Ridge - Lasso

Regresión lineal con regularización

Para el caso de la regresión Ridge podremos usar los paquetes MASS y glmnet. Seguiremos la práctica 6.5.2. del capítulo 6 del libro.

Observamos los datos

Veamos primero hasta que punto está justificado Ridge o/y Lasso.

```
library(ISLR2)
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
#Hitters <- read.csv("~/matematicas/aprendizaje estadistico/LIBRO/ALL CSV FILES - 2nd Edition/Hitters.c
names(Hitters)
```

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

```
dim(Hitters)
```

```
## [1] 322  20
```

```
?Hitters
```

Hay que eliminar las muestras con NA en Salary

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

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

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

```
attach(Hitters)
```

Observamos que hay variables no numéricas,

```r
str(Hitters)
```

```
## 'data.frame':    263 obs. of  20 variables:
##  $ AtBat    : int  315 479 496 321 594 185 298 323 401 574 ...
##  $ Hits     : int  81 130 141 87 169 37 73 81 92 159 ...
##  $ HmRun    : int  7 18 20 10 4 1 0 6 17 21 ...
##  $ Runs     : int  24 66 65 39 74 23 24 26 49 107 ...
##  $ RBI      : int  38 72 78 42 51 8 24 32 66 75 ...
##  $ Walks    : int  39 76 37 30 35 21 7 8 65 59 ...
##  $ Years    : int  14 3 11 2 11 2 3 2 13 10 ...
##  $ CAtBat   : int  3449 1624 5628 396 4408 214 509 341 5206 4631 ...
##  $ CHits    : int  835 457 1575 101 1133 42 108 86 1332 1300 ...
##  $ CHmRun   : int  69 63 225 12 19 1 0 6 253 90 ...
##  $ CRuns    : int  321 224 828 48 501 30 41 32 784 702 ...
##  $ CRBI     : int  414 266 838 46 336 9 37 34 890 504 ...
##  $ CWalks   : int  375 263 354 33 194 24 12 8 866 488 ...
##  $ League   : Factor w/ 2 levels "A","N": 2 1 2 2 1 2 1 2 1 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 2 2 1 1 2 1 2 2 1 1 ...
##  $ PutOuts  : int  632 880 200 805 282 76 121 143 0 238 ...
##  $ Assists  : int  43 82 11 40 421 127 283 290 0 445 ...
##  $ Errors   : int  10 14 3 4 25 7 9 19 0 22 ...
##  $ Salary   : num  475 480 500 91.5 750 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 2 1 2 2 1 1 1 2 1 1 ...
##  - attr(*, "na.action")= 'omit' Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
##   ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Boone" ...
```

No hace falta redefinir las variables de "texto" para hacer la regresión lineal.

```r
lineal=lm(Salary~.,Hitters)
coef(lineal)
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  163.1035878   -1.9798729    7.5007675    4.3308829   -2.3762100   -1.0449620
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    6.2312863   -3.4890543   -0.1713405    0.1339910   -0.1728611    1.4543049
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##    0.8077088   -0.8115709   62.5994230 -116.8492456    0.2818925    0.3710692
##       Errors   NewLeagueN
##   -3.3607605  -24.7623251
```

No es un modelo que vaya a tener un buen ajuste.

Veámos primero estadísticos a partir del modelo:

```r
summary(lineal)
```

```
##
## Call:
## lm(formula = Salary ~ ., data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -907.62 -178.35  -31.11  139.09 1877.04
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  163.10359   90.77854   1.797 0.073622 .
```

```
## AtBat         -1.97987    0.63398  -3.123 0.002008 **
## Hits           7.50077    2.37753   3.155 0.001808 **
## HmRun          4.33088    6.20145   0.698 0.485616
## Runs          -2.37621    2.98076  -0.797 0.426122
## RBI           -1.04496    2.60088  -0.402 0.688204
## Walks          6.23129    1.82850   3.408 0.000766 ***
## Years         -3.48905   12.41219  -0.281 0.778874
## CAtBat        -0.17134    0.13524  -1.267 0.206380
## CHits          0.13399    0.67455   0.199 0.842713
## CHmRun        -0.17286    1.61724  -0.107 0.914967
## CRuns          1.45430    0.75046   1.938 0.053795 .
## CRBI           0.80771    0.69262   1.166 0.244691
## CWalks        -0.81157    0.32808  -2.474 0.014057 *
## LeagueN       62.59942   79.26140   0.790 0.430424
## DivisionW   -116.84925   40.36695  -2.895 0.004141 **
## PutOuts        0.28189    0.07744   3.640 0.000333 ***
## Assists        0.37107    0.22120   1.678 0.094723 .
## Errors        -3.36076    4.39163  -0.765 0.444857
## NewLeagueN   -24.76233   79.00263  -0.313 0.754218
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 315.6 on 243 degrees of freedom
## Multiple R-squared:  0.5461, Adjusted R-squared:  0.5106
## F-statistic: 15.39 on 19 and 243 DF,  p-value: < 2.2e-16
```

Observamos que R^2= 0.54: 54% de la variabilidad de la variable Salary se explica con el resto de las variables. Además los t values indican que ciertos parámetros del modelo podrían ser nulos.

