Model selection

In this notebook we're going to analyse different techniques for model selection and afterwards we're going to discuss their shortcomings.

Selection criteria

First of all, we're going to look at different criteria to compare models based on their performance and complexity.

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson      293   66     1   30  29    14     1    293    66      1
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson       30   29     14      A        E     446      33     20
## -Alan Ashby         321  414    375      N        W     632      43     10
## -Alvin Davis        224  266    263      A        W     880      82     14
## -Andre Dawson       828  838    354      N        E     200      11      3
## -Andres Galarraga    48   46     33      N        E     805      40      4
## -Alfredo Griffin    501  336    194      A        W     282     421     25
##                   Salary NewLeague
## -Andy Allanson        NA         A
## -Alan Ashby        475.0         N
## -Alvin Davis       480.0         A
## -Andre Dawson      500.0         N
## -Andres Galarraga   91.5         N
## -Alfredo Griffin   750.0         A
```

```
summary(Hitters)
```

```
##      AtBat            Hits         HmRun            Runs       
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00  
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25  
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00  
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91  
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00  
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00  
##                                                                
##       RBI             Walks            Years            CAtBat       
```

```
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
##
##      CHits            CHmRun           CRuns             CRBI
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00
##
##      CWalks        League  Division    PutOuts          Assists
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0
##  Median : 170.50                    Median : 212.0   Median : 39.5
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0
##
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146
##  Median : 6.00   Median : 425.0
##  Mean   : 8.04   Mean   : 535.9
##  3rd Qu.:11.00   3rd Qu.: 750.0
##  Max.   :32.00   Max.   :2460.0
##                  NA's   :59
```

```
# removing the NA
dim(Hitters)
```

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

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

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

We're going to use cross-validation to compare the results from different selection criteria.

```
nfolds <- 10
n <- dim(Hitters)[1]
folds <- cut(1:n, nfolds, labels = F)
# a bit of shuffling
indices <- sample(1:n, size=n, replace=F)
```

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 3.6.3
```

```
get.bss.test.error<- function(train, test, cv.best){
  # estimates the error on the test dataset for the best model
  # according to each criteria
  all.best<- regsubsets(x=Salary~.,data=train,nbest=1,
```

```
                        nvmax=dim(train)[2]-1, # using all variables
                        method="forward" )
  s <- summary(all.best)
  r2 <- coef(all.best, id=which.max(s$rsq))
  adjr2 <- coef(all.best, id=which.max(s$adjr2))
  cp <- coef(all.best, id=which.min(s$cp))
  bic <- coef(all.best, id=which.min(s$bic))
  cv.coefs <- coef(all.best, id=cv.best)
  # test predictions
  r2.pred <- model.matrix(Salary~.,test)[,names(r2)]%*%r2
  adjr2.pred <- model.matrix(Salary~.,test)[,names(adjr2)]%*%adjr2
  cp.pred <- model.matrix(Salary~.,test)[,names(cp)]%*%cp
  bic.pred <- model.matrix(Salary~.,test)[,names(bic)]%*%bic
  cv.pred <- model.matrix(Salary~.,test)[,names(cv.coefs)]%*%cv.coefs
  # test errors
  errors <- mean((r2.pred - test$Salary)**2)
  errors <- c(errors,mean((adjr2.pred - test$Salary)**2))
  errors <- c(errors,mean((cp.pred - test$Salary)**2))
  errors <- c(errors,mean((bic.pred - test$Salary)**2))
  errors <- c(errors,mean((cv.pred - test$Salary)**2))
  return(errors)
}


get.cv.error <- function(ncv, nmodels, data){
  # evaluates the mean cross-validation error of the linear model
  # with the selected coefficients
  n.cv <- dim(data)[1]
  folds.cv <- cut(1:n.cv, ncv, labels=F)
  cv.errors <- matrix(nrow = ncv, ncol = nmodels)
  indices.cv <- 1:n.cv
  for(j in 1:ncv){
    test.indices.cv <- indices.cv[folds.cv==j]
    test.cv <- data[test.indices.cv,]
    train.cv <- data[-test.indices.cv,]
    cv.all.best<- regsubsets(x=Salary~.,data=train.cv,
                             nbest=1,nvmax=nmodels, # using all variables
                             method="forward" )
     for(m in 1:nmodels){
       cv.coefs <- coef(cv.all.best, id=m)
       cv.preds <- model.matrix(Salary~.,test)[,names(cv.coefs)]%*%cv.coefs
       # test errors
       cv.errors[j,m] <- mean((cv.preds - test$Salary)**2)
    }
  }
  # selecting the model with the least mean error
  # expected test MSE estimated by CV for each model
  return(which.min(colMeans(cv.errors)))
}

test.errors <- matrix(nrow=nfolds, ncol=5)

for(i in 1:nfolds){
```

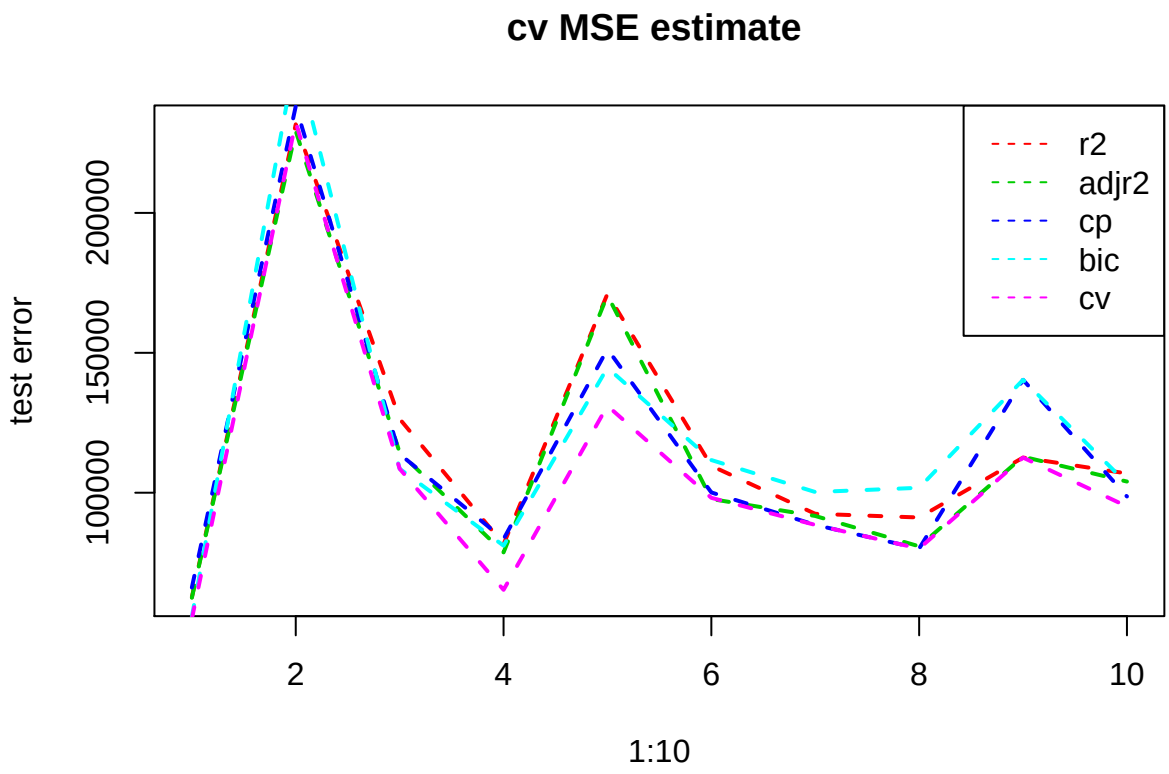
```
  test.indices <- indices[folds==i]
  test <- Hitters[test.indices,]
  train <- Hitters[-test.indices,]
  # Now we'll use BSS on the train dataset
  # And we'll record the error on the test set
  # get best cv model
  cv.best <- get.cv.error(ncv=5, nmodels=(dim(Hitters)[2]-1),data = train)
  test.errors[i,] <- get.bss.test.error(train=train, test=test, cv.best=cv.best)

}
```

