Laboratory Project- Regularized Regression Models & variable selection - partial correction

Julien Chiquet & Anastase Alexandre Charantonis & Guillem Rigaill

October 20th, 2016

Study of housing values

The `Boston` dataset from the *MASS* package reports the median value of owner-occupied homes in about 500 U.S. census tracts in the Boston area, together with several variables which might help to explain the variation in median value across tracts. The corresponding `data.frame` contains the following columns

- `crim`: crime rate
- `zn`: proportion of 25,000 square feet residential lots
- `indus`: proportion of nonretail business acres
- `chas`: is the tract bounds the Charles River ?
- `nox`: annual average nitrogen oxide concentration in parts per hundred million
- `rm`: average number of rooms
- `age`: proportion of owner units built prior to 1940
- `dis`: weighted distances to five employment centers in the Boston area
- `rad`: index of accessibility to radial highways
- `tax`: full value property tax rate ($/$10,000)
- `ptratio`: pupil/teacher ratio
- `black`: proportion of blacks in the population
- `lstat`: proportion of population that is lower status
- `medv`: median value of owner–occupied homes

The objective is to construct a linear model that has good predictive properties for the variable `medv`. Furthermore, we would prefer to only use predictors that are dirrectly tied to the response of the model, and therefore we shall proceed to perform a selection of variables.

First part: introduction

1.1 Preliminaries

- Load the `Boston` dataset

```
library(MASS)
data(Boston)
```

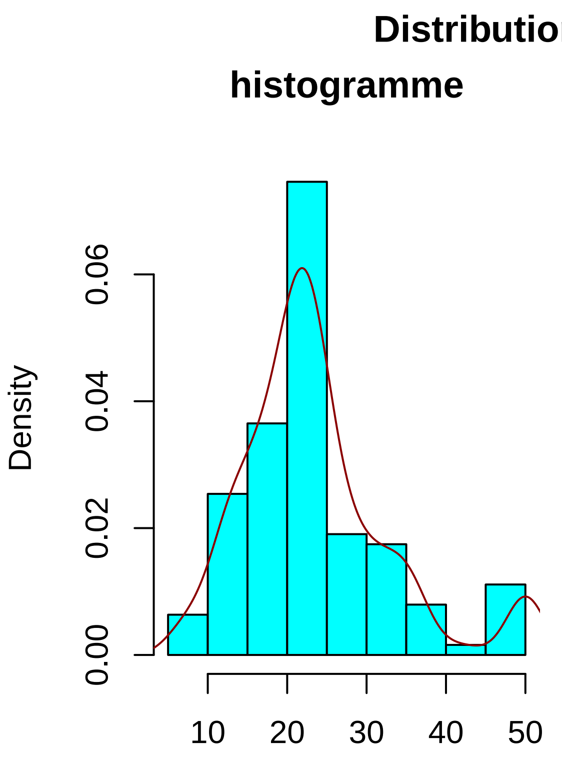
- Based on this table create two new matrices named `Boston.train` and `Boston.test`. The first one will be used to train your models and the second one will serve as a validation dataset.

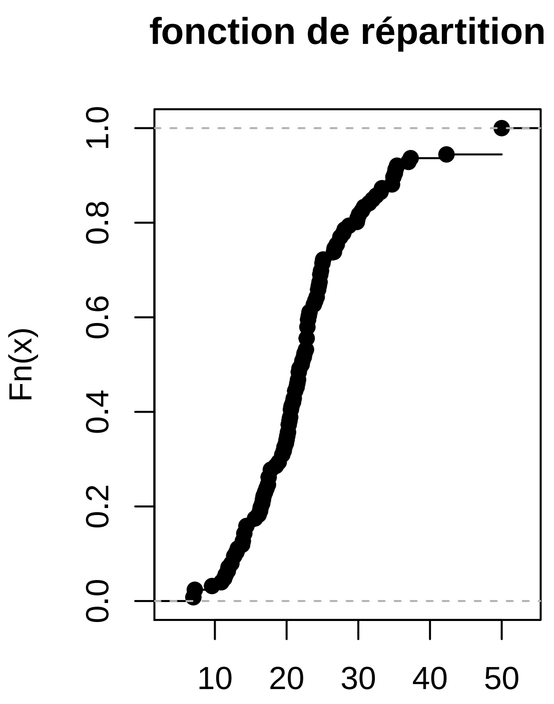
```
n <- nrow(Boston) ; train <- sample(1:n, round(n) / 4)
Boston.train <- Boston[ train, ]
Boston.test  <- Boston[-train, ]
```

1.2 Descriptive analysis

Briefly describe the data: give in broad strokes the main tendeces contained in the dataset, especially those connected to the variabe we are atempting to explain. Are there any predictors that appear redudnant?