Estudiemos/Estimemos el MSE:

```
MSE_all=mean(resid(lineal)^2)#cogiendo todas las observaciones
MSE_all
```

```
## [1] 92017.87
```

```
mean(Hitters$Salary)
```

```
## [1] 535.9259
```

Veamos con CV con 10 iteraciones una mejor estimación del MSE, y con bootstrapping, una mejor estimación de los SE.

```
library(boot)
glm.fit <- glm( Salary~., data = Hitters)
cv.glm(Hitters, glm.fit, K = 10)$delta[1] #mucho mayor que el MSE con todo el conjunto de entrenamiento
```

```
## [1] 123025.7
```

```
bootfuncion <- function(data, index)
coef(lm(formula = Salary~., data = Hitters, subset = index))
boot(Hitters, bootfuncion, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
```

```
## boot(data = Hitters, statistic = bootfuncion, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1*   163.1035878 -6.5170414831 134.12775432
## t2*    -1.9798729  0.1667942494   0.75426078
## t3*     7.5007675 -0.6618204902   2.97749795
## t4*     4.3308829 -0.4017050263   7.06274866
## t5*    -2.3762100  0.0850780690   3.21334775
## t6*    -1.0449620  0.3049558478   3.06748183
## t7*     6.2312863 -0.3875335993   1.91758278
## t8*    -3.4890543 -0.5467258412  12.60499946
## t9*    -0.1713405 -0.0301245329   0.19139116
## t10*    0.1339910  0.2058663586   0.89351541
## t11*   -0.1728611  0.4103293009   2.15412424
## t12*    1.4543049 -0.0693626424   0.80878558
## t13*    0.8077088 -0.2466794654   0.86884426
## t14*   -0.8115709  0.0933642516   0.40207019
## t15*   62.5994230  4.6267270961  67.68374358
## t16* -116.8492456  8.0574239138  38.63263855
## t17*    0.2818925 -0.0005533699   0.09949918
## t18*    0.3710692 -0.0078331749   0.27575167
## t19*   -3.3607605 -0.1653279615   4.46142238
## t20*  -24.7623251 -4.3213915328  68.79375356
```

Estudiemos ahora la colinealidad: Veamos el factor de inflación de la varianza (VIF: Variance Inflation Factor), $\text{VIF}(\beta_j) = \frac{1}{1-R^2_{X_j|X_{-j}}}$.

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:boot':
##
##     logit
```

```
vif(lineal)
```

```
##      AtBat       Hits      HmRun       Runs        RBI      Walks      Years
##  22.944366  30.281255   7.758668  15.246418  11.921715   4.148712   9.313280
##     CAtBat      CHits     CHmRun      CRuns       CRBI     CWalks     League
## 251.561160 502.954289  46.488462 162.520810 131.965858  19.744105   4.134115
##   Division    PutOuts    Assists     Errors  NewLeague
##   1.075398   1.236317   2.709341   2.214543   4.099063
```

Más indicadores:

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

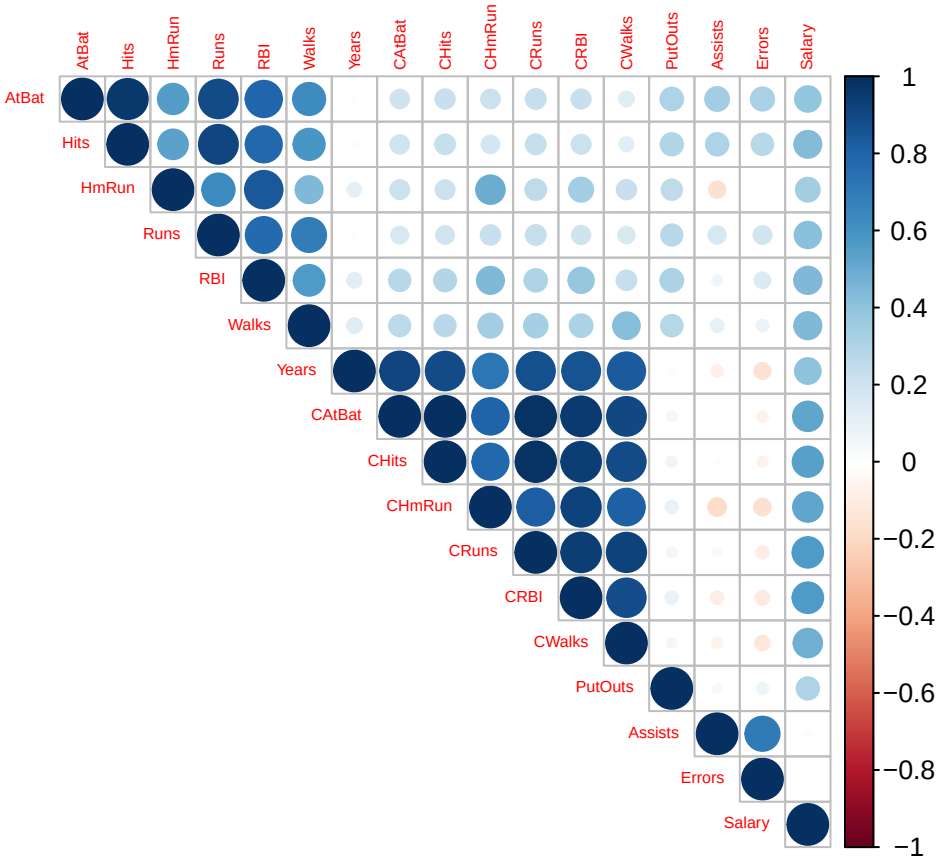
```
cor(Hitters[,c(-14,-15,-20)])
```

