```

```
ggplot(data = data.frame(n_predictores = 1:19, mean_cv_error = mean_cv_error),
              aes(x = n_predictores, y = mean_cv_error)) +
 geom_line() +
 geom_point() +
 geom_point(aes(x = n_predictores[which.min(mean_cv_error)],
                y = mean_cv_error[which.min(mean_cv_error)]),
                colour = "red", size = 3) +
 scale_x_continuous(breaks = c(0:19)) +
 theme_bw() +
 labs(title = "Cross-validation mean error vs número de predictores",
      x = "número predictores")
```



El mejor modelo identificado mediante 10-Cross-Validation es el formado por 12 predictores. Finalmente se identifica el mejor modelo formado por 12 predictores empleando todas las observaciones (*training* + *test*).

```
modelo_final <- regsubsets(Salary ~ ., data = datos, nvmax = 19, method ="forward")
coef(object = modelo_final, 12)
```

```
##  (Intercept)        AtBat         Hits         Runs        Walks
##  135.5194919   -2.0563475    7.5064072   -1.7965622    6.0619776
##       CAtBat        CRuns         CRBI       CWalks      LeagueN
##   -0.1524448    1.5589219    0.7775813   -0.8350722   39.0865444
##    DivisionW      PutOuts      Assists
## -112.6442519    0.2842332    0.2434442
```

Si bien el modelo con 12 predictores es el que menor *cv test error* estimado tiene, el gráfico muestra que a partir de 6 predictores la mejora es mínima. Acorde al principio de parsimonia, según el cual se recomienda emplear de entre los modelos buenos el más simple, el modelo más adecuado es el de 6 predictores.

Ridge regression

De nuevo, se pretende crear un modelo que permita predecir el salario de los jugadores empleando las variables disponibles en el *dataset* `Hitters`.

Para realizar *ridge regression* se va a emplear la función `glmnet()` del paquete `glmnet`. Esta función se caracteriza por no recibir como argumento una función `~` sino una matriz modelo `x` que contiene el valor de los predictores para cada observación y un vector `y` que contiene la variable respuesta. La función `model.matrix()` permite crear de forma rápida una matriz modelo a partir de un *data frame*, identificando los predictores y generando las variables *dummy* necesarias en caso de que los predictores sean cualitativos.

El objeto devuelto por la función `glmnet()` contiene toda la información relevante del o de los modelos ajustados. Además, el paquete incorpora funciones que permiten extraer dicha información de forma eficiente: `plot()`, `print()`, `coef()` y `predict()`.

Más información del paquete `glmnet` en:

https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html

```
datos <- na.omit(Hitters)
x <- model.matrix(Salary ~ ., data = datos)[, -1]
head(x)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
## -Alan Ashby         315   81     7   24  38    39    14   3449   835
## -Alvin Davis        479  130    18   66  72    76     3   1624   457
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575
## -Andres Galarraga   321   87    10   39  42    30     2    396   101
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133
## -Al Newman          185   37     1   23   8    21     2    214    42
##                   CHmRun CRuns CRBI CWalks LeagueN DivisionW PutOuts
## -Alan Ashby           69   321  414    375       1         1     632
## -Alvin Davis          63   224  266    263       0         1     880
## -Andre Dawson        225   828  838    354       1         0     200
## -Andres Galarraga     12    48   46     33       1         0     805
## -Alfredo Griffin      19   501  336    194       0         1     282
## -Al Newman             1    30    9     24       1         0      76
##                   Assists Errors NewLeagueN
## -Alan Ashby            43     10          1
## -Alvin Davis           82     14          0
## -Andre Dawson          11      3          1
## -Andres Galarraga      40      4          1
## -Alfredo Griffin      421     25          0
## -Al Newman            127      7          0
```

```
y <- datos$Salary
```

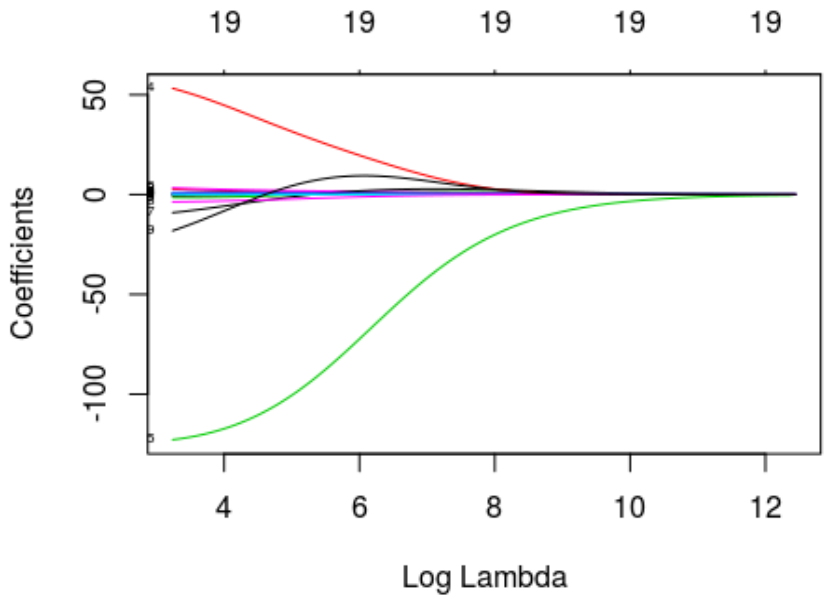
El resultado de un ajuste por *ridge regression* depende del *tunning parameter* λ que determina el grado de penalización. Mediante el argumento `lambda` se puede especificar un valor concreto obteniendo un único modelo. Si no se tiene conocimiento previo de qué valor de λ es el adecuado, se puede abarcar el rango 10^{10} a 10^{-2}, que va desde un modelo muy estricto que no contiene ningún predictor, hasta uno sin penalización equivalente al ajuste por mínimos cuadrados.

La función `glmnet()` estandariza por defecto las variables antes de realizar el ajuste del modelo.

```
library(glmnet)
# Para obtener un ajuste mediante ridge regression se indica argumento alpha=0.
modelos_ridge <- glmnet(x = x, y = y, alpha = 0)
```

`glmnet()` almacena en una matriz el valor de los coeficientes de regresión de los predictores para cada valor de λ. Esto permite acceder mediante la función `coef()` a los coeficientes resultantes para un determinado valor de λ (que haya sido incluido en el rango cuando se han generado los modelos) y también para graficar la evolución de los coeficientes a medida que se incrementa λ.

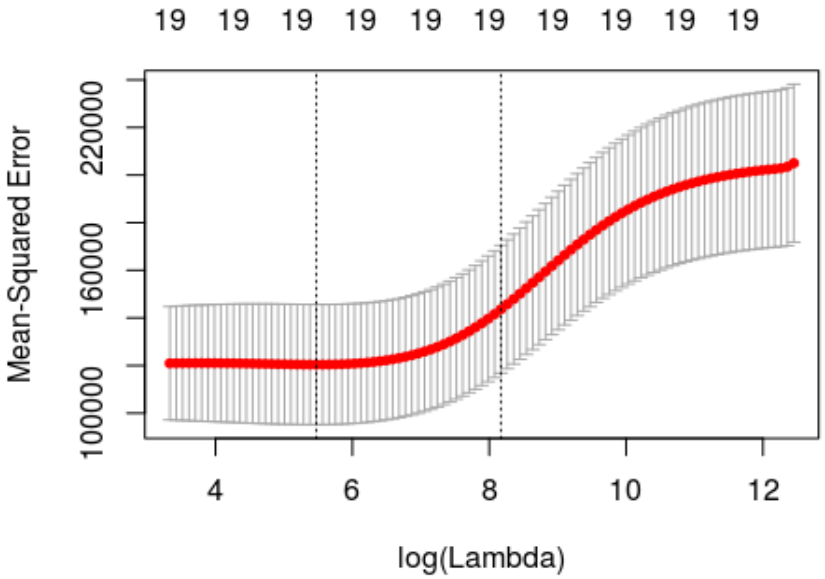
```
plot(modelos_ridge, xvar = "lambda", label = TRUE)
```



Como es de esperar, los coeficientes se van haciendo más pequeños a medida que se incrementa el valor de λ.

Con el fin de identificar el valor de λ que da lugar al mejor modelo, se puede recurrir a *Cross-Validation*. La función `cv.glmnet()` calcula el *cv-test-error*, utilizando por defecto *k=10*.

```
set.seed(1)
# x e y son la matriz modelo y el vector respuesta creados anteriormente con
# los datos de Hitters
cv_error_ridge <- cv.glmnet(x = x, y = y, alpha = 0, nfolds = 10,
                            type.measure = "mse")
plot(cv_error_ridge)
```



El gráfico muestra el *cv-test-error (Mean Square Error)* para cada valor de λ junto con la barra de error correspondiente. Entre la información almacenada en el objeto devuelto por la función `cv.glmnet()` se encuentra el valor de λ con el que se consigue el menor *cv-test error* y el valor de λ con el que se consigue el modelo más sencillo que se aleja menos de 1 desviación estandar del mínimo *cv-test-error* posible.