Let's look at the results.

```
test.errors <- data.frame(test.errors)
names(test.errors) <- c("r2","adjr2","cp","bic","cv")
test.errors
```

```
##           r2     adjr2        cp       bic        cv
## 1   62648.71  62459.66  66117.19  55671.85  54921.37
## 2  231648.14 229260.83 238095.80 256368.21 232388.43
## 3  126387.88 114379.91 113808.59 108560.92 108560.92
## 4   81572.91  78677.81  83455.40  81090.08  65296.75
## 5  171064.88 170571.17 151326.87 144677.55 131074.43
## 6  109597.09  97750.16 100097.51 111676.16  98172.32
## 7   92460.84  91691.13  88503.70 100246.77  88503.70
## 8   91151.20  80840.38  80045.13 101764.46  80045.13
## 9  112559.56 112879.48 140339.93 140339.93 112555.44
## 10 106886.66 103997.50  98660.67 103277.30  95112.09
```

```
plot(1:10, test.errors$r2, type="l", lty="dashed", col=2, ylab="test error", main="cv MSE estimate ", lw
lines(1:10, test.errors$adjr2, type="l", lty="dashed", col=3, lwd=2)
lines(1:10, test.errors$cp, type="l", lty="dashed", col=4, lwd=2)
lines(1:10, test.errors$bic, type="l", lty="dashed", col=5, lwd=2)
lines(1:10, test.errors$cv, type="l", lty="dashed", col=6, lwd=2)
legend("topright", legend = c("r2", "adjr2","cp","bic","cv"), col=c(2,3,4,5,6), lty="dashed")
```




```
colMeans(test.errors)
```

```
##       r2    adjr2       cp      bic       cv
## 118597.8 114250.8 116045.1 120367.3 106663.1
```

```
which.min(colMeans(test.errors))
```

```
## cv
##  5
```

So the cross validation criteria seems to be the most reliable in model selection. We'll now use this criteria to select the best model fitting it on the whole data.

```
best.cv <- get.cv.error(ncv=10, nmodels=(dim(Hitters)[2]-1), data=Hitters)
best.cv
```

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

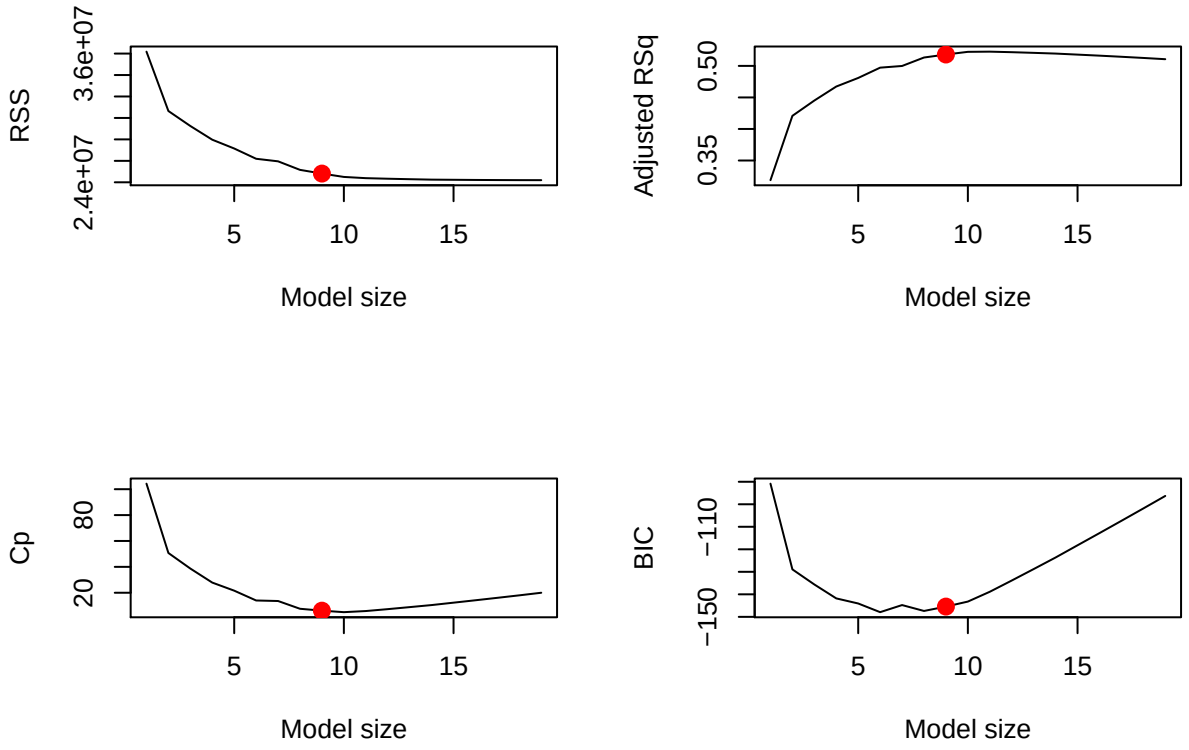
Let's now look at the best model:

```
nmodels <- 19
all.best <- regsubsets(x=Salary~.,data=Hitters,
                       nbest=1,nvmax=nmodels, # using all variables
                       method="forward")
coef(all.best, id=9)
```

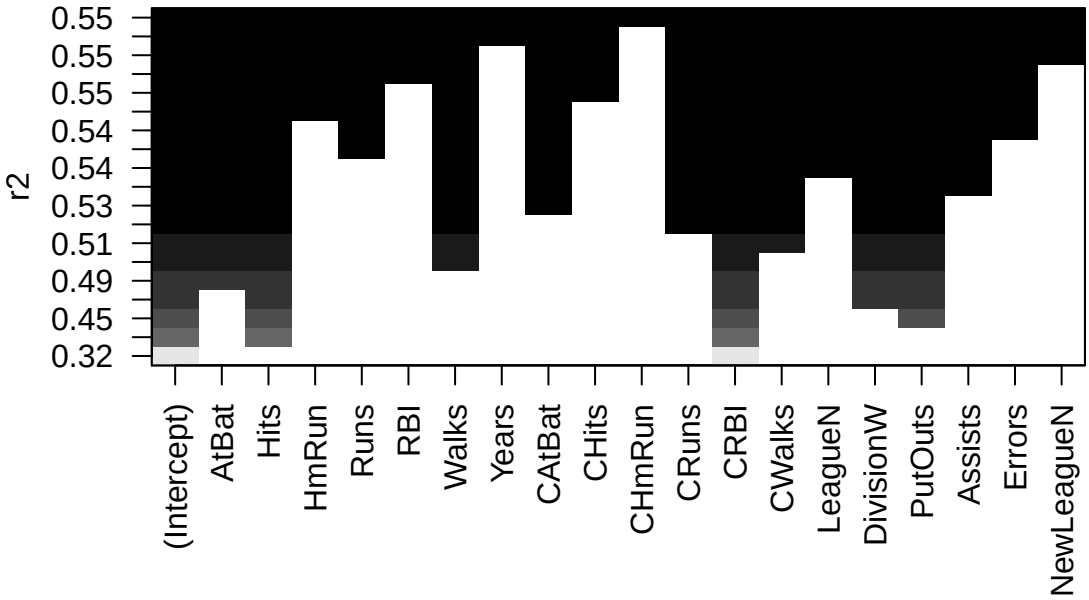
```
##   (Intercept)         AtBat          Hits         Walks        CAtBat
##  146.24960033   -1.93676754    6.65672102    5.55204413   -0.09953904
##         CRuns          CRBI        CWalks     DivisionW       PutOuts
##    1.25067124    0.66176849   -0.77798498 -115.34950146    0.27773062
```

Let's look at some plots to see how the best model is seen according to other criteria.