```
##             AtBat       Hits        HmRun        Runs        RBI      Walks
```

```
## AtBat   1.0000000 0.96396913  0.555102154  0.89982910 0.79601539 0.6244481
## Hits    0.9639691 1.00000000  0.530627358  0.91063014 0.78847819 0.5873105
## HmRun   0.5551022 0.53062736  1.000000000  0.63107588 0.84910743 0.4404537
## Runs    0.8998291 0.91063014  0.631075883  1.00000000 0.77869235 0.6970151
## RBI     0.7960154 0.78847819  0.849107434  0.77869235 1.00000000 0.5695048
## Walks   0.6244481 0.58731051  0.440453717  0.69701510 0.56950476 1.0000000
## Years   0.0127255 0.01859809  0.113488420 -0.01197495 0.12966795 0.1347927
## CAtBat  0.2071663 0.20667761  0.217463613  0.17181080 0.27812591 0.2694500
## CHits   0.2253415 0.23560577  0.217495691  0.19132697 0.29213714 0.2707951
## CHmRun  0.2124215 0.18936425  0.492525845  0.22970104 0.44218969 0.3495822
## CRuns   0.2372778 0.23889610  0.258346846  0.23783121 0.30722616 0.3329766
## CRBI    0.2213932 0.21938423  0.349858379  0.20233548 0.38777657 0.3126968
## CWalks  0.1329257 0.12297073  0.227183183  0.16370021 0.23361884 0.4291399
## PutOuts 0.3096075 0.29968754  0.250931497  0.27115986 0.31206456 0.2808555
## Assists 0.3421174 0.30397495 -0.161601753  0.17925786 0.06290174 0.1025226
## Errors  0.3255770 0.27987618 -0.009743082  0.19260879 0.15015469 0.0819372
## Salary  0.3947709 0.43867474  0.343028078  0.41985856 0.44945709 0.4438673
##               Years       CAtBat       CHits      CHmRun       CRuns
## AtBat    0.01272550  0.207166254  0.22534146  0.21242155  0.23727777
## Hits     0.01859809  0.206677608  0.23560577  0.18936425  0.23889610
## HmRun    0.11348842  0.217463613  0.21749569  0.49252584  0.25834685
## Runs    -0.01197495  0.171810798  0.19132697  0.22970104  0.23783121
## RBI      0.12966795  0.278125914  0.29213714  0.44218969  0.30722616
## Walks    0.13479270  0.269449974  0.27079505  0.34958216  0.33297657
## Years    1.00000000  0.915680692  0.89784449  0.72237071  0.87664855
## CAtBat   0.91568069  1.000000000  0.99505681  0.80167609  0.98274694
## CHits    0.89784449  0.995056810  1.00000000  0.78665204  0.98454184
## CHmRun   0.72237071  0.801676089  0.78665204  1.00000000  0.82562483
## CRuns    0.87664855  0.982746941  0.98454184  0.82562483  1.00000000
## CRBI     0.86380936  0.950730141  0.94679739  0.92790264  0.94567701
## CWalks   0.83752373  0.906711655  0.89071842  0.81087827  0.92776846
## PutOuts -0.02001921  0.053392514  0.06734799  0.09382223  0.05908718
## Assists -0.08511772 -0.007897271 -0.01314420 -0.18888646 -0.03889509
## Errors  -0.15651196 -0.070477521 -0.06803583 -0.16536941 -0.09408054
## Salary   0.40065699  0.526135310  0.54890956  0.52493056  0.56267771
##                CRBI      CWalks     PutOuts      Assists       Errors
## AtBat    0.22139318  0.13292568  0.30960746  0.342117377  0.325576978
## Hits     0.21938423  0.12297073  0.29968754  0.303974950  0.279876183
## HmRun    0.34985838  0.22718318  0.25093150 -0.161601753 -0.009743082
## Runs     0.20233548  0.16370021  0.27115986  0.179257859  0.192608787
## RBI      0.38777657  0.23361884  0.31206456  0.062901737  0.150154692
## Walks    0.31269680  0.42913990  0.28085548  0.102522559  0.081937197
## Years    0.86380936  0.83752373 -0.02001921 -0.085117725 -0.156511957
## CAtBat   0.95073014  0.90671165  0.05339251 -0.007897271 -0.070477521
## CHits    0.94679739  0.89071842  0.06734799 -0.013144204 -0.068035829
## CHmRun   0.92790264  0.81087827  0.09382223 -0.188886464 -0.165369407
## CRuns    0.94567701  0.92776846  0.05908718 -0.038895093 -0.094080542
## CRBI     1.00000000  0.88913701  0.09537515 -0.096558877 -0.115316131
## CWalks   0.88913701  1.00000000  0.05816016 -0.066243445 -0.129935875
## PutOuts  0.09537515  0.05816016  1.00000000 -0.043390143  0.075305857
## Assists -0.09655888 -0.06624345 -0.04339014  1.000000000  0.703504693
## Errors  -0.11531613 -0.12993587  0.07530586  0.703504693  1.000000000
## Salary   0.56696569  0.48982204  0.30048036  0.025436136 -0.005400702
##               Salary
```