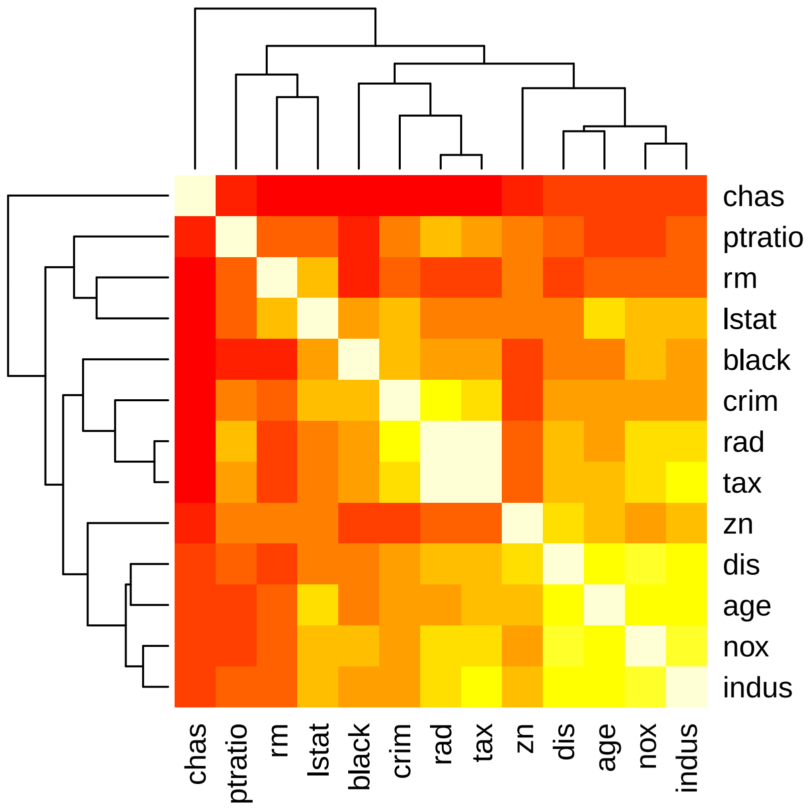
```r
par(mfrow=c(1,2))
hist(Boston.train$medv, freq=FALSE, col="cyan", main="histogramme",xlab="")
lines(density(Boston.train$medv), col="darkred")
plot(ecdf(Boston.train$medv), main="fonction de répartition",xlab="")
title(outer=TRUE, main="\n Distribution du loyer median")
```





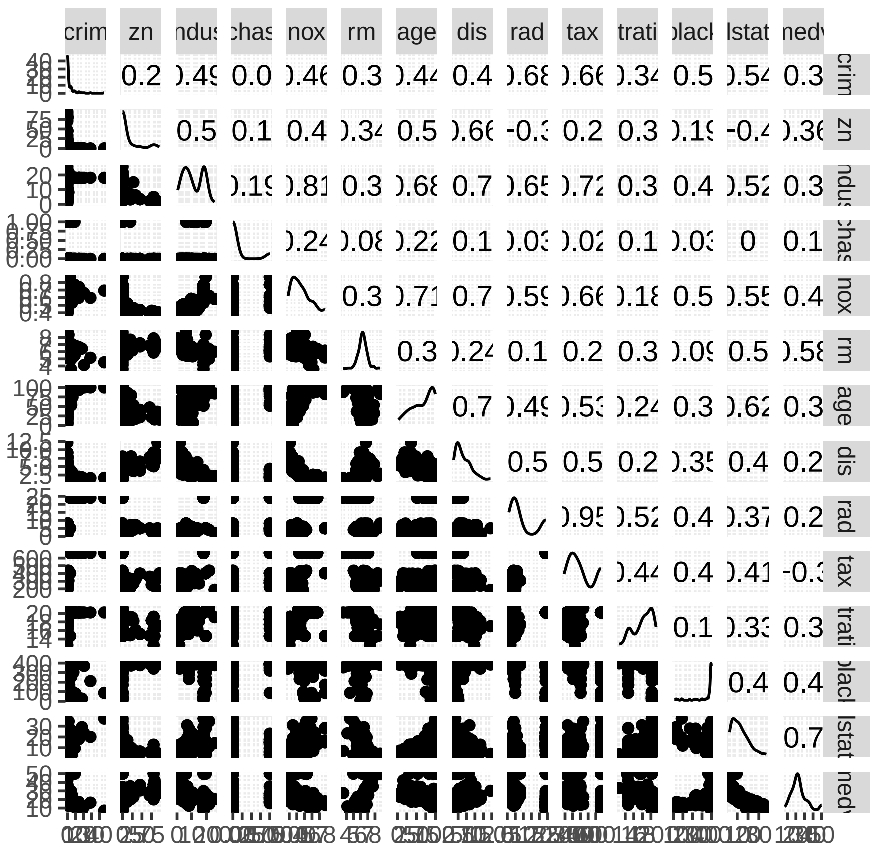
Some of variables appear to be redundant:

```r
heatmap(abs(cor(Boston.train[, -14])), symm=TRUE)
```



Using dispersion diagrams we can present the similarity of the response, the body mass index and the first serological variable.

```
library(GGally)
ggscatmat(Boston.train)
```



Second part: multilinear regression and variable selection

2.1 Multilinear regression

Generate a multilinear model containing all the prefictors (we will refer to this model as `full`). Run a diagnostic of the model. Do the same with the model containing only the intercept (we will refer to this model as `null`). Compare the two models and comment.