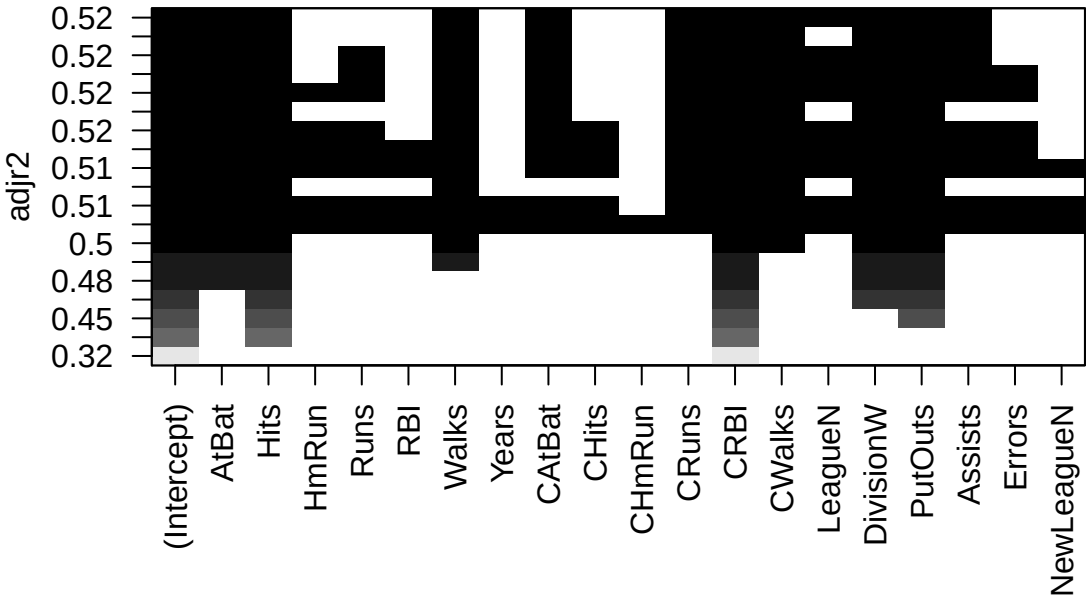
```
par(mfrow=c(2,2))
s <- summary(all.best)
# rss
plot(s$rss,xlab="Model size",ylab="RSS",type="l")
points(9, s$rss[9], col="red",cex=2,pch=20)
# adjr2
plot(s$adjr2,xlab="Model size",ylab="Adjusted RSq",type="l")
points(9, s$adjr2[9], col="red",cex=2,pch=20)
# Cp
plot(s$cp,xlab="Model size",ylab="Cp",type='l')
points(9, s$cp[9], col="red", cex=2, pch=20)
# BIC
plot(s$bic,xlab="Model size",ylab="BIC",type='l')
points(9, s$bic[9], col="red", cex=2, pch=20)
```



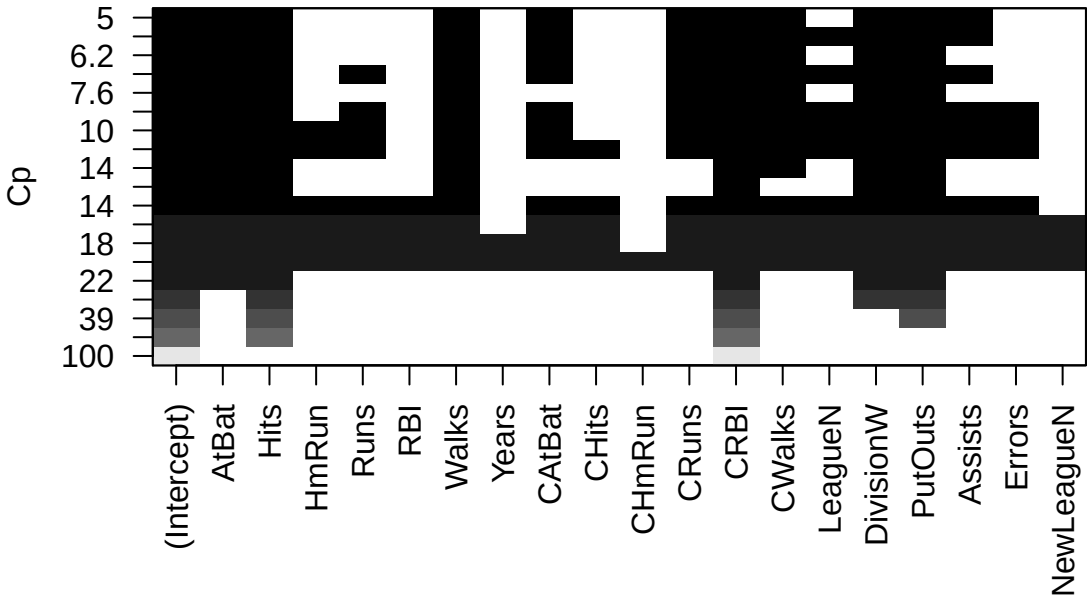

```
par(mfrow=c(1,1))
plot(all.best, scale="r2")
```



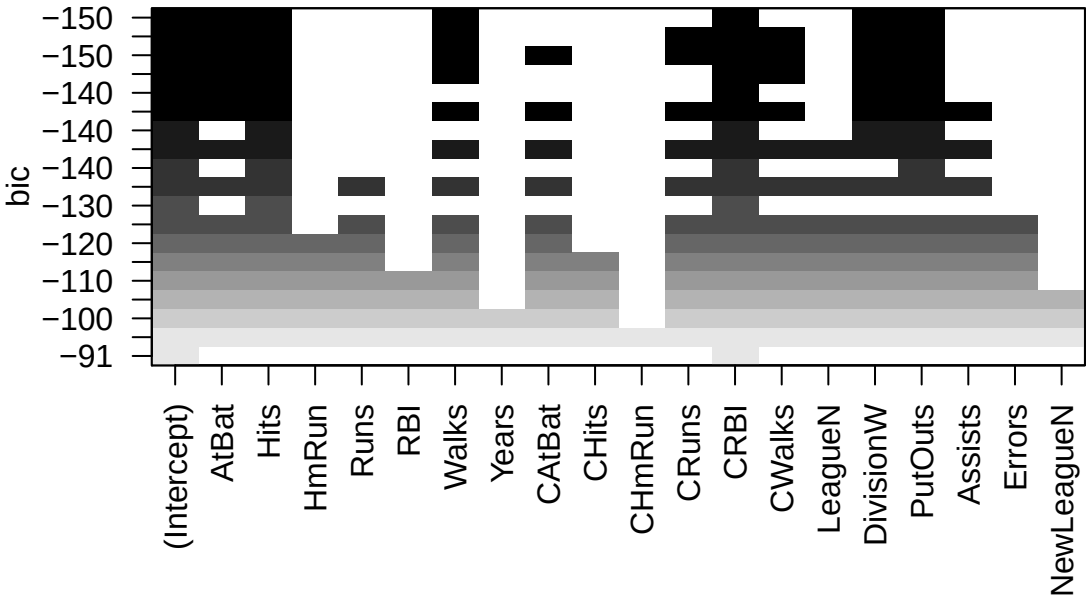

```
plot(all.best, scale="adjr2")
```




```
plot(all.best, scale="Cp")
```




```
plot(all.best, scale="bic")
```



Shrinkage methods

We'll now loook at a different set of selection tools: shrinkage methods like LASSO and RIDGE. We'll actually use the ElasticNet model, of which Lasso and RIdge are special cases.

```
require(hdi)
```

```
## Loading required package: hdi
## Warning: package 'hdi' was built under R version 3.6.3
## Loading required package: scalreg
## Loading required package: lars
## Loaded lars 1.2
```

```
data("riboflavin")
```

The riboflavin dataset records the riboflavin production by Bacillus subtilis together with the gene expressions. Each row refers to one gene, storing in x its expression level.

```
require(glmnet)
```

```
## Loading required package: glmnet
## Warning: package 'glmnet' was built under R version 3.6.3
## Loading required package: Matrix
## Loaded glmnet 3.0-2
```

```
attach(riboflavin)
lambda.grid <- grid <- 10^seq(10,-2, length = 100)
lasso <- glmnet(x = x, y=y, alpha = 1, lambda = lambda.grid)
ridge <- glmnet(x = x, y=y, alpha = 0, lambda = lambda.grid)
```

Exploration of the output:

```
dim(coef(lasso))
```

```
## [1] 4089  100
```

```
dim(coef(ridge))
```

```
## [1] 4089  100
```

```
lasso$lambda[50]
```

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

```
round(coef(lasso)[,50],2)[coef(lasso)[,50]!=0]
```

```
## (Intercept)
##       -7.16
```

```
ridge$lambda[50]
```