```
## AtBat     0.394770945
## Hits      0.438674738
## HmRun     0.343028078
## Runs      0.419858559
## RBI       0.449457088
## Walks     0.443867260
## Years     0.400656994
## CAtBat    0.526135310
## CHits     0.548909559
## CHmRun    0.524930560
## CRuns     0.562677711
## CRBI      0.566965686
## CWalks    0.489822036
## PutOuts   0.300480356
## Assists   0.025436136
## Errors   -0.005400702
## Salary    1.000000000
```

```
corrplot(cor(Hitters[,c(-14,-15,-20)]),type="upper",tl.cex=0.5)
```



Todas estas observaciones llevan a pensar que el modelo lineal así considerado no es el idóneo. La colinialidad que existe entre algunas variables implica la mala estabilidad del modelo de regresión lineal.

Ridge y Lasso

Todo indica que si hacemos ridge o lasso, facilmente encontraremos un modelo mejor.

Para ello, utilizaremos el paquete glmnet. Esta función tiene una sintaxis ligeramente diferente a la de otras funciones de ajuste del modelo que hemos encontrado hasta ahora en este libro. En particular, como los predictores como la variable que se quiere predecir se introducen en forma vectorial/matricial.

La función model.matrix() no solo produce la matriz correspondiente correspondiente a los 19 predictores, sino que también transforma automáticamente cualquier variable cualitativa en variables dummy. Esta última propiedad es importante porque glmnet() solo puede tomar entradas numéricas.

```
x<- model.matrix(Salary ~ ., Hitters)[,-1]##¿Por qué -1?
y <- Hitters$Salary
```

Nota: el condition number=σ_1/σ_{p+1} de la matriz $A^t \cdot A$ es 4.242998854.10^8, lo que indica que está mal condicionada, es decir, la solución por mínimo cuadrados no es nada estable.

Realizamos el ajuste de Ridge tomando valores diferentes del parámetro λ (tuning parameter). Recuerda que a mayor valor de lambda, menos flexibilidad, menor valor numérico en valor absoluto de los parámetros del modelo porque su longitud es menor.

Si seguimos la práctica del libro, elegimos los lambda's nosotros. Dicho esto, por defecto R elige 100 valores.

```
#glmnet(x,y,alpha=0)
#seq(10, -2, length = 100)
malla <- 10^seq(10, -2, length = 100)
```

Aunque utilicemos la herramienta de glmnet, también se puede realizar el método de Ridge con "lm.ridge".

```
malla_ridge<-glmnet(x,y,alpha=0,lambda=malla)
```

Obtenemos así tantas columnas como valores de lambda; cada una nos da un modelo.