Generating the two models :

```
null <- lm(medv~1, Boston.train)
full <- lm(medv~., Boston.train)
```

As expected, the we reject the nul hypothesis.

```
anova(null,full)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ 1
## Model 2: medv ~ crim + zn + indus + chas + nox + rm + age + dis + rad +
##     tax + ptratio + black + lstat
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1    125 11006
## 2    112  3408 13    7597.8 19.207 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

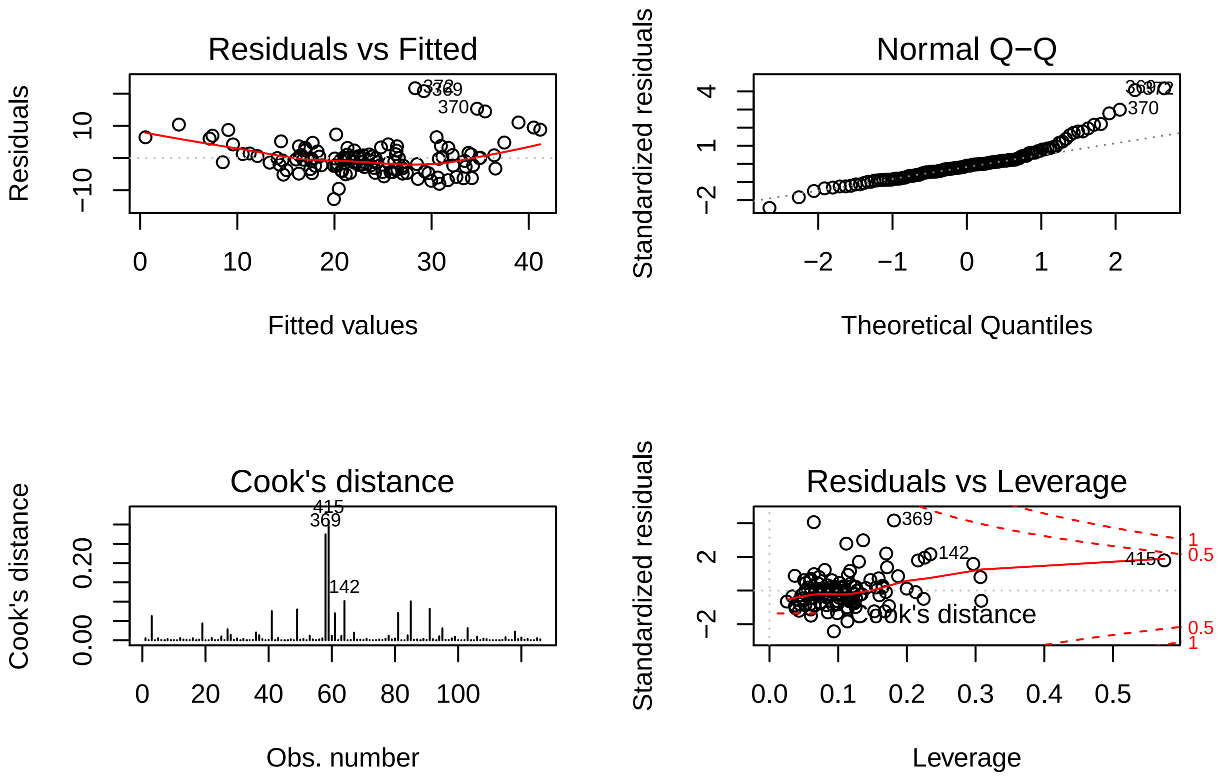
The R^2 parameter is rather good:

```
summary(full)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.7486  -3.1713  -0.5271   1.3795  21.6948
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  38.387307  13.033316   2.945 0.003925 **
## crim          0.061619   0.134852   0.457 0.648602
## zn            0.076310   0.031567   2.417 0.017248 *
## indus        -0.020256   0.156333  -0.130 0.897141
## chas          1.505521   1.674610   0.899 0.370567
## nox         -16.520251   9.042122  -1.827 0.070358 .
## rm            2.572210   0.927404   2.774 0.006497 **
## age           0.033243   0.030963   1.074 0.285290
## dis          -1.651966   0.465154  -3.551 0.000562 ***
## rad           0.230607   0.209188   1.102 0.272657
## tax          -0.010411   0.011373  -0.915 0.361917
## ptratio      -0.712118   0.326178  -2.183 0.031106 *
## black         0.014715   0.006796   2.165 0.032478 *
## lstat        -0.718331   0.115011  -6.246  7.8e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.516 on 112 degrees of freedom
## Multiple R-squared:  0.6903, Adjusted R-squared:  0.6544
## F-statistic: 19.21 on 13 and 112 DF,  p-value: < 2.2e-16
```

Now for the diagnostic plots:

```
par(mfrow=c(2,2))
plot(full, which=c(1,2,4,5))
```

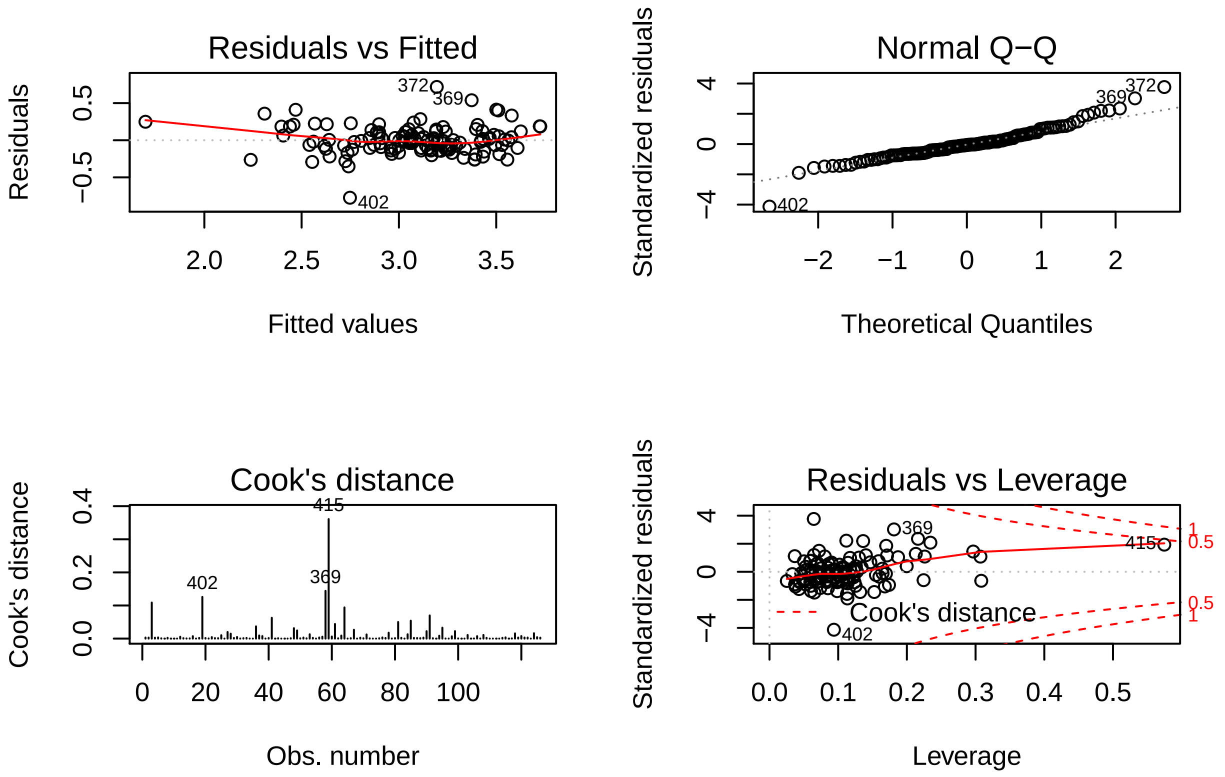


The diagnostics appear to be good, with a weaker variance to the small and large adjusted values, but that is mostly due to a smaller presence of observations in these ranges. A root-square or logarithmic transformation slightly enhance the symmetry of the residuals. There is no presence of strongly aberrant values (the distance of Cook remains small for all members), still, 3 points might be removed from the study.

```r
null <- lm(log(medv)~1, Boston.train)
full <- lm(log(medv)~., Boston.train)
summary(full)
```

```
##
## Call:
## lm(formula = log(medv) ~ ., data = Boston.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77438 -0.11614 -0.01169  0.10243  0.71741
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.2562072  0.4653053   9.147 3.07e-15 ***
## crim        -0.0084300  0.0048144  -1.751 0.082683 .
## zn           0.0023489  0.0011270   2.084 0.039412 *
## indus        0.0027495  0.0055813   0.493 0.623243
## chas         0.0882527  0.0597856   1.476 0.142709
## nox         -0.8916934  0.3228148  -2.762 0.006711 **
## rm           0.0436110  0.0331094   1.317 0.190468
## age          0.0008363  0.0011054   0.757 0.450886
## dis         -0.0589533  0.0166066  -3.550 0.000564 ***
## rad          0.0074325  0.0074683   0.995 0.321780
## tax          -0.0002861  0.0004060  -0.705 0.482488
## ptratio      -0.0314036  0.0116450  -2.697 0.008083 **
## black         0.0005357  0.0002426   2.208 0.029269 *
## lstat        -0.0321420  0.0041060  -7.828 3.03e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1969 on 112 degrees of freedom
## Multiple R-squared:  0.7768, Adjusted R-squared:  0.7509
## F-statistic: 29.98 on 13 and 112 DF,  p-value: < 2.2e-16
```

```r
par(mfrow=c(2,2))
plot(full, which=c(1,2,4,5))
```



2.2 Exhaustive search

Use the `regsubset` function of the `leaps` package to generate all the possible models containing up to 10 predictors. Represent, in relation to the number of predictors used, the evolution of the square error, of the adjusted R^2, the BIC and the C_p for the best 500 models. CThese mesures are accessible by using the `summary` function on the outputs of the `regsubset` function.

There are a total of $2^{10} = 1024$ possible models. We focus on the best 500 of them.