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

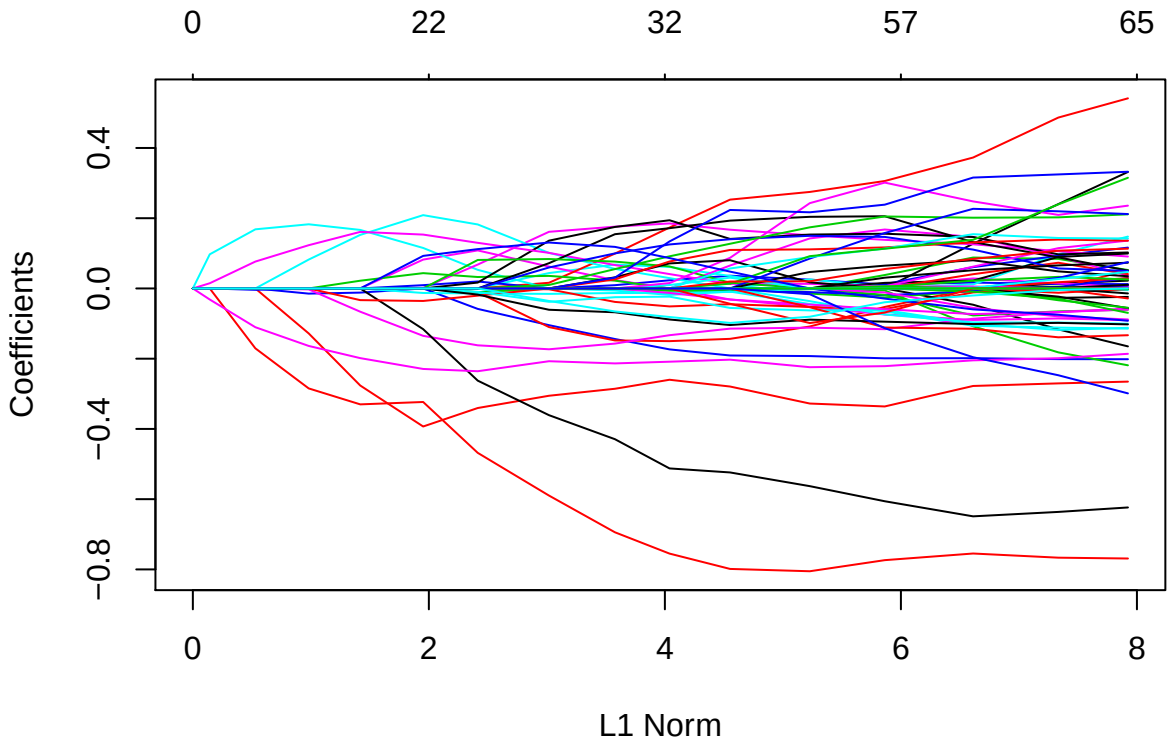
```
round(coef(ridge)[,50],2)[round(coef(ridge)[,50],2)!=0]
```

```
## (Intercept)
##       -7.15
```

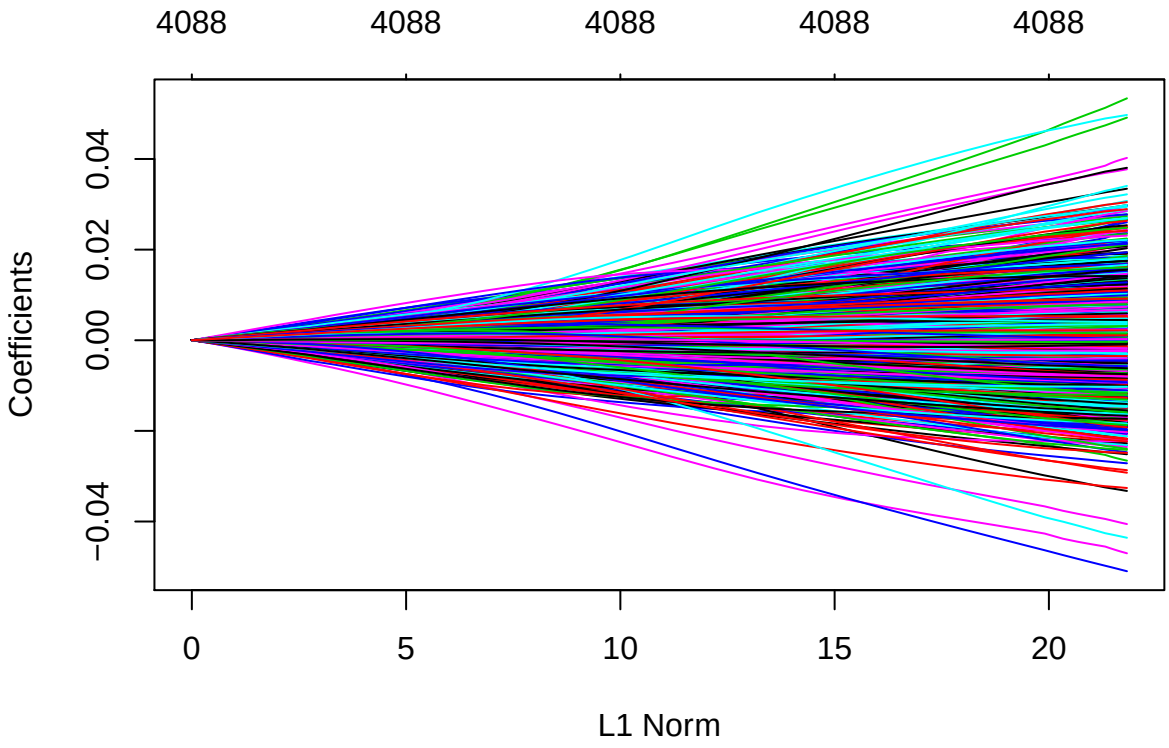
The above lambda is too high to allow any value to be different from zero. Let's have a broader look at the results with a plot:

```
plot(lasso)
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

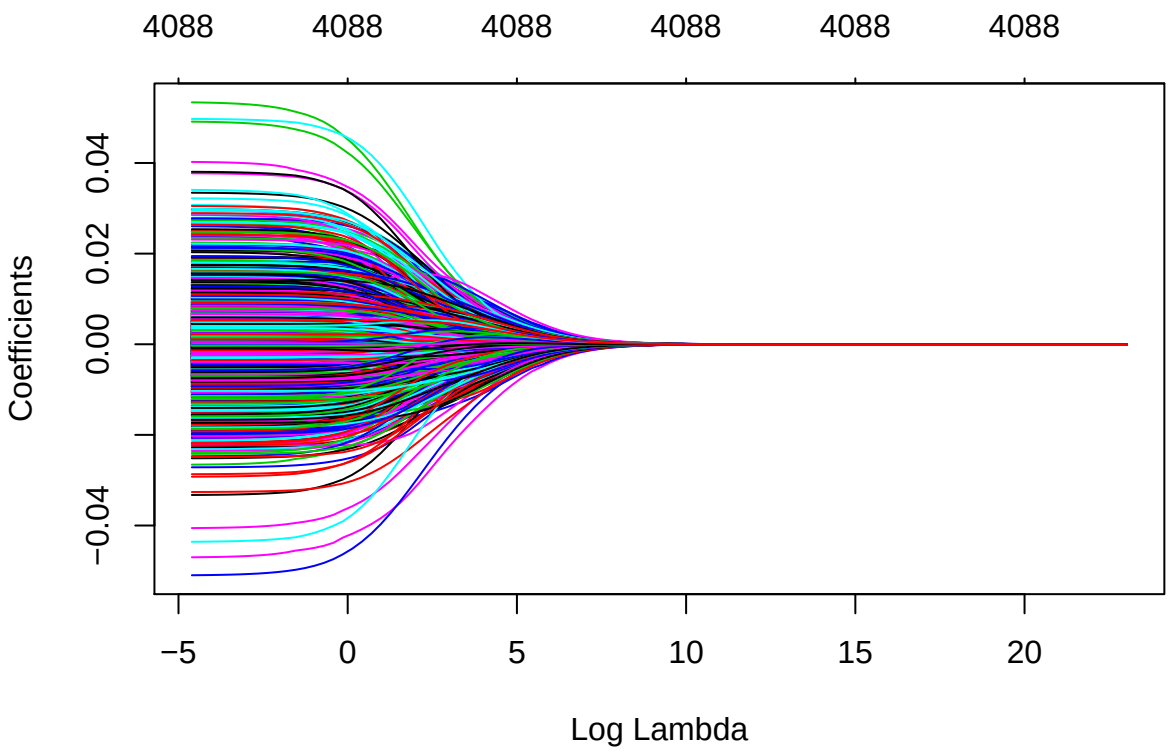



```
plot(ridge)
```



The above plots are a perfect synthesis of the differences between ridge and lasso: while ridge performs a "soft thresholding", slowly shrinking all variables to 0 as lambda increases, lasso performs a "hard thresholding", cutting off variables as they reach a certain threshold (determined by lambda).