```
dim(coef(malla_ridge))
```

```
## [1]  20 100
```

```
malla_ridge$lambda[50]
```

```
## [1] 11497.57
```

```
coef(malla_ridge)[,50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

La norma de los coeficientes estimados:

```
sqrt(sum(coef(malla_ridge)[-1,50]^2))
```

```
## [1] 6.360612
```

Si comparamos con otro valor de λ:

```
malla_ridge$lambda[75]
```

```
## [1] 10.72267
```

```
coef(malla_ridge)[,75]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  121.03941275   -1.17121685    4.16555335   -0.67027625    0.50301494
##           RBI         Walks         Years        CAtBat         CHits
##    0.45074152    4.34665150  -10.91561927   -0.02154818    0.17263636
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.74497092    0.46162220    0.30888439   -0.45438108   58.89440480
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -124.17545130    0.27401855    0.23373456   -3.85516497  -25.24429108
```

La norma tiene que se mayor porque λ es menor:

```
sqrt(sum(coef(malla_ridge)[-1,75]^2))
```

```
## [1] 140.3536
```

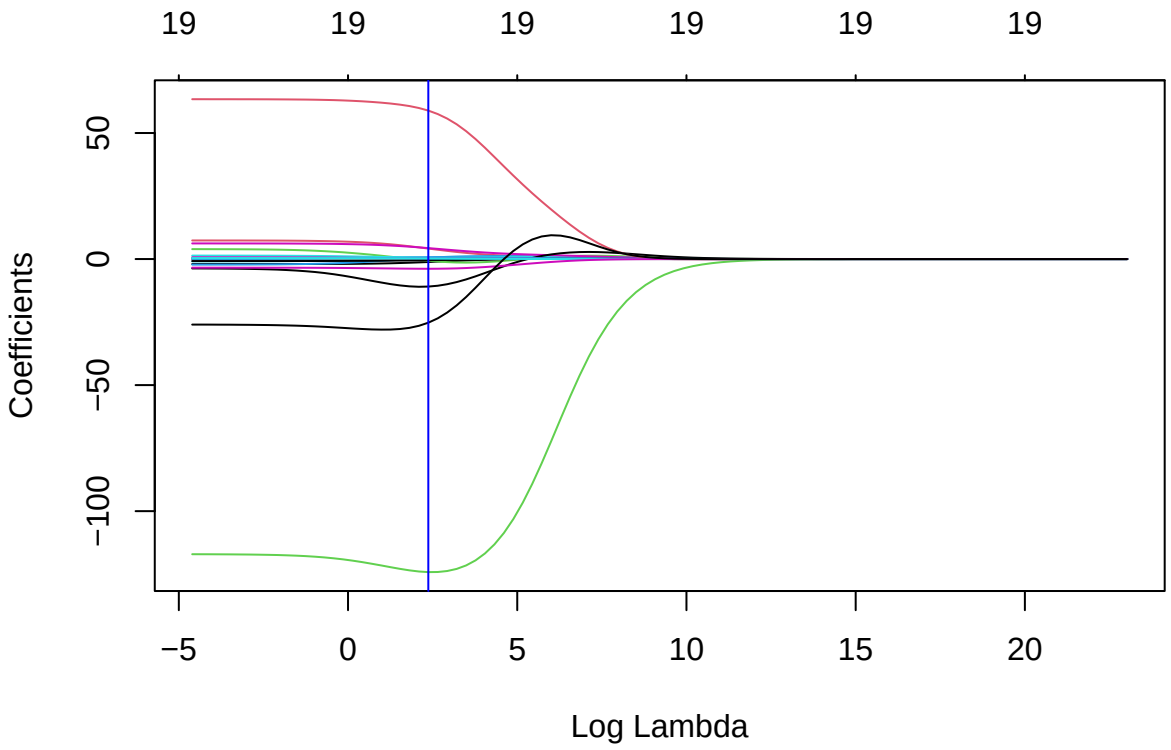
La norma de los coeficientes del modelo lineal es mucho mayor:

```
sqrt(sum(coef(lineal)^2))
```

```
## [1] 211.9908
```

Con un plot visualizamos los diferentes valores de los parámetros para los diferentes valores de λ

```
plot(malla_ridge,xvar="lambda" )
abline(v=log(malla_ridge$lambda[75] ),col='blue')
```



Podemos usar la función predict() para varios propósitos. Por ejemplo, podemos obtener los coeficientes de regresión de Ridge para un nuevo valor de λ, digamos 50:

```
predict(malla_ridge, s = 50, type = "coefficients")[1:20, ]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
##           RBI         Walks         Years        CAtBat         CHits
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
```

```
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

Ahora dividimos la base de datos en un conjunto de entrenamiento y un conjunto de validación para estimar el MSE.