```r
library(leaps)
bss <- summary(regsubsets(log(medv) ~ . , data=Boston.train, nvmax=10, nbest=500, really.big=TRUE))
```

Using this object (by examining it structure - you can use the `str` command), we can recover all the values we need. We recover the best results for each parameter in order to represent them graphicly in separate plots.

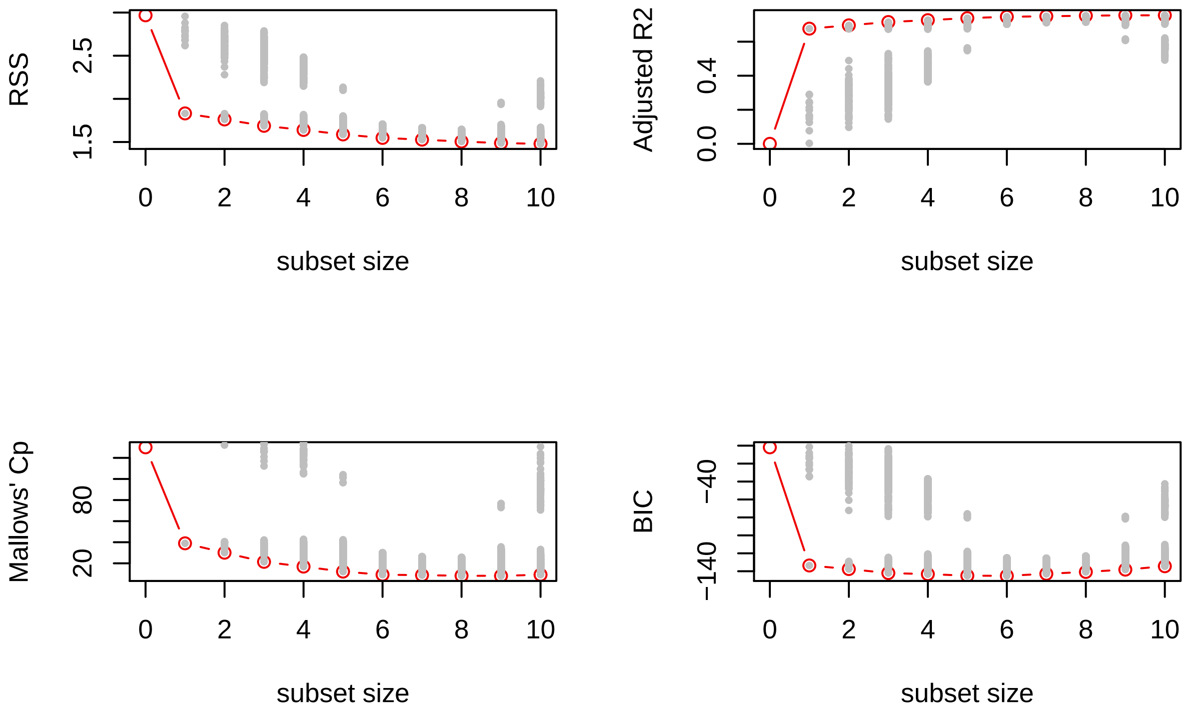
```r
bss.size <- as.numeric(rownames(bss$which))
bss.best.rss   <- tapply(bss$rss, bss.size, min)
bss.best.adjr2 <- tapply(bss$adjr2, bss.size, max)
bss.best.bic   <- tapply(bss$bic  , bss.size, min)
bss.best.Cp    <- tapply(bss$cp   , bss.size, min)
```

We complete the statistics with the performance of the null model which is not ouputed by the `regsubset` function.

```r
n <- nrow(Boston.train)
RSS0 <- sum(resid(null)^2)
bss.best.rss   <- c(RSS0, bss.best.rss)
bss.best.adjr2 <- c(summary(null )$adj.r.squared, bss.best.adjr2)
bss.best.bic   <- c(log(RSS0/n), bss.best.bic)
bss.best.Cp    <- c(n+RSS0/(n*summary(full)$sigma^2), bss.best.Cp)
```

Plot time!