```
plot(ridge, xvar="lambda")
```

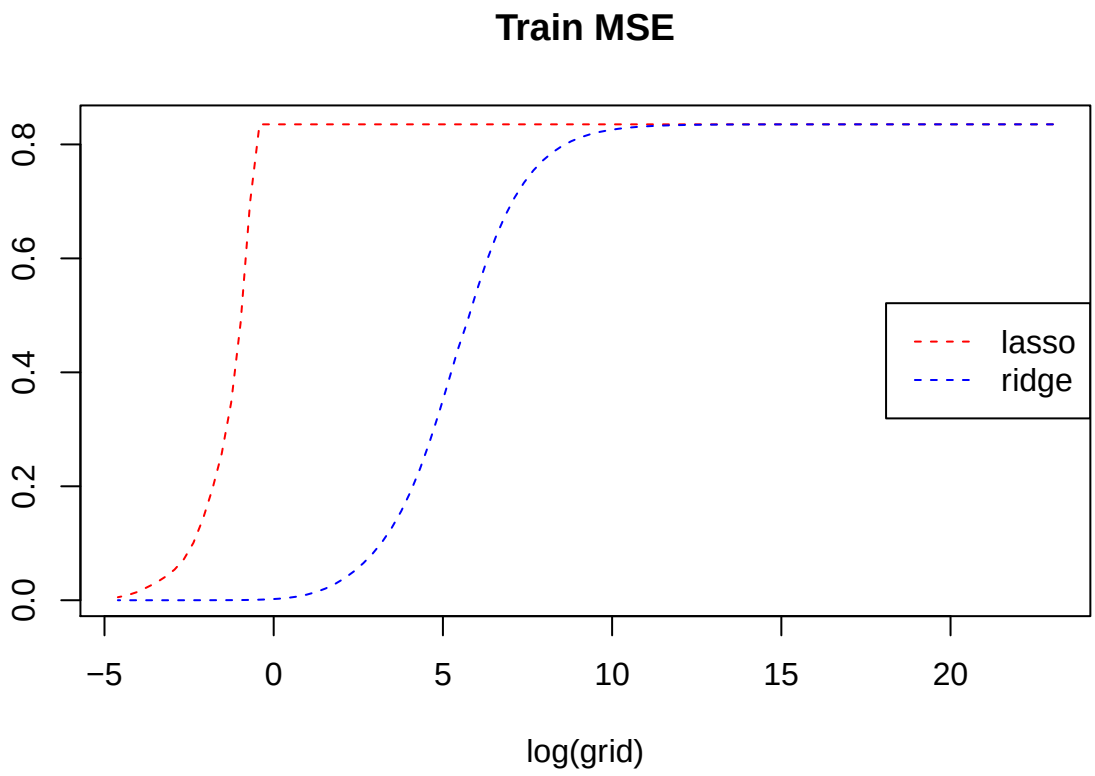


How to use a model for predictions?

```
preds <- predict(lasso, s = 0.01, newx=riboflavin$x)
mse.train <- mean((riboflavin$y - preds)**2)
mse.train
```

```
## [1] 0.005253187
```

```
preds.lasso <- predict(lasso, s = grid, newx=riboflavin$x)
plot(log(grid), colMeans((riboflavin$y - preds.lasso)**2), col="red", main="Train MSE", type="l", ylab =

preds.ridge <- predict(ridge, s = grid, newx=riboflavin$x)
lines(log(grid), colMeans((riboflavin$y - preds.ridge)**2), col="blue", type="l", ylab = "", lty="dashe

legend("right", legend=c("lasso","ridge"), col=c("red","blue"), lty="dashed")
```



Now let's use cross validation to do both model assessment and model selection: we're going to select the best lasso and ridge models on a train dataset and evaluate them with a second cross-validation against hold-out sets.

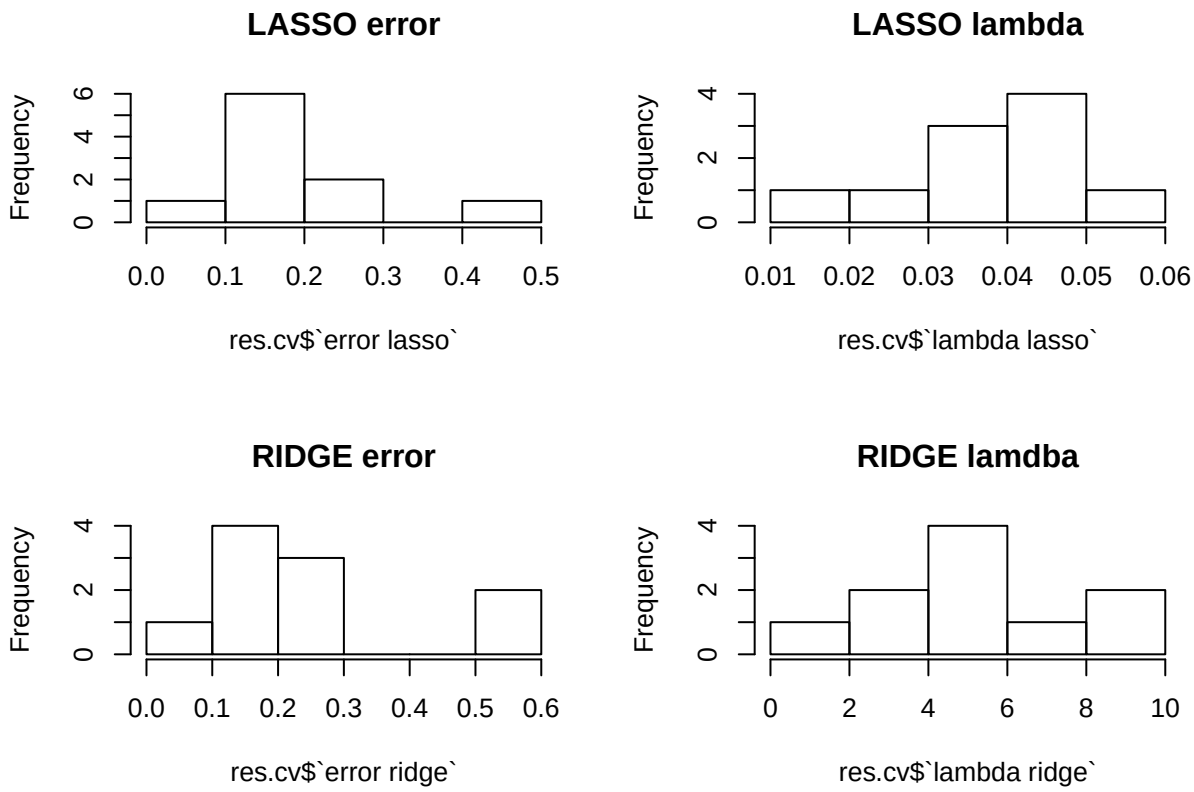
```
nfolds <- 10
n <- dim(riboflavin)[1]
folds <- cut(1:n, breaks = nfolds, labels = F)
#shuffling
indices <- sample(1:n, size=n, replace = F)
res.cv <- matrix(nrow=10, ncol=4)
for(i in 1:nfolds){
  test.indices <- indices[folds==i]
  test <- riboflavin[test.indices, ]
  train <- riboflavin[-test.indices, ]
  ## model selection using cv on the train dataset
  ## LASSO
  lasso.cv <- cv.glmnet(x=train$x, y=train$y,
                        alpha = 1, lambda = lambda.grid, nfolds =10)
  lasso.lambda <- lasso.cv$lambda.min
  res.cv[i,1] <- lasso.lambda
  lasso.fit <- glmnet(train$x, train$y, alpha =1, lambda = lasso.lambda)
  ## RIDGE
  ridge.cv <- cv.glmnet(x=train$x, y=train$y,
                        alpha = 0, lambda = lambda.grid, nfolds = 10)
  ridge.lambda <- ridge.cv$lambda.min
  res.cv[i,3] <- ridge.lambda
  ridge.fit <- glmnet(train$x, train$y, alpha =0, lambda = ridge.lambda)
```

```
  ## model assessment on the test dataset
  ## LASSO
  lasso.predict <- predict(lasso.fit, newx=test$x)
  lasso.error<- mean((test$y-lasso.predict)**2)
  res.cv[i,2]<-lasso.error
  ## RIDGE
  ridge.predict <- predict(ridge.fit, newx=test$x)
  ridge.error<- mean((test$y-ridge.predict)**2)
  res.cv[i,4]<-ridge.error
}
res.cv <- data.frame(res.cv)
names(res.cv) <- c("lambda lasso","error lasso","lambda ridge", "error ridge")
```