```
set.seed(1)
train <- sample(1:nrow(x), nrow(x) / 2)
#train
test <- (-train)
y.test <- y[test]
```

A continuación, obtenemos el modelo de Ridge en el conjunto de entrenamiento y evaluamos su MSE en el conjunto de validación, usando λ = ridge.train$lambda[75].

```
ridge.train <- glmnet(x[train, ], y[train], alpha=0, lambda = malla, thresh = 1e-12)
ridge.pred <- predict(ridge.train, s = ridge.train$lambda[75], newx = x[test, ])
```

MSE para este modelo:

```
mean((ridge.pred - y.test)^2)
```

```
## [1] 142577.4
```

Ahora comprobamos si hay algún beneficio en realizar la regresión Ridge con λ = ridge.train$lambda[75] en lugar de solo realizar la regresión de mínimos cuadrados.

Podemos realizarlo de dos maneras diferentes. Una es utilizando el modelo "ridge.train", con s=0:

```
ridgelineal.pred <- predict(ridge.train, s = 0, newx = x[test, ], exact = T, x = x[train, ], y = y[trai
mean((ridgelineal.pred - y.test)^2)
```

```
## [1] 168588.6
```

Y otra es utilizando lm:

```
#lm.train<-lm( Salary[train]~.,Hitters[train,  ])
lm.train<-lm(y ~ x, subset = train)
lm.pred<-predict(lm.train, Hitters)
mean( (y - lm.pred)[-train]^2 )
```

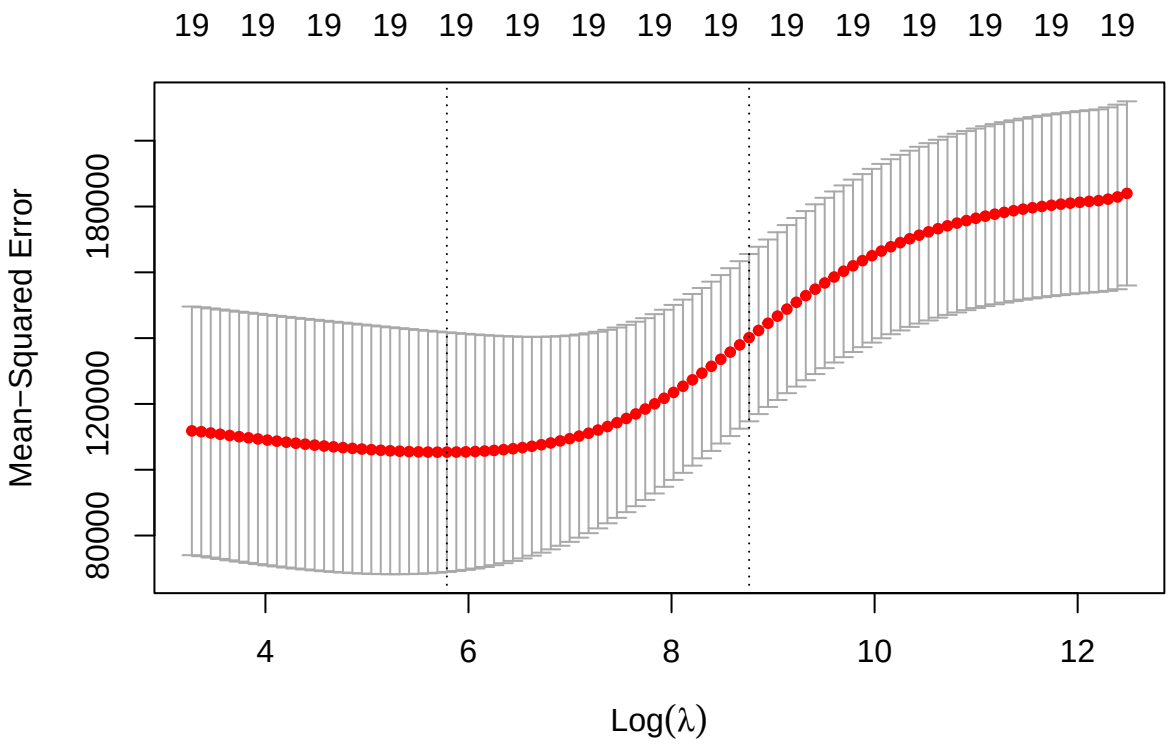
```
## [1] 168593.3
```

Observamos que el MSE con el modelo lineal es mayor.

SELECCIÓN DEL PARÁMETRO DE PENALIZACIÓN

En general, en lugar de elegir arbitrariamente λ, sería mejor usar la validación cruzada para elegir el parámetro λ con la función cv.glmnet(). Por defecto, se hace la CV con 10 iteraciones (10-fold CV).