```r
model.size <- 0:10
par(mfrow=c(2,2))
plot(model.size, log(bss.best.rss), type="b",
     xlab="subset size", ylab="RSS", col="red2" )
points(bss.size, log(bss$rss), pch=20, col="gray", cex=0.7)

plot(model.size, bss.best.adjr2, type="b",
     xlab="subset size", ylab="Adjusted R2", col="red2" )
points(bss.size, bss$adjr2, pch=20, col="gray", cex=0.7)

plot(model.size, bss.best.Cp, type="b",
     xlab="subset size", ylab="Mallows' Cp", col="red2" )
points(bss.size, bss$cp, pch=20, col="gray", cex=0.7)

plot(model.size, bss.best.bic, type="b",
     xlab="subset size", ylab="BIC", col="red2" )
points(bss.size, bss$bic, pch=20, col="gray", cex=0.7)
```



2.3 Stepwise Selection

Propose two models, named `step.AIC` and `step.BIC`, that use the `stepwise` regression procedure in the forward/backwards mode. Briefly study the outputs of these models (`anova` and `summary` functions).

```r
scope <- list(lower = terms(log(medv) ~ 1, data=Boston.train),
              upper = terms(log(medv) ~ ., data=Boston.train))
step.AIC <- step(null, scope, direction='both', trace=FALSE)
step.BIC <- step(null, scope, direction='both', k=log(n), trace=FALSE)
```

Both the AIC and BIC models include common variables. The variables selected, however, differ between the two models, due to the existing correlations between these. While we have a slightly worse R^2 with the BIC, its interpretation is easier.

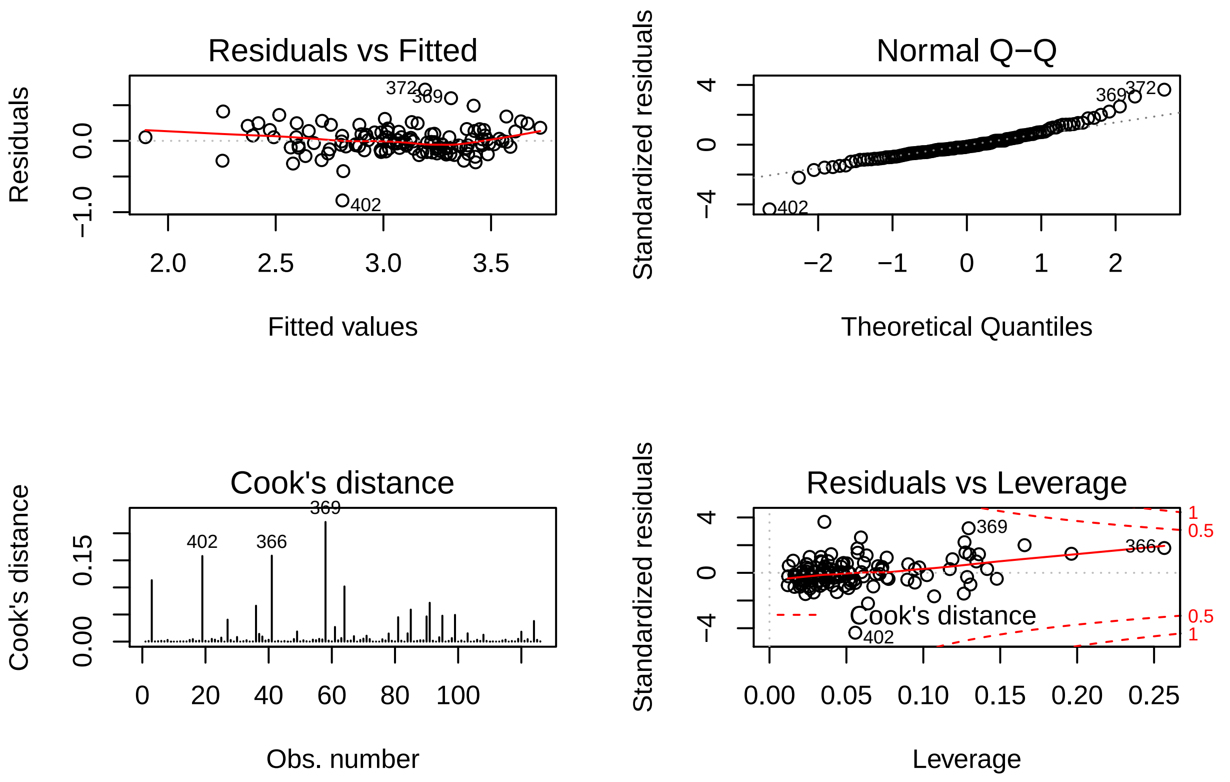
```r
step.AIC
```

```
##
## Call:
## lm(formula = log(medv) ~ lstat + ptratio + black + dis + rm +
##     nox + chas + zn + crim, data = Boston.train)
##
## Coefficients:
## (Intercept)        lstat      ptratio        black          dis
##    4.041942    -0.031709    -0.025934     0.000560    -0.061408
##          rm          nox         chas           zn         crim
##    0.051620    -0.714675     0.097801     0.001979    -0.005861
```