```
res.cv
```

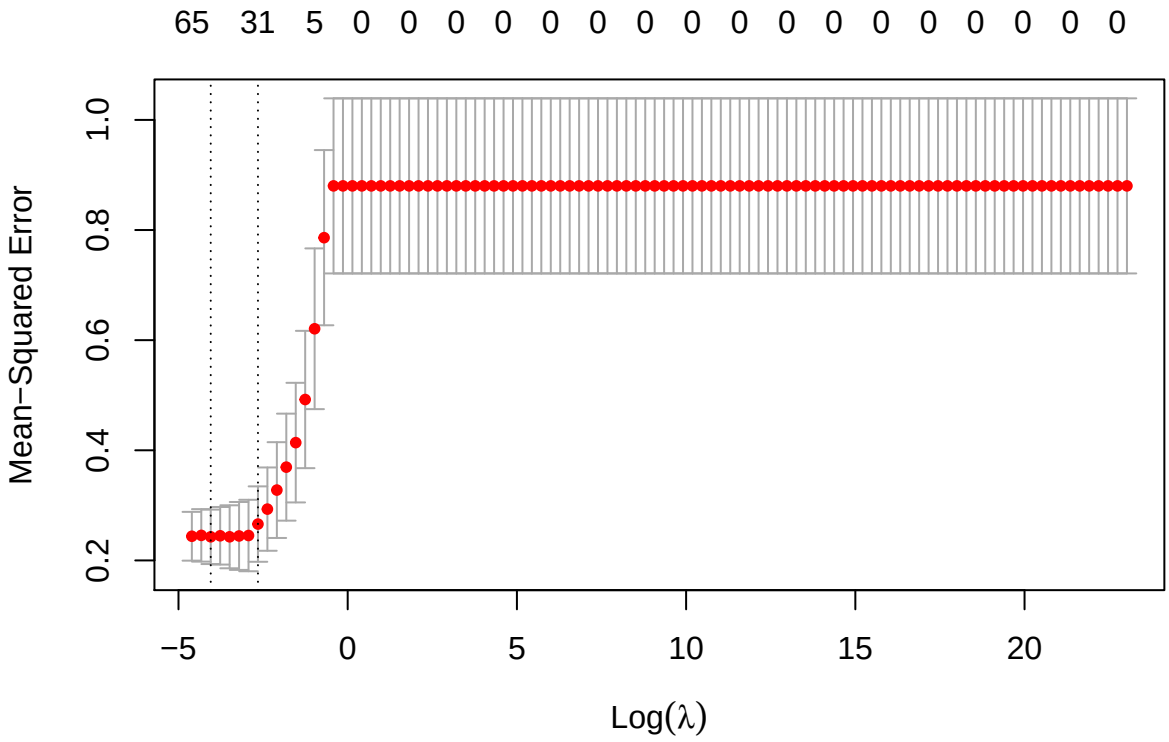
```
##    lambda lasso error lasso lambda ridge error ridge
## 1    0.04037017  0.45705220     4.641589  0.54365784
## 2    0.03053856  0.19270363     6.135907  0.21195031
## 3    0.01747528  0.20466899     0.869749  0.10625377
## 4    0.02310130  0.17999531     3.511192  0.29436952
## 5    0.04037017  0.13099458     4.641589  0.13060834
## 6    0.05336699  0.18280612     4.641589  0.26529239
## 7    0.04037017  0.29195843     8.111308  0.51758168
## 8    0.03053856  0.11686597     4.641589  0.11873909
## 9    0.04037017  0.08975441     3.511192  0.17214071
## 10   0.03053856  0.15526969     8.111308  0.09893368
```

```
par(mfrow=c(2,2))
hist(res.cv$`error lasso`, main="LASSO error")
hist(res.cv$`lambda lasso`, main="LASSO lambda")
hist(res.cv$`error ridge`, main="RIDGE error")
hist(res.cv$`lambda ridge`, main="RIDGE lamdba")
```



Not only the ridge presents a higher expected error on the test set, but it also has more variability in the lambda values. Therefore we're going to use Lasso, fitting it to the whole dataset.

```
lasso.cv <- cv.glmnet(x=x, y=y,alpha = 1, lambda = lambda.grid, nfolds =10)
plot(lasso.cv)
```




```
best.lambda <- lasso.cv$lambda.min
lasso.fit <- glmnet(x,y, alpha=1, lambda=best.lambda, thresh=1e-12)
sum(coef(lasso.fit)!=0)
```

```
## [1] 51
```

In the final model only 41 of the initial >4000 genes are active.

Selection bias

We'll now investigate the most common (and overlooked) mistake that comes with model selection.

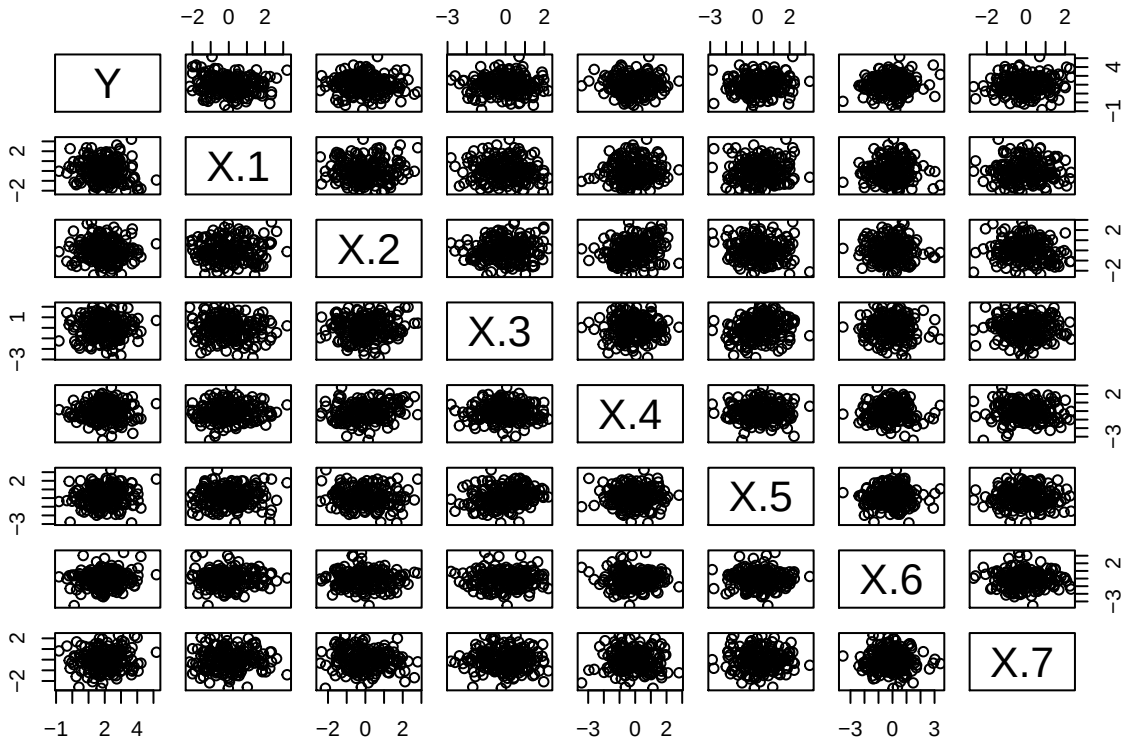
Let's start by generating some data.

```
n <- 200
sigma <- 1
intercept <- 2
y <- intercept + rnorm(n=n, mean=0, sd=sigma)
X <- matrix(rnorm(n*7, mean=0, sd=1), nrow=n, ncol=7)
data <- data.frame(Y=y, X=X)
head(data)
```

```
##          Y        X.1        X.2        X.3        X.4        X.5        X.6
## 1 2.113029 -1.5280196  0.5198705 -2.3838932 -1.7230770 -0.6695977 -0.4031820
## 2 2.573052 -0.3467834  0.8390241 -0.1267916 -0.2783374 -0.1292192 -0.6289025
## 3 1.629246 -0.6671402 -0.7315807 -1.5409877  1.0573607 -1.7230526 -0.8639732
## 4 2.669648 -0.5392800 -0.8666997  0.3543112 -0.8377383 -1.8826505 -0.1679860
## 5 4.053743 -1.8556788 -0.2829429 -1.2870113 -0.8293313 -0.3037580  1.3800388
```

```
## 6 2.662555 -0.4684841 -0.4453348 -0.4568530  1.0472535  0.2049943  1.0967805
##          X.7
## 1  0.3600737
## 2 -0.7227033
## 3  1.4347866
## 4  0.7789299
## 5 -1.0725668
## 6 -0.3401546
```

```
pairs(data)
```



So, there's no relationship between Y and X, but let's suppose we don't know it and to make it more realistic let's give some fancy names to the X variables.