```
# Valor lambda con el que se consigue el mínimo test-error
cv_error_ridge$lambda.min
```

```
## [1] 238.0769
```

```
# Valor lambda óptimo: mayor valor de lambda con el que el test-error no se
# aleja más de 1 sd del mínimo test-error posible.
cv_error_ridge$lambda.1se
```

```
## [1] 3535.367
```

Acorde al principio de parsimonia y la norma de *one standard error rule*, el mejor modelo es el que se obtiene con $\lambda = 3535.367$. `lambda.1se` siempre es mayor que `lambda.min`.

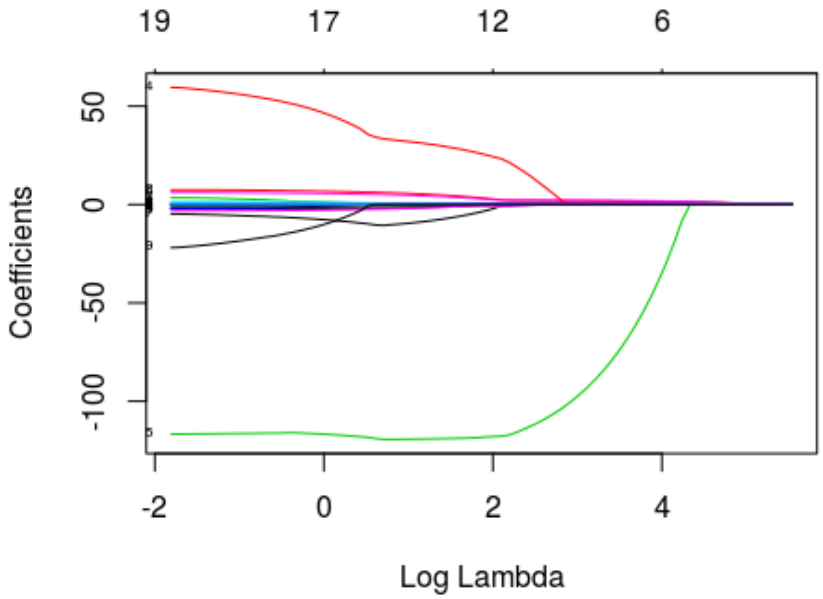
```
# Se muestra el valor de los coeficientes para el valor de lambda óptimo
modelo_final_ridge <- glmnet(x = x, y = y, alpha = 0, lambda = 3535.367)
coef(modelo_final_ridge)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## (Intercept) 254.528726621
## AtBat         0.079397926
## Hits          0.317341743
## HmRun         1.064954804
## Runs          0.515808839
## RBI           0.520487677
## Walks         0.661883701
## Years         2.231109592
## CAtBat        0.006679349
## CHits         0.025457534
## CHmRun        0.189674596
## CRuns         0.051060307
## CRBI          0.052777074
## CWalks        0.052169935
## LeagueN       2.114763532
## DivisionW   -17.479810164
## PutOuts       0.043040055
## Assists       0.006296905
## Errors       -0.099466782
## NewLeagueN    2.085830168
```

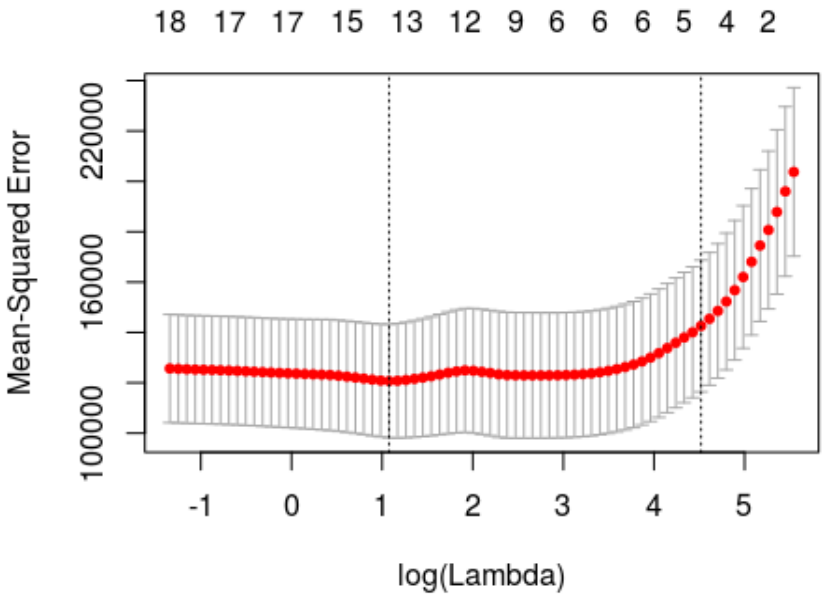
Lasso

El proceso para realizar un ajuste mediante *lasso* y la identificación del mejor valor de λ es equivalente al seguido en el caso de *ridge regression* pero indicando en la función `glmnet()` que `alpha=1`.

```
library(glmnet)
# x e y son la matriz modelo y el vector respuesta creados anteriormente con
# los datos de Hitters
modelos_lasso <- glmnet(x = x, y = y, alpha = 1)
plot(modelos_lasso, xvar = "lambda", label = TRUE)
```




```
set.seed(1)
cv_error_lasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = 10)
plot(cv_error_lasso)
```




```
cv_error_lasso$lambda.min
```

```
## [1] 2.935124
```

```
cv_error_lasso$lambda.1se
```

```
## [1] 91.74363
```

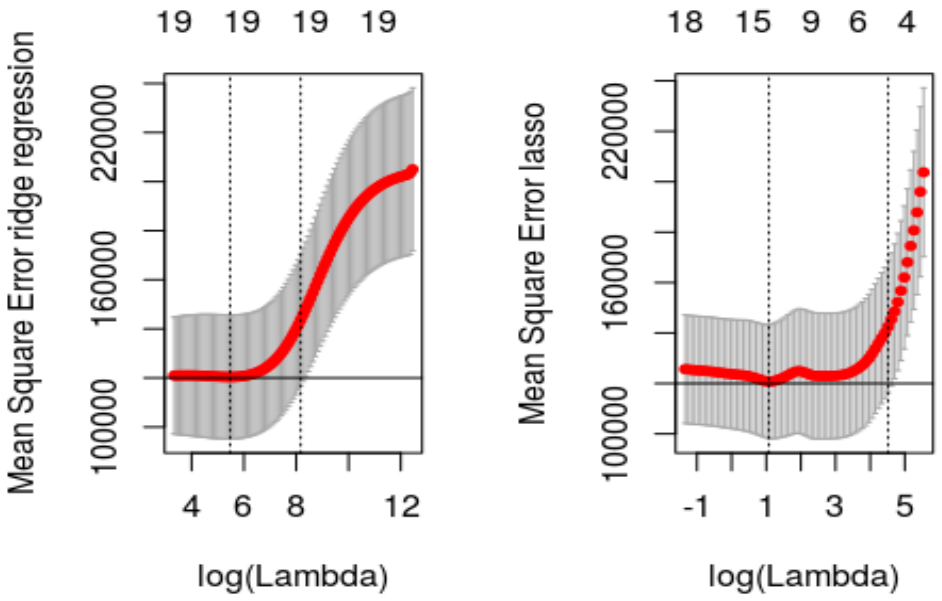
```
# Se reajusta el modelo con todas las observaciones empleando el valor de
# lambda óptimo
modelo_final_lasso <- glmnet(x=x, y=y, alpha=1,lambda=cv_error_lasso$lambda.1se)
coef(modelo_final_lasso)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) 193.91035035
## AtBat         .
## Hits          1.21576856
## HmRun         .
## Runs          .
## RBI           .
## Walks         1.29215655
## Years         .
## CAtBat        .
## CHits         .
## CHmRun        .
## CRuns         0.12267555
```

```
## CRBI          0.32146185
## CWalks        .
## LeagueN       .
## DivisionW     .
## PutOuts       0.02499026
## Assists       .
## Errors        .
## NewLeagueN    .
```

Tanto el método *ridge regression* como el de *lasso* consiguen, empleando sus respectivos valores óptimos de λ, reducir el *MSE (test error)* a unos niveles muy parecidos. La ventaja del modelo final obtenido por *lasso* es que es mucho más simple ya que contiene únicamente 5 predictores.

```
par(mfrow = c(1, 2))
plot(cv_error_ridge, ylab = "Mean Square Error ridge regression")
abline(h = 120000)
plot(cv_error_lasso, ylab = "Mean Square Error lasso")
abline(h = 120000)
```



Principal Component regression PCR

De nuevo, se pretende crear un modelo que permita predecir el salario de los jugadores empleando las variables disponibles en el *dataset* `Hitters`.

Los modelos de regresión basados en *Principal Components* pueden ajustarse mediante la función `pcr()` del paquete `pls`.