```
step.BIC
```

```
##
## Call:
## lm(formula = log(medv) ~ lstat + ptratio + black + dis + rm +
##     nox, data = Boston.train)
##
## Coefficients:
## (Intercept)        lstat      ptratio        black          dis
##   3.9083116   -0.0334732   -0.0339577    0.0007239   -0.0446258
##          rm          nox
##   0.0711423   -0.5784040
```

```
par(mfrow=c(2,2))
plot(step.BIC, which=c(1,2,4,5))
```



```
summary(step.BIC)
```

```
##
## Call:
## lm(formula = log(medv) ~ lstat + ptratio + black + dis + rm +
##     nox, data = Boston.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.83559 -0.10701 -0.02696  0.09228  0.71853
##
```

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.9083116  0.4068919   9.605  < 2e-16 ***
## lstat       -0.0334732  0.0035727  -9.369 5.85e-16 ***
## ptratio     -0.0339577  0.0093857  -3.618 0.000437 ***
## black        0.0007239  0.0002249   3.219 0.001660 **
## dis         -0.0446258  0.0131588  -3.391 0.000945 ***
## rm           0.0711423  0.0316522   2.248 0.026442 *
## nox         -0.5784040  0.2626378  -2.202 0.029572 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1988 on 119 degrees of freedom
## Multiple R-squared:  0.7584, Adjusted R-squared:  0.7462
## F-statistic: 62.26 on 6 and 119 DF,  p-value: < 2.2e-16
```

Third part: penalisation methods

In this part we are interested in the regularised methods `ridge` and `lasso` in order to constrain the variance of our estimator and control the variance of our estimator and -eventualy- improve our prediction error. To generate these models we shall use the `glmnet` package. You will mostly need the `glmnet`, `predict`, `cv.glmnet` and `plot` functions of this package. Type`help(glmnet)`, `help(plot.glmnet)`, `help(cv.glmnet)` and `help(predict.glmnet)` to get help on these functions.

3.1 Ridge Regression

Generate the Ridge regression model on the 10 predictors ensemble. Trace the obtained regularisation path and comment on it. Select a λ through 10-fold cross-validation with the minimum and a "1 standard error" rules (the most penalized model with a 1 std distance from the model with the least error). We shall name the corresponding models `ridge.min` and `ridge.1se`

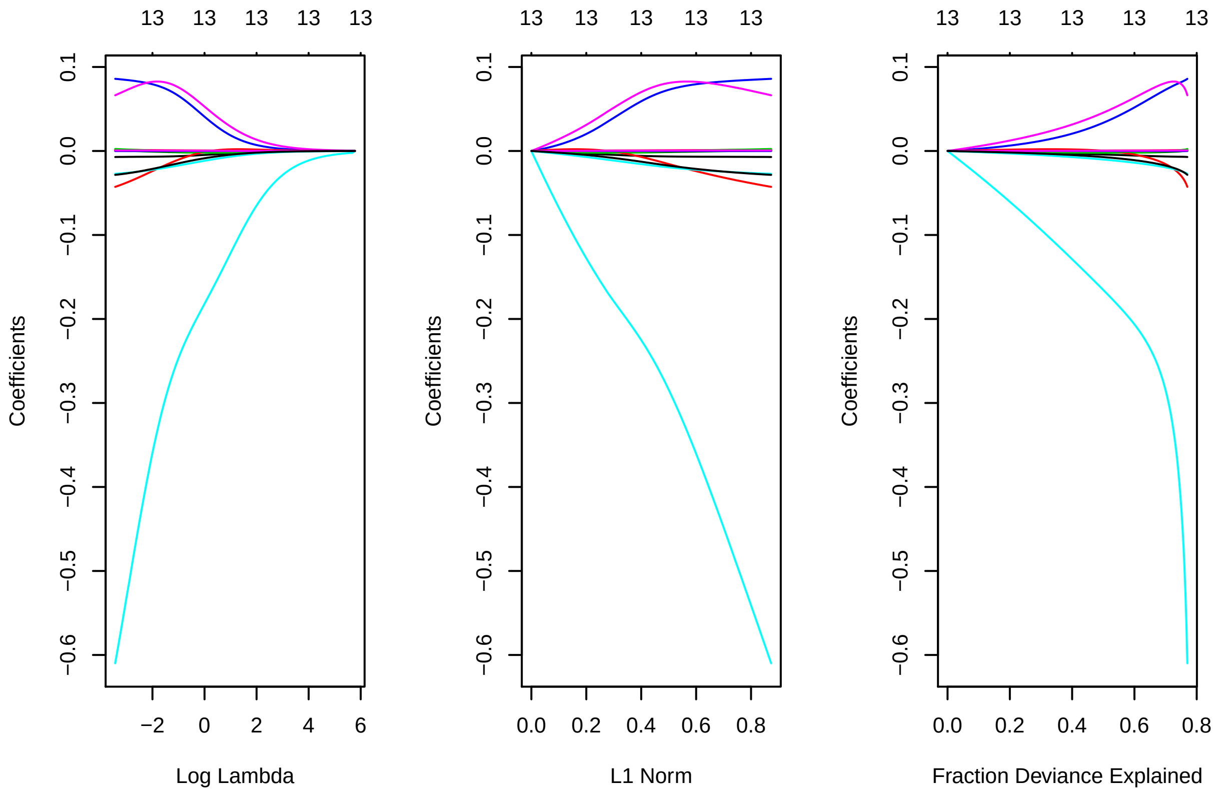
Take care with the input format for the `glmnet` function.