```
names(data)<-c("HealthIdx","Poverty12mo","MedianIncome","`%Obese"," InjuryRate","HeartRisk", "noConvict
head(data)
```

```
##   HealthIdx Poverty12mo MedianIncome    `%Obese  InjuryRate  HeartRisk
## 1  2.113029  -1.5280196    0.5198705 -2.3838932  -1.7230770 -0.6695977
## 2  2.573052  -0.3467834    0.8390241 -0.1267916  -0.2783374 -0.1292192
## 3  1.629246  -0.6671402   -0.7315807 -1.5409877   1.0573607 -1.7230526
## 4  2.669648  -0.5392800   -0.8666997  0.3543112  -0.8377383 -1.8826505
## 5  4.053743  -1.8556788   -0.2829429 -1.2870113  -0.8293313 -0.3037580
## 6  2.662555  -0.4684841   -0.4453348 -0.4568530   1.0472535  0.2049943
##    noConvict FamilyIssue
## 1 -0.4031820   0.3600737
## 2 -0.6289025  -0.7227033
## 3 -0.8639732   1.4347866
```

```
## 4 -0.1679860   0.7789299
## 5  1.3800388  -1.0725668
## 6  1.0967805  -0.3401546
```

And now let's use our super-powerful model selection tools to look for the best way to model this data.

```
library(leaps)
p <- dim(data)[2]
bss.res <- regsubsets(HealthIdx~., data=data, nbest = 1, nvmax =p)
s <- summary(bss.res)
coef(bss.res, id=which.min(s$bic))
```

```
## (Intercept) Poverty12mo
##   1.9843180  -0.1222146
```

Okay our Best subset selection is telling us noCOnvict is the best predictor for our health index. Let's look at its p-value by fitting a linear model to it.

```
fit.bss <- lm(HealthIdx~noConvict, data=data)
summary(fit.bss)
```

```
##
## Call:
## lm(formula = HealthIdx ~ noConvict, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.83145 -0.57975 -0.00058  0.65623  3.14911
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.98789    0.06823  29.137   <2e-16 ***
## noConvict    0.09190    0.06532   1.407    0.161
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9636 on 198 degrees of freedom
## Multiple R-squared:  0.0099, Adjusted R-squared:  0.0049
## F-statistic:  1.98 on 1 and 198 DF,  p-value: 0.161
```

The p-value of no convict is not low enough, so no discovery! But will the p-value always save us from false discoveries or was this just luck? But what if we repeat this approach multiple times?

```
nsim <- 1000
# we're going to record all p-values of the selected variables for each simulation
p.values <- matrix(nrow=nsim, ncol=p)
p.values <- data.frame(p.values)
names(p.values)<- names(data)[-1]
create.formula<- function(selected){
  formula <- selected[1]
  if(length(selected)>1){
    for(i in 2:length(selected)){
    formula <- paste(formula, selected[i], sep="+")
    }
  }
  return(formula)
}
```

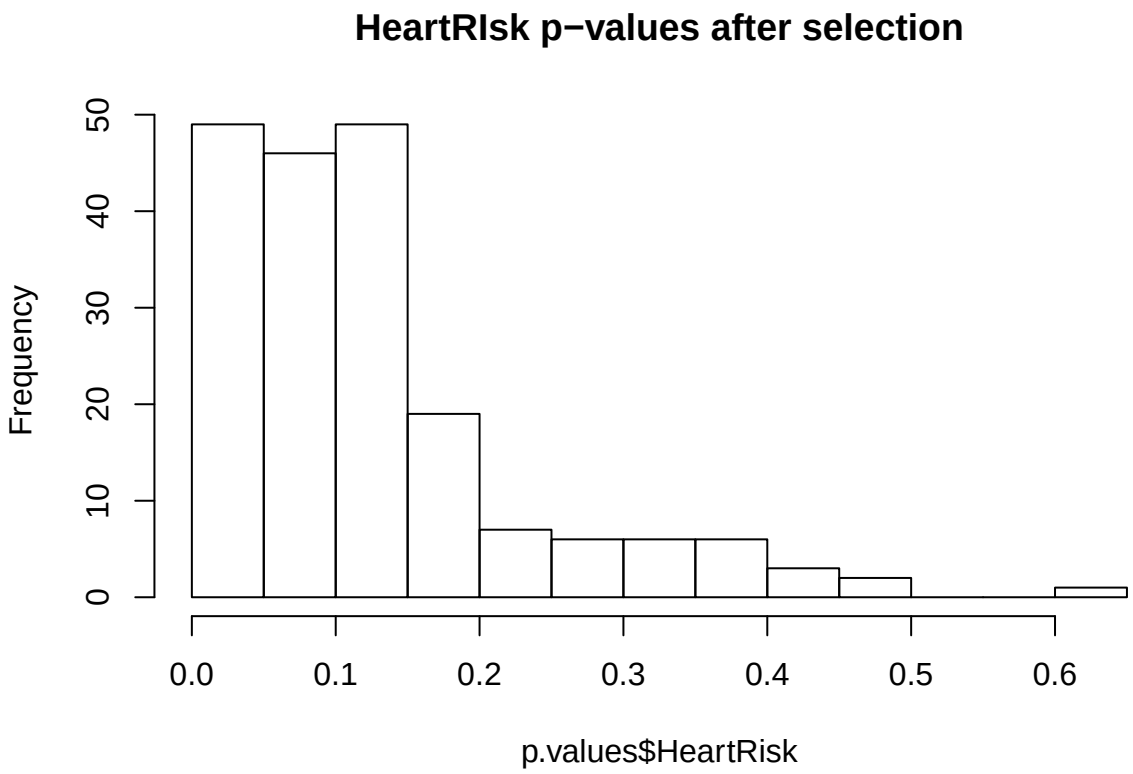
```
for(j in 1:nsim){
  ##Let's draw again some data from the noise
  data$HealthIdx <- intercept + rnorm(n=n, mean=0, sd=sigma)
  ##Model selection
  bss.res <- regsubsets(HealthIdx~., data=data, nbest = 1, nvmax =p)
  s <- summary(bss.res)
  selected <- names(coef(bss.res, id=which.min(s$cp)))[-1]
  ##Model assessment
  fit <- lm(paste("HealthIdx~",create.formula(selected)), data=data)
  s <- summary(fit)
  # extracting pvalues
  pvalues <- s$coefficients[selected,4]
  p.values[j, selected]=pvalues
}
```