```
set.seed(1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 326.0828
```

MSE estimado con el conjunto test, asociado con este valor de lambda:

```
k1=glmnet(x[train, ], y[train], alpha=0, lambda =bestlam)
ridge.pred <- predict(k1,newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
## [1] 139865.1
```

Finalmente no olvidemos que se debe realizar el modelo Ridge con toda la muestra para su posible uso con un test set "futuro"

```
ridge_TODO<-glmnet(x , y, alpha=0, lambda =bestlam)
coef(ridge_TODO)[,1]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  15.35054691   0.07732728   0.85869070   0.60323738   1.06375713   0.87966628
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.62423165   1.36199728   0.01136944   0.05748019   0.40672024   0.11441552
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12100983   0.05304031  22.08147382 -79.01444759   0.16613881   0.02936847
##       Errors   NewLeagueN
##  -1.35975125   9.13619606
```

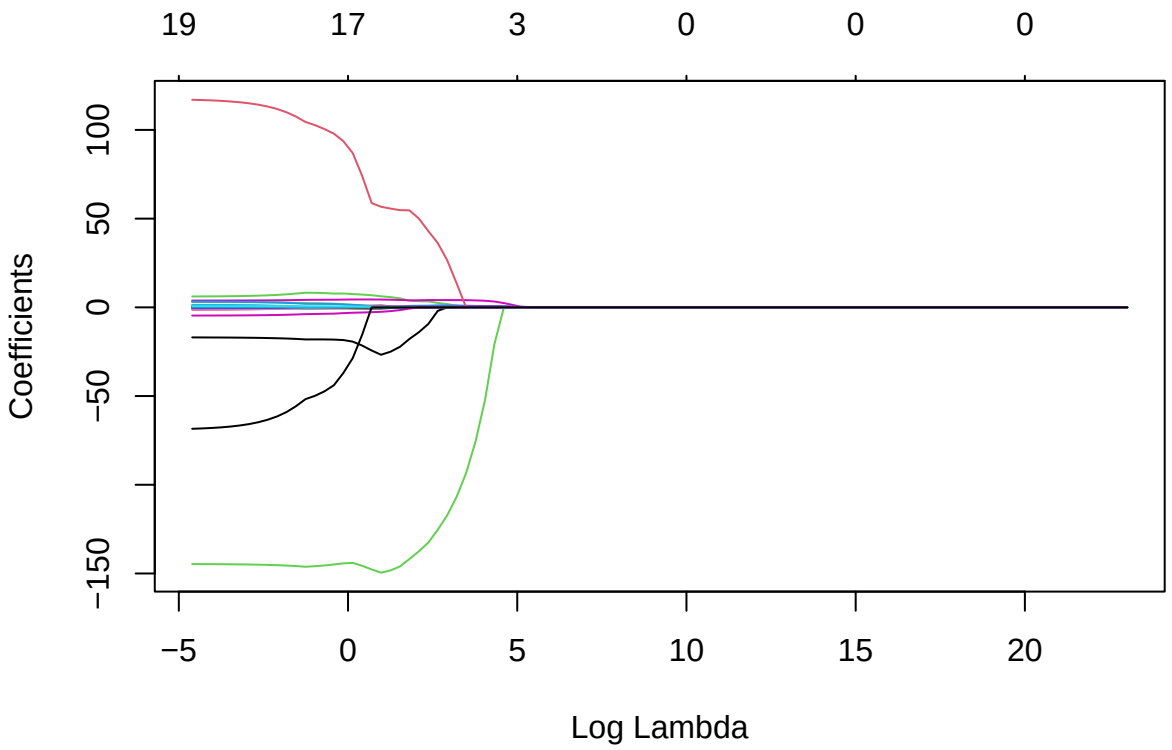
```
ridge.predT <- predict(ridge_TODO,newx = x)
mean((ridge.predT - y )^2)
```