```
library(pls)
datos <- na.omit(Hitters)
set.seed(2)
# Importante estandarizar las variables indicándolo con el argumento scale=TRUE
# Indicando validation = CV, se emplea 10-fold-cross-validation para
# identificar el número óptimo de componentes.
modelo_pcr <- pcr(Salary ~ ., data = datos, scale = TRUE, validation = "CV")
summary(modelo_pcr)
```

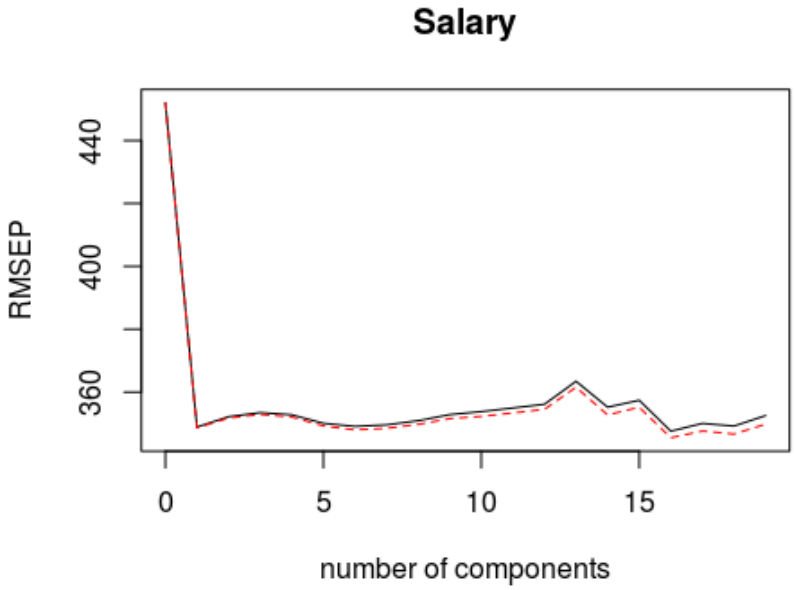
```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV             452    348.9    352.2    353.5    352.8    350.1    349.1
## adjCV          452    348.7    351.8    352.9    352.1    349.3    348.0
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       349.6    350.9    352.9     353.8     355.0     356.2     363.5
## adjCV    348.5    349.8    351.6     352.3     353.4     354.5     361.6
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        355.2     357.4     347.6     350.1     349.2     352.6
## adjCV     352.8     355.2     345.5     347.6     346.7     349.8
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69
##         8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X         94.96    96.28     97.26     97.98     98.65     99.15     99.47
## Salary    46.75    46.86     47.76     47.82     47.85     48.10     50.40
##         15 comps  16 comps  17 comps  18 comps  19 comps
## X          99.75     99.89     99.97     99.99    100.00
## Salary     50.55     53.01     53.85     54.61     54.61
```

El `summary` del modelo `pcr` devuelve la estimación del *RMSEP* (raíz cuadrada del *MSE*) para cada posible número de componentes introducidas en el modelo. También se muestra el %

de varianza explicada acumulada por cada número de componentes. Si se incluye hasta la 6 componente, se explica un 88.63% de la varianza observada.

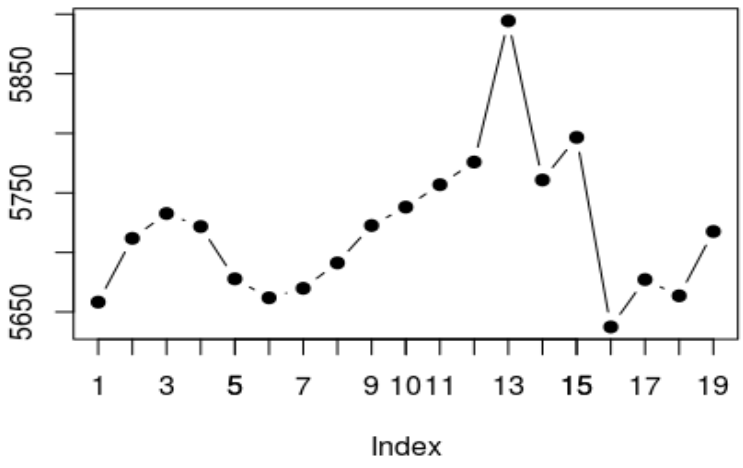
Es posible obtener una representación gráfica con la función `validationplot()` que facilite la identificación del número de componentes óptimo que debe contener el modelo.

```
validationplot(modelo_pcr, val.type = "RMSEP")
```




```
# Para ver con más detalle a partir de la componente1
#PRESS es el Predicted Sum of Squares
plot(as.numeric(sqrt(modelo_pcr$validation$PRESS)), type = "b", pch = 19)
axis(side = 1, at = 1:19)
```

$\sqrt{PRESS}$



```
# Para conocer el número de componentes con el que se minimiza el error
which.min(x = modelo_pcr$validation$PRESS)
```

```
## [1] 16
```

Si bien el menor error se alcanza con 16 componentes, se observa que a partir de la primera componente la mejora es mínima. El modelo que incluye únicamente la primera componente cumple el principio de parsimonia.

Comparación entre métodos

Tal como se ha ido describiendo a lo largo de este capítulo, no existe un método que por defecto supere a los otros. Dependiendo del escenario (pocos predictores importantes, muchos predictores importantes, muchos predictores correlacionados...) un método puede superar de forma sustancial a los otros, o puede que todos alcancen aproximadamente la misma precisión.

La forma de identificar cuál es el mejor método para un determinado estudio consiste en dividir las observaciones disponibles en dos grupos (*trainig* y *test*). Siguiendo los pasos de cada uno de los métodos se ajusta el modelo empleando únicamente el *trainig set* y se calcula el *MSE* (`mean(predicción - valor real)^2`) utilizando el *test set*. Aquel método con el que se obtenga menor *MSE* es el que mejor precisión logra.

Ejemplo: Caso de estudio

Se dispone del set de datos `Hitters` que contiene 19 variables sobre jugadores de la liga de béisbol. Se quiere crear un modelo lineal múltiple que permita predecir el salario de los jugadores. Se van a comparar los siguientes métodos de regresión: ordinary least squares (OLS) con todos los predictores, subset selection, ridge regression, lasso y PCR.

En primer lugar se divide aleatoriamente el set de datos en dos grupos, uno se empleará para entrenar los modelos y el otro para validarlos.

```
require(ISLR)
data("Hitters")
Hitters <- na.omit(Hitters)
dim(Hitters)
```

```
## [1] 263  20
```

```
set.seed(1)
indices_entrenamiento <- sample(x = 1:nrow(Hitters),
                                size = round(nrow(Hitters) * (2/3)))
# 2/3 de las observaciones
indices_test <- (1:nrow(Hitters))[-indices_entrenamiento]
Hitters_1 <- Hitters[indices_entrenamiento, ]
Hitters_2 <- Hitters[indices_test, ]
```

Ordinary least square (regresión por mínimos cuadrados)

```
modelo_OLS <- lm(formula = Salary ~ ., data = Hitters_1)
test_MSE_OLS <- mean((predict(modelo_OLS, Hitters_2) - Hitters_2$Salary)^2)
test_MSE_OLS
```

```
## [1] 129504
```

Best subset selection mediante k-cross-validation

```
set.seed(3553)
require(leaps)
require(ggplot2)
grupo <- sample(rep(x = 1:10, length = nrow(Hitters_1)))
# Se comprueba que la distribución es aproximadamente equitativa
table(grupo)
```

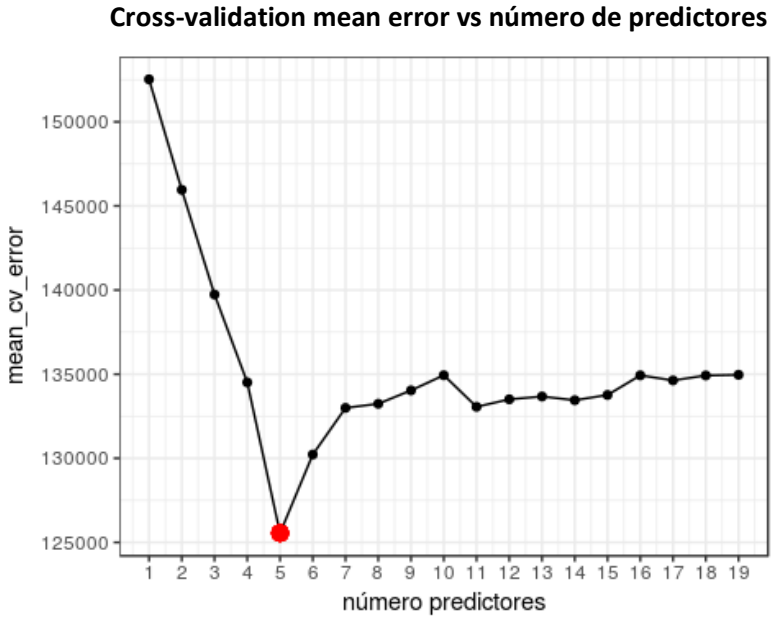
```
## grupo
##  1  2  3  4  5  6  7  8  9 10
## 18 18 18 18 18 17 17 17 17 17
```