```
head(p.values)
```

```
##   Poverty12mo MedianIncome `%Obese  InjuryRate HeartRisk noConvict FamilyIssue
## 1          NA           NA      NA          NA        NA        NA  0.10871330
## 2          NA           NA      NA          NA 0.1918373        NA          NA
## 3          NA           NA      NA          NA        NA 0.1052738  0.08596629
## 4   0.2993023           NA      NA          NA        NA        NA          NA
## 5          NA           NA      NA          NA        NA        NA          NA
## 6          NA           NA      NA          NA 0.3095082        NA          NA
##   NA `\\`%Obese` ` InjuryRate`
## 1 NA          NA            NA
## 2 NA          NA            NA
## 3 NA          NA            NA
## 4 NA          NA            NA
## 5 NA   0.2961612            NA
## 6 NA          NA            NA
```

Let's have a look at our results!

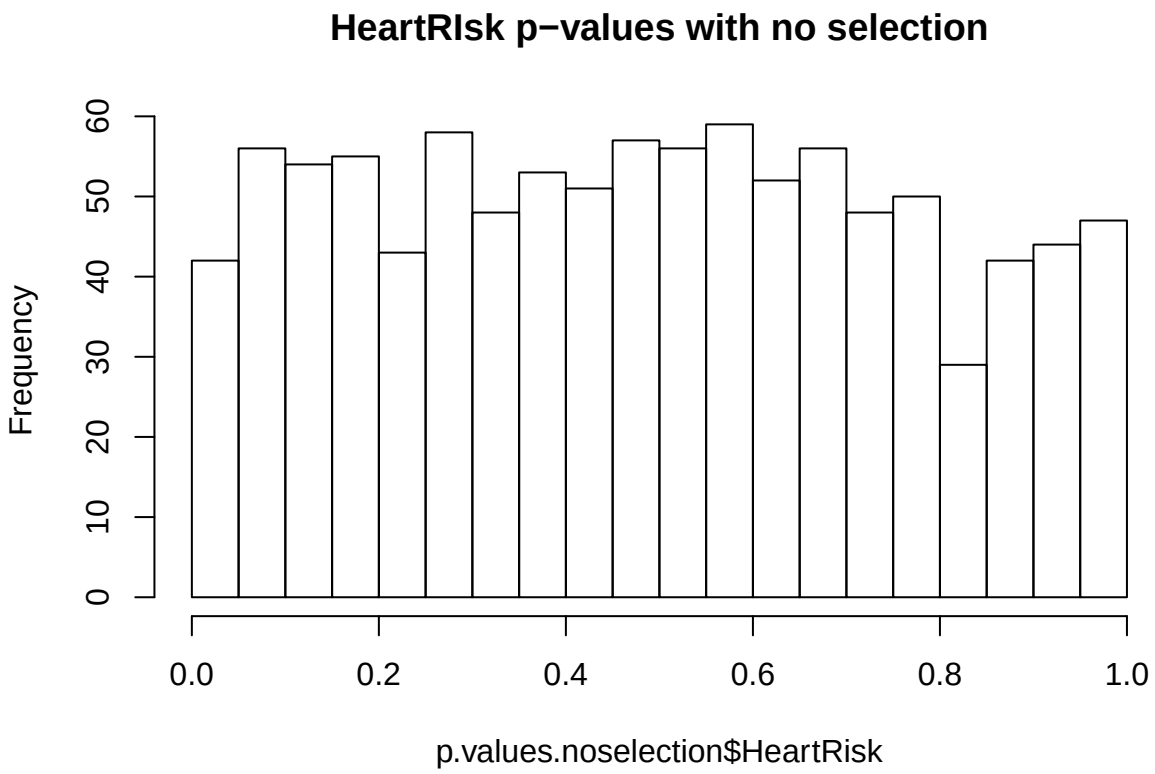
```
hist(p.values$HeartRisk, main="HeartRIsk p-values after selection", breaks=20)
```



Let's now look at more honest p-values: the ones we get by fitting on the whole model, with no model selection.

```
nsim <- 1000
# we're going to record all p-values of the selected variables for each simulation
p.values.noselection <- matrix(nrow=nsim, ncol=p)
p.values.noselection <- data.frame(p.values.noselection)
names(p.values.noselection)<- names(data)[-1]

for(j in 1:nsim){
  ##Let's draw again some data from the noise
  data$HealthIdx <- intercept + rnorm(n=n, mean=0, sd=sigma)
  # No model selection this time
  selected <- names(data)[-1] # all but the response
  fit <- lm("HealthIdx~.",data=data)
  s <- summary(fit)
  # extracting pvalues
  pvalues <- s$coefficients[-1,4]
  p.values.noselection[j, selected]=pvalues
}
```

```
hist(p.values.noselection$HeartRisk, main="HeartRIsk p-values with no selection", breaks=20)
```



Now this is the true distribution of the p-values under the null. Why didn't we get this same distribution above? Because we applied what is called the *selection bias* to our analysis. Any time we use the data to make a decision (e.g. pick one model instead of some others), we introduce a selection effect (bias). What is wrong with the naive approach we've seen first is that it's not taking into account that the test we're conducting is *conditioned on* the fact that that specific model has already been selected.

Forward stepwise, Lasso, elastic net with cross-validation, etc, all use the data in a way that would result in such bias. Significance tests, prediction error, R2, goodness of fit tests, etc, will all suffer from selection bias.

So, how do we solve this?

Solutions

The idea is basically to account for the conditioning, or, put in another way: if a variable "surprises" us enough to be included in the model, it must surprise us again in order to be declared significant.

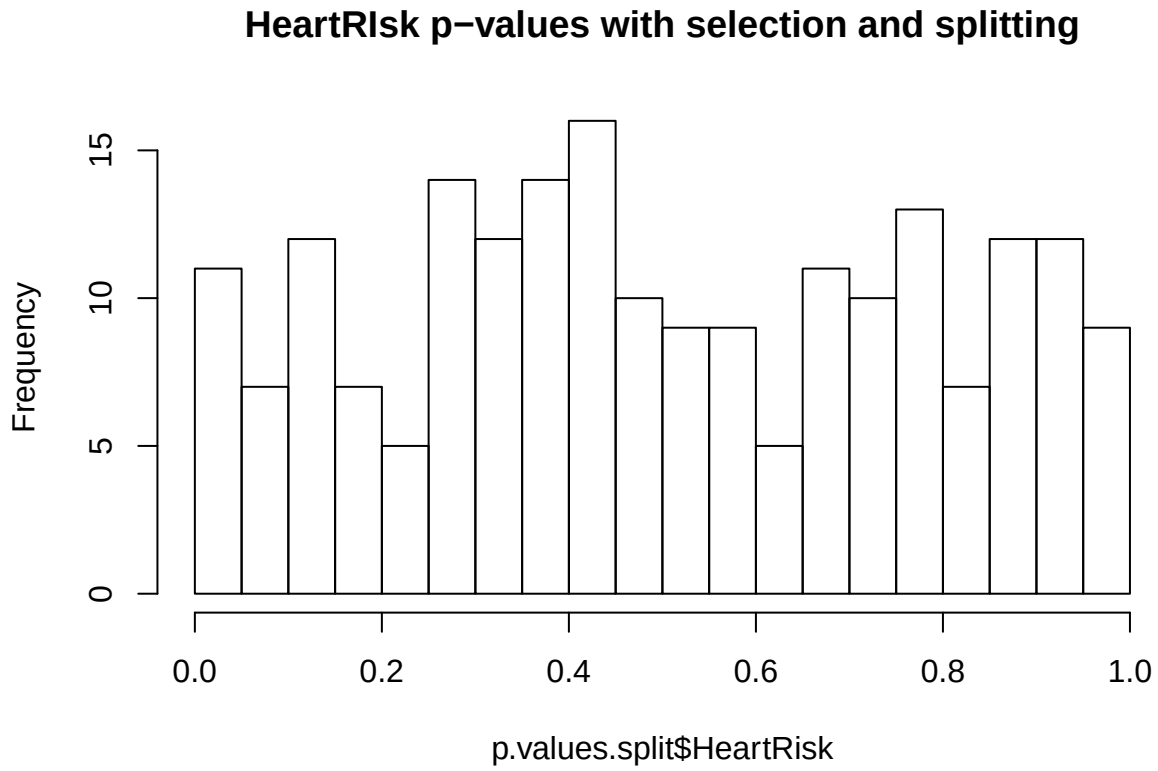
The first and easiest solution is to do what we've already done above: fit the whole model and look at those p-values. However, we should take into consideration a multiple testing issue and adjust our tests accordingly, since we're testing for the 0.5% on all the predictors.

Another solution is to simply split the data in train and test set and NEVER use the test set before we've completed all the tests, selections and fitting. Let's look at the results of this approach on the above experiment:

```
nsim <- 1000
# we're going to record all p-values of the selected variables for each simulation
p.values.split <- matrix(nrow=nsim, ncol=p)
p.values.split <- data.frame(p.values.split)
names(p.values.split)<- names(data)[-1]
```

```r
#we'll remove the noise that comes from the splitting and pre-determine the splitting before the simula
train <- sample(c(TRUE,FALSE), size=(n*3/4), replace = T)
test <- (!train)

for(j in 1:nsim){
  ##Let's draw again some data from the noise
  data$HealthIdx <- intercept + rnorm(n=n, mean=0, sd=sigma)
  ##Model selection on the training set
  bss.res <- regsubsets(HealthIdx~., data=data[train,], nbest = 1, nvmax=p)
  s <- summary(bss.res)
  selected <- names(coef(bss.res, id=which.min(s$cp)))[-1]
  ##Model assessment on the test set
  fit <- lm(paste("HealthIdx~",create.formula(selected)), data=data[test,])
  s <- summary(fit)
  # extracting pvalues
  pvalues <- s$coefficients[selected,4]
  p.values.split[j, selected]=pvalues
}
```

```r
hist(p.values.split$HeartRisk, main="HeartRIsk p-values with selection and splitting", breaks=20)
```



Again we obtain honest p-values, which confirms the train-test splitting as a valid and selection-bias-free procedure. However, for how simple and robust (assumptions-wise) the splitting approach might be it has some drawbacks, the main one is the lack of reproducibility, due to the randomness introduced by the splitting.

Some new research is working on *selective error control*, and here are some useful slides if you want to know

more about it: http://joshualoftus.com/turing/shorttalk.pdf .