```
## [1] 108300.7
```

Este paso en el libro se hace de manera diferente.

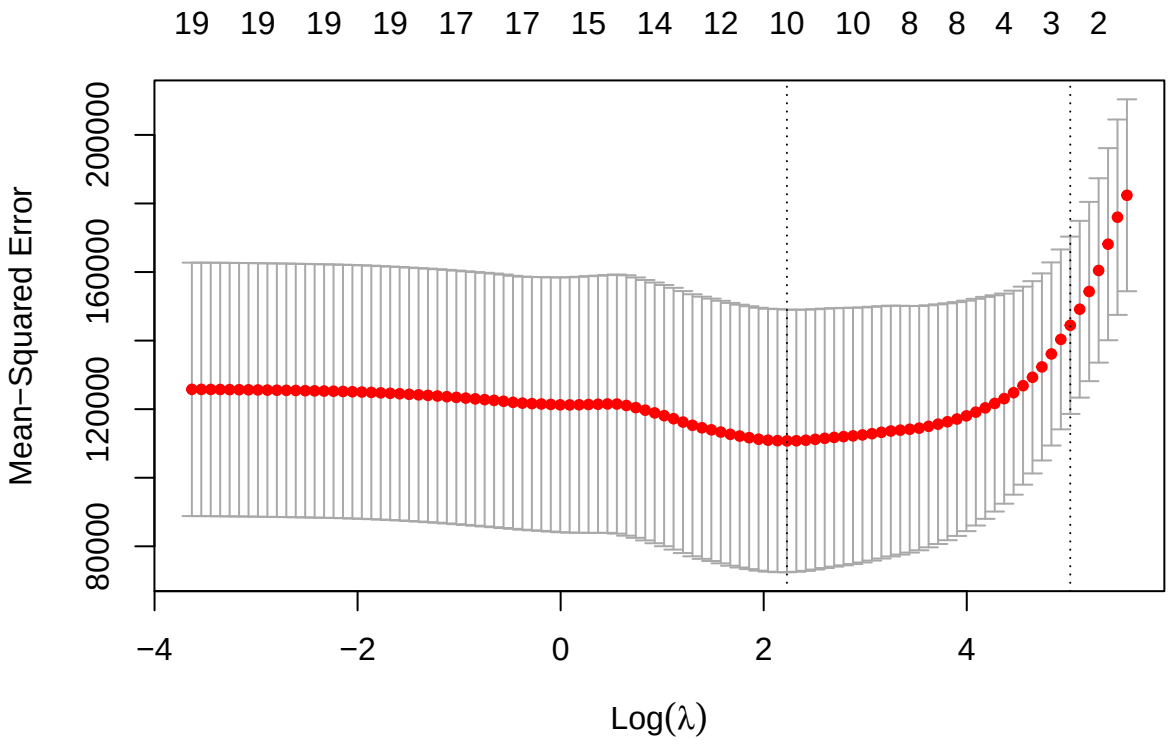
Lasso

```
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = malla)
plot(lasso.mod,xvar="lambda")
```



Podemos ver en el gráfico de coeficientes que, dependiendo de la elección del parámetro de ajuste λ, algunos de los coeficientes serán exactamente iguales a cero. Ahora realizamos la validación cruzada y calculamos el MSE.

```
set.seed(1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 9.286955
```

```
lasso.pred <- predict(lasso.mod, s = bestlam,newx = x[test, ])
mean((lasso.pred - y.test)^2)
```

```
## [1] 143673.6
```

Mejor que el del modelo lineal y parecido al de Ridge. Sin embargo, aquí hay 11 variables.

```
#coef(cv.out,s="lambda.min")
out <- glmnet(x, y, alpha = 1, lambda = malla)
lasso.coef <- predict(out, type = "coefficients",s = bestlam)[1:20, ]
lasso.coef[1:10]
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##  1.27479059 -0.05497143  2.18034583  0.00000000  0.00000000  0.00000000
##       Walks       Years      CAtBat       CHits
##  2.29192406 -0.33806109  0.00000000  0.00000000
```

También, si calculamos el modelo directamente con el mejor lambda, aunque no hallemos las mismas variables, la estimación de Salary es similar:

```
lasso.Directo<-glmnet(x, y, alpha = 1, lambda =  bestlam)
coef(lasso.Directo)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)   -3.42073206
## AtBat          .
## Hits           2.02965136
## HmRun          .
```

```
## Runs           .
## RBI            .
## Walks          2.24850782
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         0.04994886
## CRuns          0.22212444
## CRBI           0.40183027
## CWalks         .
## LeagueN       20.83775664
## DivisionW   -116.39019204
## PutOuts        0.23768309
## Assists        .
## Errors        -0.93567863
## NewLeagueN     .
```

```
lasso.predDir<-predict(lasso.Directo,newx = x)
mean((lasso.predDir - y )^2)
```

```
## [1] 103753.2
```

```
lasso.predDir[1:10]
```

```
##  [1] 535.08547 670.78147 963.41353 478.78965 592.78094 161.57711  88.40655
##  [8] 120.69735 873.87176 850.88865
```

```
predict(out, s = bestlam,newx = x)[1:10]
```

```
##  [1] 534.95601 673.60851 962.73265 479.37672 592.05484 161.84697  88.25532
##  [8] 120.87411 872.35873 850.53631
```