```
predict.regsubsets <- function(object, newdata, id, ...) {
    # Extraer la fórmula del modelo (variable dependiente ~ predictores)
    form <- as.formula(object$call[[2]])
    # Generar una matriz modelo con los nuevos datos y la fórmula
    mat <- model.matrix(form, newdata)
    # Extraer los coeficientes del modelo
    coefi <- coef(object, id = id)
```

```
    # Almacenar el nombre de las variables predictoras del modelo
    xvars <- names(coefi)
    # Producto matricial entre los coeficientes del modelo y los valores de los
    # predictores de las nuevas observaciones para obtener las predicciones
    mat[, xvars] %*% coefi
}

# Matriz que almacena los test-error estimados. Cada columna representa un
# modelo Cada fila es uno de los 10 grupos en los que se han dividido las
# observaciones
error_matrix <- matrix(data = NA, nrow = 10, ncol = 19, dimnames = list(NULL,
                       c(1:19)))

# Loop en el que se excluye en cada iteración un grupo distinto
for (k in 1:10) {
    # Identificación de Hitters empleados como training
    train <- Hitters_1[grupo != k, ]
    # Selección de los mejores modelos para cada tamaño basándose en RSS
    mejores_modelos <- regsubsets(Salary ~ ., data = train, nvmax = 19, method =
                        "backward")
    # Para cada uno de los modelos 'finalistas' se calcula el test-error con el
    # grupo excluido
    for (i in 1:19) {
        test <- Hitters_1[grupo == k, ]
        # Las predicciones del modelo i almacenado en el objeto regsubsets se
        # extraen mediante la función predict.regsubsets() definida arriba
        predicciones <- predict.regsubsets(object = mejores_modelos, newdata =
                                           test, id = i)
        # Cálculo y almacenamiento del MSE para el modelo i
        error_matrix[k, i] <- mean((test$Salary - predicciones)^2)
    }
}

mean_cv_error <- apply(X = error_matrix, MARGIN = 2, FUN = mean)
which.min(x = mean_cv_error)
```

```
## 5
```

```
ggplot(data = data.frame(n_predictores = 1:19, mean_cv_error = mean_cv_error),
       aes(x = n_predictores, y = mean_cv_error)) + geom_line() + geom_point() +
geom_point(aes(x = n_predictores[which.min(mean_cv_error)],
               y = mean_cv_error[which.min(mean_cv_error)]),
           colour = "red", size = 3) +
scale_x_continuous(breaks = c(0:19)) + theme_bw() +
labs(title = "Cross-validation mean error vs número de predictores",
     x = "número predictores")
```



El mejor modelo identificado mediante 10-Cross-Validation es el formado por 5 predictores. Finalmente se identifica el mejor modelo formado por 5 predictores empleando todas las observaciones de Hitters_1 y se calcula el *test-MSE* empleando el set de datos Hitters_2.

```
modelo_final <- regsubsets(Salary ~ ., data = Hitters_1, nvmax = 19,
                           method = "backward")
coef(object = modelo_final, 5)
```

```
##  (Intercept)        Walks        CRuns       CWalks    DivisionW
##   63.6749055    6.2709156    1.1857930   -0.8033519 -159.8153519
##      PutOuts
##    0.3716267
```

```
# Como el modelo está dentro de un objeto regsubsets, en predict() se tiene
# que identificar el id
test_MSE_subset <- mean((predict(modelo_final, Hitters_2, id = 5) -
                                 Hitters_2$Salary)^2)
test_MSE_subset
```

```
## [1] 103517.5
```

Ridge regression

```
# La función glmnet() requiere pasar los predictores como matriz y la
# variable dependiente como vector.
x_Hitters_1 <- model.matrix(Salary ~ ., data = Hitters_1)[, -1]
y_Hitters_1 <- Hitters_1$Salary

x_Hitters_2 <- model.matrix(Salary ~ ., data = Hitters_2)[, -1]
y_Hitters_2 <- Hitters_2$Salary

library(glmnet)
set.seed(1)
# Se identifica mediante k-cross-validation el mejor valor de lambda para la
# ridge regresion
cv_error_ridge <- cv.glmnet(x=x_Hitters_1, y = y_Hitters_1, alpha = 0, nfolds = 10,
                            type.measure = "mse")
# Para obtener un ajuste mediante *ridge regression* se indica argumento alpha=0
modelo_ridge <- glmnet(x = x_Hitters_1, y = y_Hitters_1, alpha = 0,
                       lambda = cv_error_ridge$lambda.1se)
# Se almacenan las predicciones en una variable separada para no concatenar
# tanto código
predicciones <- predict(object = modelo_ridge, newx = x_Hitters_2,
                        s = cv_error_ridge$lambda.1se, exact = TRUE)

test_MSE_ridge <- mean((predicciones - Hitters_2$Salary)^2)
test_MSE_ridge
```

```
## [1] 124221.7
```

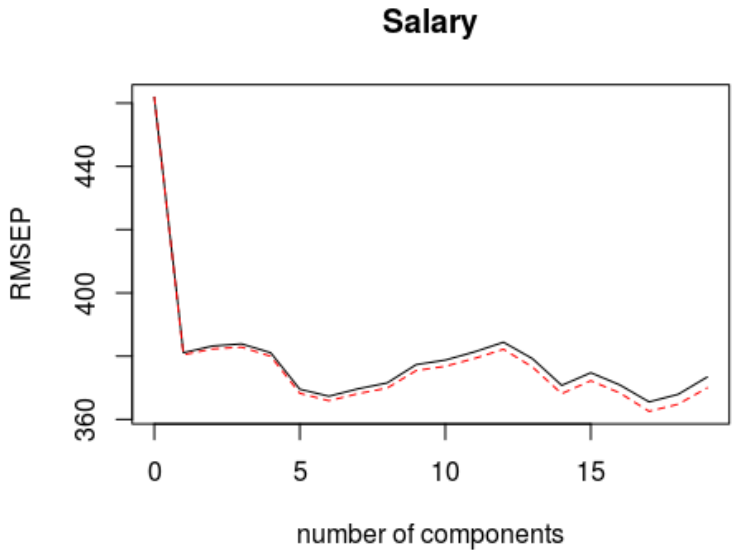
Lasso

```
library(glmnet)
set.seed(1)
# Se identifica mediante k-cross-validation el mejor valor de lambda para lasso
cv_error_ridge <- cv.glmnet(x = x_Hitters_1, y = y_Hitters_1, alpha = 1,
                            nfolds = 10, type.measure = "mse")
# Para obtener un ajuste mediante lasso se indica argumento alpha=1
modelo_lasso <- glmnet(x = x_Hitters_1, y = y_Hitters_1, alpha = 1,
                       lambda = cv_error_ridge$lambda.1se)
# Se almacenan las predicciones en una variable separada
predicciones <- predict(object = modelo_ridge, newx = x_Hitters_2,
                        s = cv_error_ridge$lambda.1se, exact = TRUE)
test_MSE_lasso <- mean((predicciones - Hitters_2$Salary)^2)
test_MSE_lasso
```

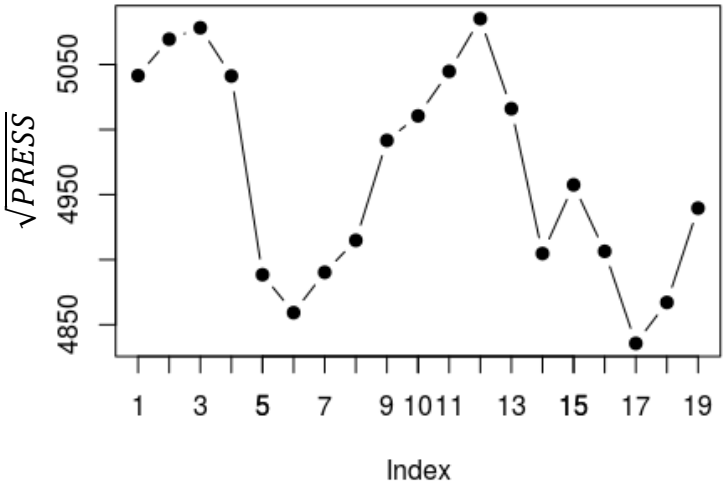
```
## [1] 117791.5
```

Principal Component Regresion PCR

```
library(pls)
set.seed(233)
# Importante estandarizar las variables indicándolo con el argumento scale=TRUE
# Indicando validation = CV, se emplea 10-fold-cross-validation para
# identificar el número óptimo de componentes.
modelo_pcr <- pcr(Salary ~ ., data = Hitters_1, scale = TRUE, validation = "CV")
validationplot(modelo_pcr, val.type = "RMSEP")
```




```
# Para ver con más detalle a partir de la componente 1
# PRESS es el Predicted Sum of Squares
plot(as.numeric(sqrt(modelo_pcr$validation$PRESS)), type = "b", pch = 19)
axis(side = 1, at = 1:19)
```



```
# Para conocer el número de componentes con el que se minimiza el error
which.min(x = modelo_pcr$validation$PRESS)
```

```
## [1] 17
```

A pesar de que el mínimo PRESS se alcanza con 17 componentes, con 6 se consigue prácticamente la misma precisión. Acorde al principio de parsimonia, el mejor modelo es el que incluye solo las 6 primeras componentes.

```
predicciones <- predict(object = modelo_pcr, newdata = Hitters_2, ncomp = 6)
test_MSE_PCR <- mean((predicciones - Hitters_2$Salary)^2)
test_MSE_PCR
```

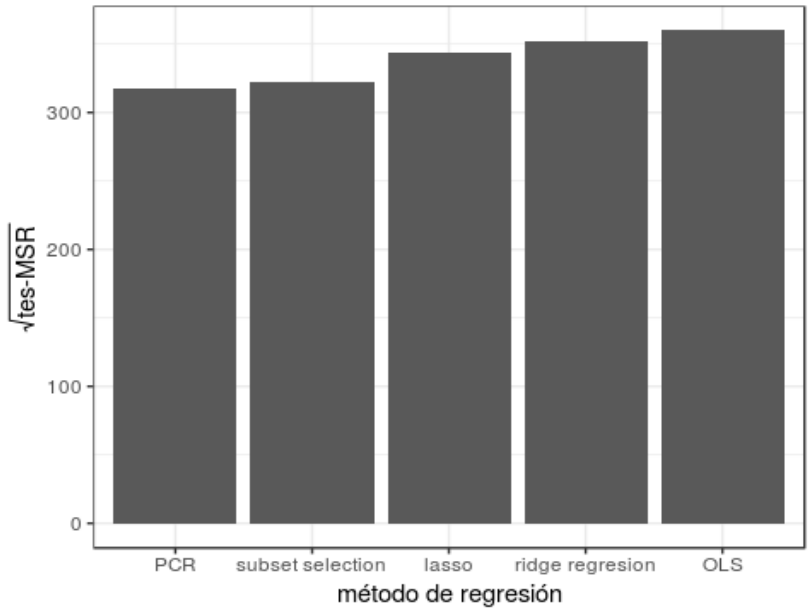
```
## [1] 100962.9
```

Conclusión

```
metodo <- c("OLS", "subset selection", "ridge regresion", "lasso", "PCR")
test_MSE <- c(test_MSE_OLS, test_MSE_subset, test_MSE_ridge, test_MSE_lasso,
              test_MSE_PCR)
resultados <- data.frame(metodo, test_MSE)
resultados
```

```
##             metodo test_MSE
## 1              OLS 129504.0
## 2 subset selection 103517.5
## 3  ridge regresion 124221.7
## 4            lasso 117791.5
## 5              PCR 100962.9
```

```
ggplot(data = resultados, aes(x = reorder(metodo, test_MSE), y = sqrt(test_MSE))) +
geom_bar(stat = "identity") +
labs(x = "método de regresión", y = expression(sqrt("tes-MSR"))) +
theme_bw()
```



Para el set de datos disponible, no existen grandes diferencias entre los resultados obtenidos por los distintos métodos. De entre todos ellos, el que consigue mayor precisión (*menor tes-MSR*) es *PCR* con las 6 primeras componentes.

Apuntes varios (miscellaneous)

En este apartado recojo comentarios, definiciones y puntualizaciones que he ido encontrando en diferentes fuentes y que, o bien no he tenido tiempo de introducir en el cuerpo principal del documento, o que he considerado que es mejor mantenerlos al margen como información complementaria.

Comparación de métodos

Linear Models with R, by Julian J. Faraway

La cuantificación del contenido en grasa de la carne pude hacerse mediante técnicas de analítica química, sin embargo, este proceso es costoso en tiempo y recursos. Una posible alternativa para reducir costes y optimizar tiempo es emplear un espectrofotómetro (instrumento capaz de detectar la absorbancia que tiene un material a diferentes tipos de luz en función de sus características). Para comprobar su efectividad se mide el espectro de absorbancia de 100 longitudes de onda en 215 muestras de carne, cuyo contenido en grasa se obtiene también por análisis químico para poder comparar los resultados. El set de datos `meatspec` del paquete `faraway` contiene toda la información.

```
library(faraway)
data(meatspec)
dim(meatspec)
```

```
## [1] 215 101
```

El set de datos contiene 101 columnas. Las 100 primeras, nombradas como $V1$, ..., $V100$ recogen el valor de absorbancia para cada una de las 100 longitudes de onda analizadas, y la columna *fat* el contenido en grasa medido por técnicas químicas.

Para poder evaluar la capacidad predictiva del modelo, se dividen las observaciones disponibles en dos grupos: uno de entrenamiento para ajustar el modelo (80% de los datos) y uno de test (20% de los datos).

```
training <- meatspec[1:172, ]
test <- meatspec[173:215, ]
```

En primer lugar se ajusta un modelo incluyendo todas las longitudes de onda como predictores.

```
modelo <- lm(fat ~ ., data = training)
summary(modelo)
```

```
## Call:
## lm(formula = fat ~ ., data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09837 -0.35779  0.04555  0.38080  2.33860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      6.324      2.012   3.143 0.002439 **
## V1           12134.077   3659.798   3.316 0.001443 **
## V2          -12585.857   5971.891  -2.108 0.038605 *
## V3           -5107.556   9390.265  -0.544 0.588200
## V4           23880.493  17143.644   1.393 0.167977
## V5          -40509.555  22129.359  -1.831 0.071360 .
## V6           28469.416  19569.400   1.455 0.150134
## V7          -20901.082  12501.639  -1.672 0.098952 .
## V8            8369.465   7515.467   1.114 0.269193
## V9           -1539.328   5397.505  -0.285 0.776327
## V10           4706.267   7406.895   0.635 0.527217
## V11           7012.943  11720.620   0.598 0.551516
## V12          14891.444  20169.170   0.738 0.462749
## V13         -30963.902  26186.839  -1.182 0.240983
## V14          34338.612  22323.830   1.538 0.128444
## V15         -22235.237  13842.268  -1.606 0.112640
## V16          -7466.797   8558.172  -0.872 0.385890
## V17           6716.653   6561.805   1.024 0.309500
## V18          -2033.071   6741.330  -0.302 0.763851
## V19           8541.212   9419.998   0.907 0.367627
## V20          -1667.207  17300.433  -0.096 0.923500
## V21         -31972.494  24622.615  -1.299 0.198317
## V22          59526.389  27730.712   2.147 0.035244 *
## V23         -49241.388  23117.226  -2.130 0.036632 *
## V24          16184.597  16679.609   0.970 0.335180
## V25          12077.951  10751.912   1.123 0.265081
## V26         -12632.330   6774.573  -1.865 0.066361 .
## V27          -6298.837   7032.334  -0.896 0.373442
## V28          29625.988   9011.227   3.288 0.001573 **
## V29         -39374.835  13561.228  -2.903 0.004914 **
## V30          31251.427  18742.000   1.667 0.099829 .
## V31         -27238.189  21335.756  -1.277 0.205887
## V32          23009.543  19776.156   1.163 0.248522
## V33          -4584.373  14572.471  -0.315 0.753995
## V34          -5437.943  10344.728  -0.526 0.600754
## V35          -6128.931   8762.663  -0.699 0.486564
```

```
## V36          5599.605   6652.640   0.842 0.402776
## V37         -5569.160   6670.198  -0.835 0.406557
## V38            97.451   9291.480   0.010 0.991661
## V39         36021.407  12574.711   2.865 0.005488 **
## V40        -54273.400  17144.384  -3.166 0.002280 **
## V41         52084.876  21758.024   2.394 0.019318 *
## V42        -48458.089  23950.549  -2.023 0.046813 *
## V43         29334.488  20232.617   1.450 0.151500
## V44        -18282.834  13508.157  -1.353 0.180200
## V45         22110.934   9725.348   2.274 0.026020 *
## V46        -11735.692   6631.245  -1.770 0.081061 .
## V47          -514.521   3800.612  -0.135 0.892696
## V48          2551.480   6131.893   0.416 0.678592
## V49          3707.639   8970.401   0.413 0.680618
## V50        -25762.703  10934.783  -2.356 0.021236 *
## V51         46844.468  15367.852   3.048 0.003233 **
## V52        -47783.626  18069.344  -2.644 0.010065 *
## V53         26233.604  18822.491   1.394 0.167744
## V54            87.825  17403.836   0.005 0.995988
## V55         -8475.119  13232.005  -0.641 0.523908
## V56          3488.507   7228.428   0.483 0.630858
## V57         -1520.733   4988.093  -0.305 0.761355
## V58          2275.175   5495.630   0.414 0.680124
## V59         -5415.427   5721.475  -0.947 0.347099
## V60          7152.015   4754.317   1.504 0.136935
## V61         -4494.234   4512.937  -0.996 0.322702
## V62          3662.045   4811.634   0.761 0.449129
## V63         13993.987   7098.106   1.972 0.052563 .
## V64        -23252.133   8973.839  -2.591 0.011604 *
## V65          4373.731  10048.591   0.435 0.664695
## V66          4580.913  10146.146   0.451 0.653011
## V67          -837.676  10747.974  -0.078 0.938097
## V68         -7074.425  10852.430  -0.652 0.516587
## V69          9506.571   9739.256   0.976 0.332325
## V70         -2765.100   9519.031  -0.290 0.772295
## V71         -1125.135   8586.061  -0.131 0.896113
## V72         -7295.096   7489.488  -0.974 0.333341
## V73         17059.811   6522.093   2.616 0.010870 *
## V74         -9889.553   6543.945  -1.511 0.135162
## V75          -325.615   6125.973  -0.053 0.957759
## V76           782.219   5421.002   0.144 0.885677
## V77          8058.935   5793.416   1.391 0.168554
## V78        -15869.978   6448.208  -2.461 0.016282 *
## V79         21768.619   6435.678   3.382 0.001172 **
## V80        -28338.145   8180.874  -3.464 0.000906 ***
## V81          8523.317  10053.153   0.848 0.399384
## V82         22319.451  12098.046   1.845 0.069226 .
## V83        -17244.722  13991.685  -1.232 0.221829
## V84        -18325.836  14959.964  -1.225 0.224627
## V85         33345.457  13868.197   2.404 0.018808 *
```

```
## V86          -7955.157  14571.278  -0.546 0.586813
## V87          -7837.966  16141.553  -0.486 0.628762
## V88          -1815.552  17261.928  -0.105 0.916532
## V89            631.595  15684.751   0.040 0.967992
## V90          -2701.955  16187.612  -0.167 0.867911
## V91           4375.678  19400.005   0.226 0.822199
## V92          12925.188  16456.244   0.785 0.434816
## V93          -7441.235  12417.883  -0.599 0.550923
## V94          -2464.532  11815.234  -0.209 0.835366
## V95          -2090.635   9666.576  -0.216 0.829394
## V96          10912.352   9950.716   1.097 0.276505
## V97         -20331.405  11022.234  -1.845 0.069270 .
## V98           3948.443   8227.133   0.480 0.632753
## V99           6358.930   8652.372   0.735 0.464800
## V100          -263.365   4104.463  -0.064 0.949019
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.074 on 71 degrees of freedom
## Multiple R-squared:  0.997,  Adjusted R-squared:  0.9928
## F-statistic: 237.5 on 100 and 71 DF,  p-value: < 2.2e-16
```

El valor $R^2_{ajustado}$ obtenido es muy alto (0.9928) lo que indica que el modelo es capaz de predecir con gran exactitud el contenido en grasa de las observaciones con las que se ha entrenado. El hecho de que el modelo en conjunto sea significativo (p-value: < 2.2e-16), pero que muy pocos de los predictores lo sean a nivel individual, es indicativo de una posible redundancia entre los predictores (colinealidad).

¿Cómo de bueno es el modelo prediciendo nuevas observaciones que no han participado en el ajuste? Al tratarse de un modelo de regresión, la estimación del error de predicción se obtiene mediante el *Mean Square Error (MSE)*.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

```
# MSE empleando las observaciones de entrenamiento
training_mse <- mean((modelo$fitted.values - training$fat)^2)
training_mse
```

```
## [1] 0.4765372
```

```
# MSE empleando nuevas observaciones
predicciones <- predict(modelo, newdata = test)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 14.54659
```

Se observa que el modelo tiene un MSE muy bajo (0.48) cuando predice las mismas observaciones con las que se ha entrenado, pero 30 veces más alto (14.54) al predecir nuevas observaciones. Esto significa que el modelo no es útil, ya que el objetivo es aplicarlo para predecir el contenido en grasa de futuras muestras de carne. A este problema se le conoce como *overfitting*. Una de las causas por las que un modelo puede sufrir *overfitting* es la incorporación de predictores innecesarios, que no aportan información o que la información que aportan es redundante.

Stepwise Selection

Se recurre en primer lugar a la selección de predictores mediante *stepwise selection* empleando el *AIC* como criterio de evaluación:

```
modelo_step_selection <- step(object = modelo, trace = FALSE)

# Número de predictores del modelo resultante
length(modelo_step_selection$coefficients)
```

```
## [1] 73
```

```
# Training-MSE
training_mse <- mean((modelo_step_selection$fitted.values - training$fat)^2)
training_mse
```

```
## [1] 0.5034001
```

```
# Test-MSE
predicciones <- predict(modelo_step_selection, newdata = test)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 12.88986
```

El proceso de *stepwise selection* devuelve como mejor modelo el formado por 73 de los 100 predictores disponibles. Al haber eliminado predictores del modelo, el *training MSE* siempre aumenta, en este caso de 0.48 a 0.5, pero el *test-MSE* se ha reducido a 12.88986.

PCR

Véase ahora el resultado si se ajusta el modelo empleando las componentes principales por *PCR*:

```
# Cálculo de componentes principales. Se excluye la columna con la variable
# respuesta *fat*
pca <- prcomp(training[, -101], scale. = TRUE)

# Se muestra la proporción de varianza explicada y acumulada de las 9
# primeras componentes
summary(pca)$importance[, 1:9]
```

```
##                             PC1      PC2       PC3       PC4        PC5
## Standard deviation     9.92492 1.043606 0.5357885 0.3312792 0.07898436
## Proportion of Variance 0.98504 0.010890 0.0028700 0.0011000 0.00006000
## Cumulative Proportion  0.98504 0.995930 0.9988000 0.9999000 0.99996000
##                               PC6        PC7        PC8         PC9
## Standard deviation     0.04974461 0.02700194 0.02059129 0.008603878
## Proportion of Variance 0.00002000 0.00001000 0.00000000 0.000000000
## Cumulative Proportion  0.99999000 0.99999000 1.00000000 1.000000000
```

El estudio de la proporción de varianza explicada muestra que la primera componente recoge la mayor parte de la información (98%), decayendo drásticamente la varianza en las sucesivas componentes.

Una vez obtenido el valor de las componentes para cada observación (*principal component scores*), puede ajustarse el modelo lineal empleando dichos valores junto con la variable respuesta que le corresponde a cada observación. Con la función `pcr()` del paquete `pls` se evita tener que codificar cada uno de los pasos intermedios. Acorde a la proporción de varianza acumulada, emplear las 4 primeras componentes podría ser una buena elección, ya que en conjunto explican el 99.99 % de varianza.

```
library(pls)
modelo_pcr <- pcr(formula = fat ~ ., data = training, scale. = TRUE, ncomp = 4)
# Test-MSE
predicciones <- predict(modelo_pcr, newdata = test, ncomp = 4)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 20.55699
```

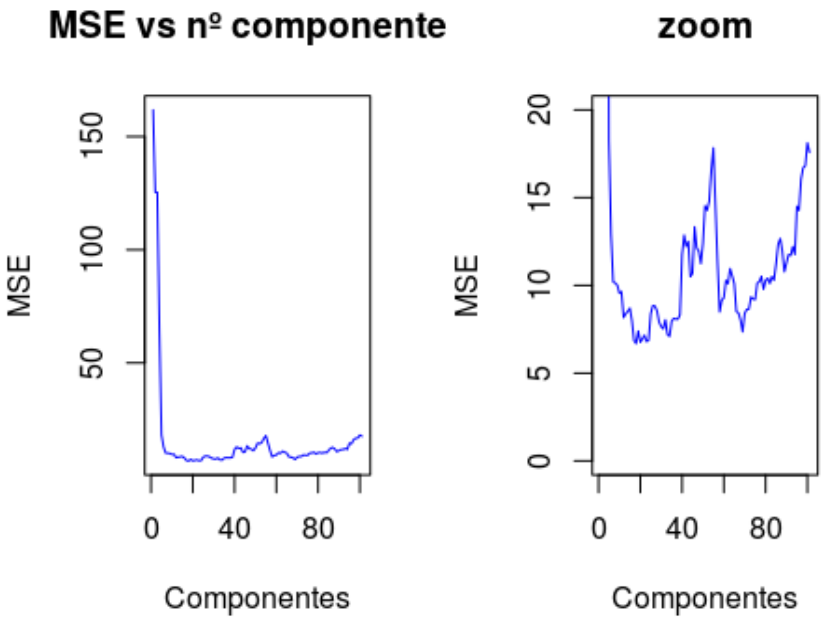
El *test-MSE* obtenido (20.56) para el modelo que emplea como predictores las 4 primeras componentes es mucho mayor que el obtenido con el modelo generado por *stepwise selection* (12.89) e incluso que el obtenido incluyendo todos los predictores (14.54659). Esto significa que, o bien el hecho de emplear componentes principales como predictores no es útil para este caso, o que el número de componentes incluido no es el adecuado.

La función `pcr()` incluye la posibilidad de recurrir a *cross validation* para identificar el número óptimo de componentes con el que se minimiza el *MSE*.

```
set.seed(123)
modelo_pcr <- pcr(formula = fat ~ ., data = training, scale. = TRUE,
                  validation = "CV")
modelo_pcr_CV <- MSEP(modelo_pcr, estimate = "CV")
which.min(modelo_pcr_CV$val)
```

```
## [1] 18
```

```
par(mfrow = c(1, 2))
plot(modelo_pcr_CV$val, main = "MSE vs nº componentes", type = "l", ylab = "MSE",
     col = "blue", xlab = "Componentes")
plot(modelo_pcr_CV$val, main = "zoom", type = "l", ylab = "MSE",
     xlab = "Componentes", col = "blue", ylim = c(0, 20))
```




```
# Test-MSE
predicciones <- predict(modelo_pcr, newdata = test, ncomp = 18)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 4.524698
```

El número óptimo de componentes principales identificado por *cross validation* es de 18. Empleando este número en la *PCR* se consigue reducir el *test-MSE* a 4.52, un valor muy por debajo del conseguido con los otros modelos.

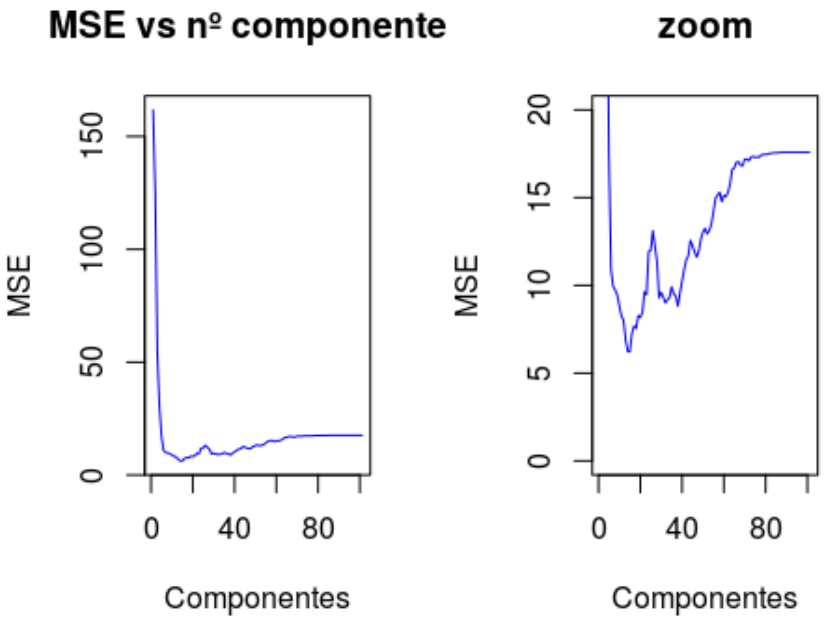
PLS

Véase ahora el resultado si se ajusta el modelo empleando las componentes principales por *PLS*:

```
set.seed(123)
modelo_pls <- plsr(formula = fat ~ ., data = training, scale. = TRUE,
                   validation = "CV")
modelo_pls_CV <- MSEP(modelo_pls, estimate = "CV")
which.min(modelo_pls_CV$val)
```

```
## [1] 15
```

```
par(mfrow = c(1, 2))
plot(modelo_pls_CV$val, main = "MSE vs nº componentes", type = "l", ylab = "MSE",
     col = "blue", xlab = "Componentes")
plot(modelo_pls_CV$val, main = "zoom", type = "l", ylab = "MSE",
     xlab = "Componentes", col = "blue", ylim = c(0, 20))
```




```
# Test-MSE
predicciones <- predict(modelo_pls, newdata = test, ncomp = 15)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 3.888104
```

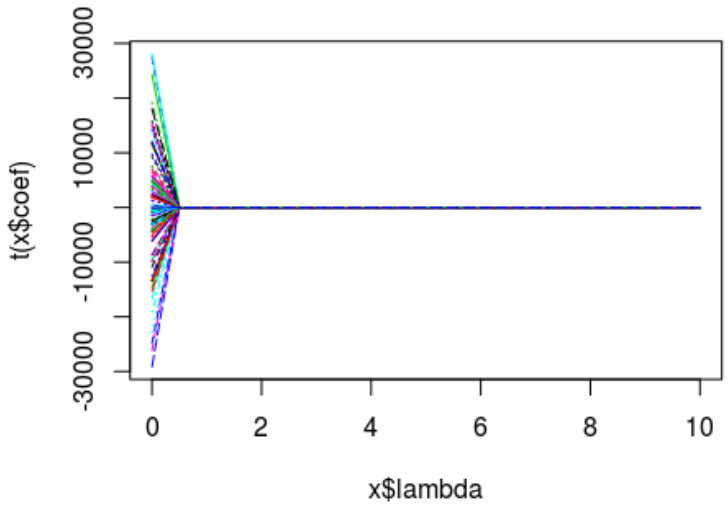
Si se comparan los resultados obtenidos por *PCR* y *PLS* se observa que el número de componentes óptimo es inferior en *PLS*. Esto suele ser así ya que en el proceso de *PLS* se está incluyendo información adicional a través de la variable respuesta. Para este ejemplo, el método *PLS* consigue un *test-MSE* ligeramente inferior al obtenido por *PCR*. En el caso de querer utilizar cualquiera de los modelos anteriores con fines predictivos, es necesario verificar que se cumplen las condiciones necesarias para regresión por mínimos cuadrados.

Ridge Regression

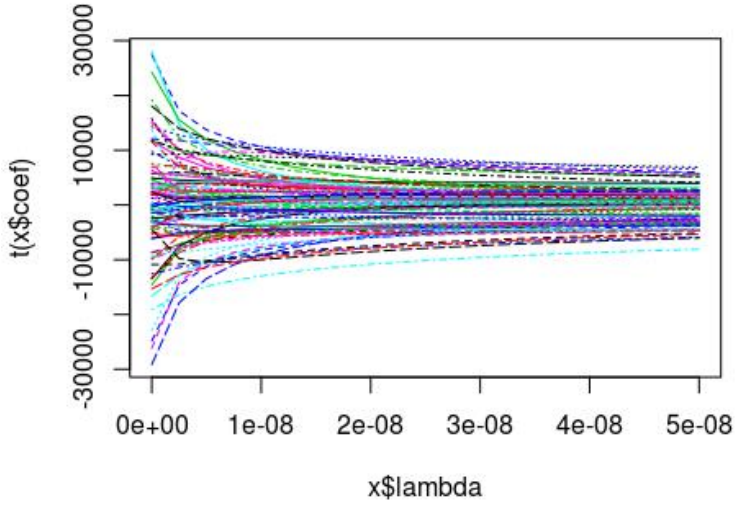
Nota: Dos de las funciones de `R` *que permiten realizar ajustes por ridge regression son* `lm.ridge()` *y* `glmnet()`*. Los resultados obtenidos por cada una de las funciones no son iguales. La razón de ello es que una utiliza "penalized least squares" y la otra "penalized mean squared error". Esto significa que una minimiza la suma de los residuos al cuadrado y la otra la media de los residuos al cuadrado, ambas con la penalización de ridge. Desconozco las ventajas y desventajas de cada método, pero es importante tener en cuenta que existe esta diferencia.*

En esta ocasión se recurre a la función `lm.ridge()` del paquete `MASS` para ajustar el modelo lineal por *ridge regression*.

```
library(MASS)
# La función lm.ridge() estandariza las variables previo ajuste (sustrae la
# media y divide por la desviación estándar)
modelo_ridge <- lm.ridge(formula = fat ~ ., data = training,
                         lambda = seq(0, 10, len = 21))
plot(modelo_ridge)
```



```
# La representación de los coeficientes en función de lambda muestra que se
# puede ajustar el rango de valores estudiado
modelo_ridge <- lm.ridge(formula = fat ~ ., data = training,
                         lambda = seq(0, 5e-08, len = 21))
plot(modelo_ridge)
```



Empleando *cross-validation* se puede identificar el valor óptimo de *lambda* λ. La función `lm.ridge()` tiene implementado el método *generalized cross-validation (GCV)*, que es similar a *cross-validation* pero computacionalmente más sencillo. La estimación del error para cada λ se almacena en el elemento GCV.

```
which.min(modelo_ridge$GCV)
```

```
## 1.75e-08
##        8
```

Acorde a *generalized cross-validation (GCV)*, el valor de λ con el que se minimiza el *MSE* es 1.75e-08. Este es el octavo valor de la secuencia de lambdas testadas. Los coeficientes de regresión del modelo cuando se emplea este λ se encuentran en la octava fila de la matriz coef(modelo_ridge).

Una vez identificado el valor óptimo de λ, se calcula el *test-MSE*.

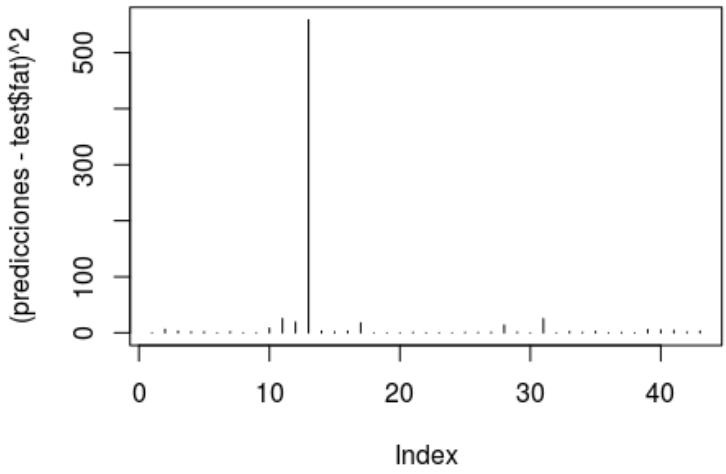
```
# No existe método predict() aplicable a modelos lm.ridge, por lo que las
# predicciones se obtienen por producto matricial del valor de los
# predictores y los coeficientes del modelo

predicciones <- cbind(1, as.matrix(test[, -101])) %*% coef(modelo_ridge)[8, ]
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 16.81874
```

El valor de *test-MSE* obtenido es sorprendentemente alto, por encima del obtenido empleando regresión lineal por mínimos cuadrados incluyendo todos los predictores.

Una evaluación detallada de los residuos muestra que hay una observación muy atípica, la 13.

```
plot((predicciones - test$fat)^2, type = "h")
```




```
which.max((predicciones - test$fat)^2)
```

```
## [1] 13
```

Si se excluye esta observación de la estimación del *test-MSE* se reduce en gran medida.

```
mean((predicciones[-13] - test$fat[-13])^2)
```

```
## [1] 3.918271
```

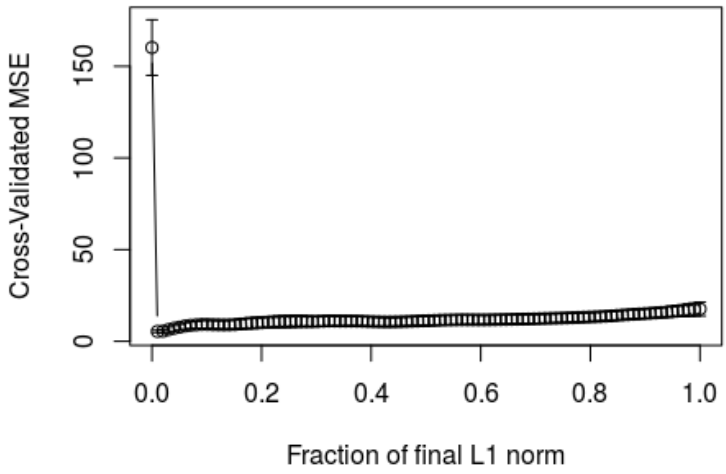
Lasso

La función `lars()` del paquete `lars` permite ajustar modelos mediante Lasso. Esta función no acepta fórmula, por lo que se tienen que pasar como argumentos una matriz con el valor de los predictores y un vector con la variable respuesta.

```
library(lars)
modelo_lasso <- lars(x = as.matrix(training[, -101]), y = training$fat,
                     type = "lasso", normalize = TRUE)
```

Al igual que en *Ridge Regression*, el valor óptimo de penalización λ se puede identificar mediante *cross-validations*.

```
set.seed(123)
cv_modelo_lasso <- cv.lars(x = as.matrix(training[, -101]), y = training$fat,
                           K = 10, plot.it = TRUE)
```

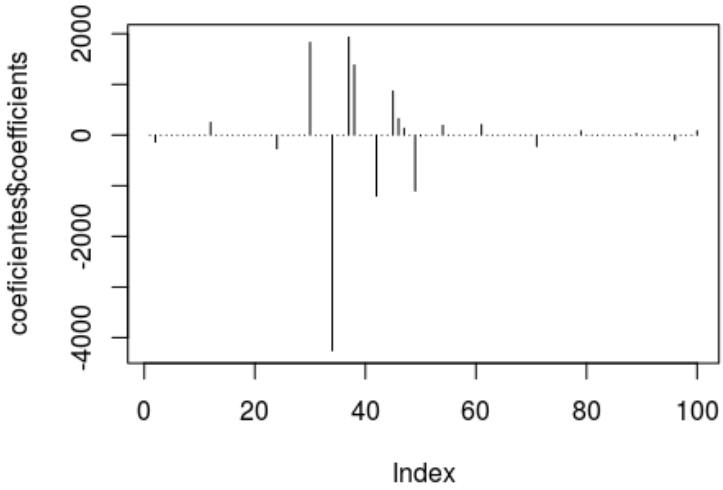



```
cv_modelo_lasso$index[which.min(cv_modelo_lasso$cv)]
```

```
## [1] 0.01010101
```

En este caso, el λ óptimo es 0.01010101. Para conocer el valor de los coeficientes de regresión del modelo empleando esta lambda

```
coeficientes <- predict(modelo_lasso, type = "coefficients", s = 0.0101,
                        mode = "fraction")
plot(coeficientes$coefficients, type = "h")
```




```
# Los predictores seleccionados por Lasso son aquellos cuyo valor de
# coeficiente ha resultado diferente de cero
coeficientes$coefficients[coeficientes$coefficients != 0]
```

```
##          V2         V12         V24         V30         V34         V37
##  -137.11044   249.46016  -266.11921  1827.73322 -4255.89431  1931.27628
##         V38         V42         V45         V46         V47         V49
##  1383.86494 -1202.58184   867.17648   324.93092   131.61133 -1102.57134
##         V50         V54         V61         V71         V79         V89
##   -15.74004   189.47166   205.20030  -223.67400    80.76254    27.25873
##         V96        V100
##   -96.86846    81.65118
```

```
length(coeficientes$coefficients[coeficientes$coefficients != 0])
```

```
## [1] 20
```

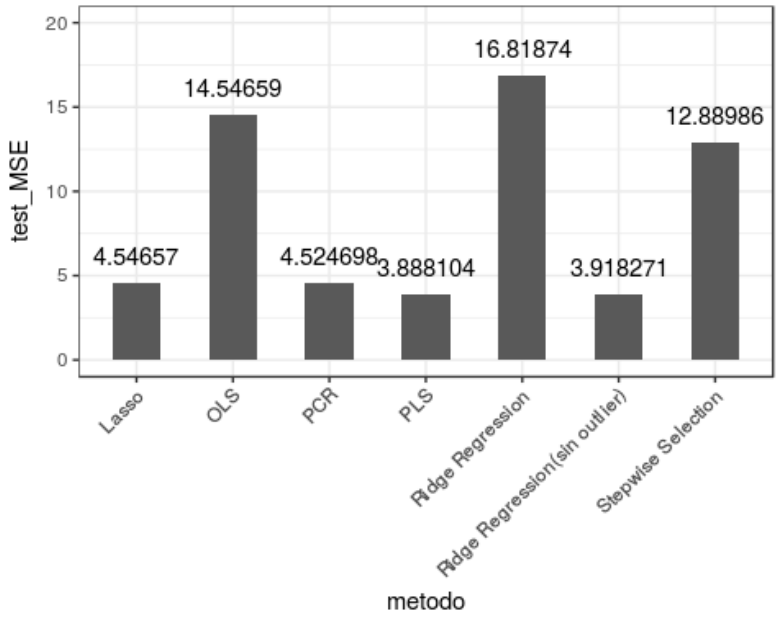
La estimación del *test-MSE* del modelo ajustado por Lasso empleando este valor de $\lambda = 0.0101$ es:

```
predicciones <- predict(modelo_lasso, newx = as.matrix(test[, -101]), s = 0.0101,
                        mode = "fraction")
test_mse <- mean((predicciones$fit - test$fat)^2)
test_mse
```

```
## [1] 4.54657
```

Conclusión

```
library(ggplot2)
valores_testMSE <- data.frame(metodo = c("OLS", "Stepwise Selection", "PCR",
                                         "PLS", "Ridge Regression",
                                         "Ridge Regression(sin outlier)",
                                         "Lasso"),
                              test_MSE = c(14.54659, 12.88986, 4.524698, 3.888104,
                                           16.81874, 3.918271, 4.54657))
ggplot(data = valores_testMSE, aes(x = metodo, y = test_MSE)) +
geom_col(width = 0.5) +
lims(y = c(0, 20)) +
geom_text(aes(label = test_MSE), vjust = -1) +
theme_bw() + theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Los modelos obtenidos por *PLS*, *Ridge Regression*, *PCR* y *Lasso* son aproximadamente igual de buenos y superiores a los obtenidos por *OLS* y *Stepwise Selection*. Cabe mencionar que si bien el método *Ridge Regression* consigue un *test-RME* bajo, ha sido necesario excluir una observación. Sin su eliminación, es el peor modelo. Esto pone de manifiesto la sensibilidad de *Ridge Regression* a valores atípicos.

Para este escenario, en el que obtener los valores de las 100 longitudes de onda es aproximadamente igual de costoso que el obtener solo unas pocas, cualquiera de los 4 primeros métodos es aproximadamente igual de bueno. Sin embargo, si leer solo unas pocas longitudes de onda supone un ahorro en costes, el modelo *Lasso* presenta la ventaja frente a los otros de que solo necesita 20 de ellas.

Bibliografía

Introduction to Statistical Learning, Gareth James, Daniela Witten, Trevor Hastie and Robert

Tibshirani *Linear Models with R, by Julian J. Faraway*

Regularization and variable selection via the elastic net, Hui Zou and Trevor Hastie, J. R. Statist. Soc.B (2005)