Take care with the input format for the `glmnet` function.

```r
library(glmnet)
x <- as.matrix(Boston.train[, -14])
y <- log(Boston.train$medv)
ridge <- glmnet(x,y,alpha=0)
```

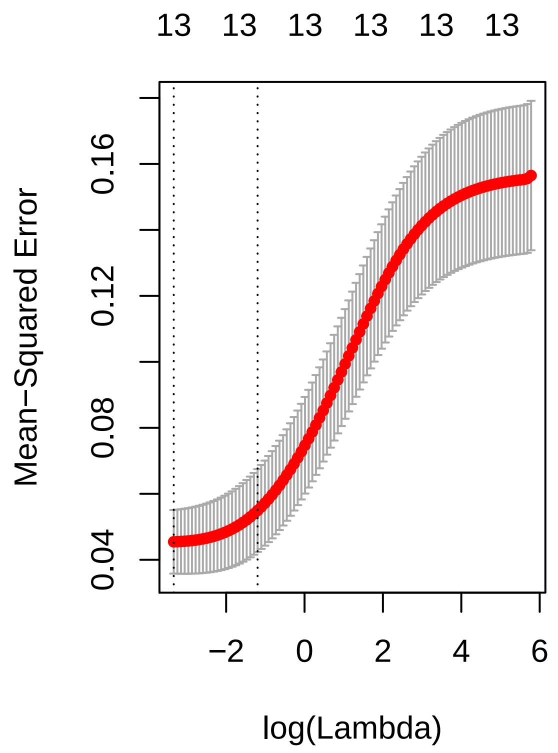
We can represent the regularisation path in function of different mesurements:

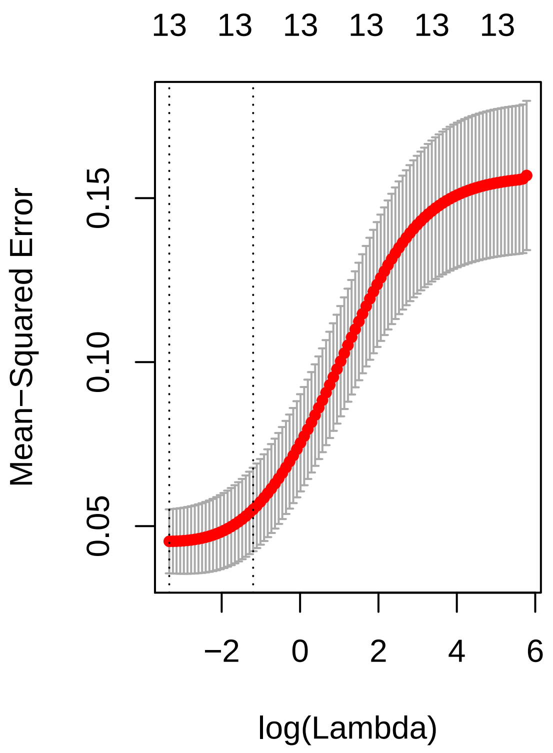
```r
par(mfrow=c(1,3))
plot(ridge, xvar="lambda")
plot(ridge, xvar="norm")
plot(ridge, xvar="dev")
```



The results are interchangable for both methods of cross-validation.

```
ridge.10cv <- cv.glmnet(x,y,nfolds=10, alpha=0, grouped=FALSE)
ridge.loo  <- cv.glmnet(x,y,nfolds=n , alpha=0, grouped=FALSE)
par(mfrow=c(1,2))
plot(ridge.10cv)
plot(ridge.loo)
```





We can access the generated models through the `predict` function. Take care as the new value of the predictors used for the prediction must be formated as a matrix.

```
x0 <- as.matrix(Boston.train[1:5, -14]) # A new observation as an example
predict(ridge, newx=x0, s=ridge.10cv$lambda.min)
```

```
##             1
## 237 3.323187
## 101 3.200516
## 149 2.546460
## 133 3.076295
## 472 3.078646
```

```
predict(ridge, newx=x0, s=ridge.10cv$lambda.1se)
```

```
##             1
## 237 3.273972
## 101 3.161665
## 149 2.716489
## 133 3.039378
## 472 3.011169
```

3.2 Lasso Regression

Generate the Ridge regression model on the 10 predictors ensemble. Trace the obtained regularisation path and comment on it. Select a λ through 10-fold cross-validation with the minimum and a "1 standard error" rules (the most penalized model with a 1 std distance from the model with the least error).

Also trace the BIC et mBIC criteria, whose analytic expression is reminded to be:

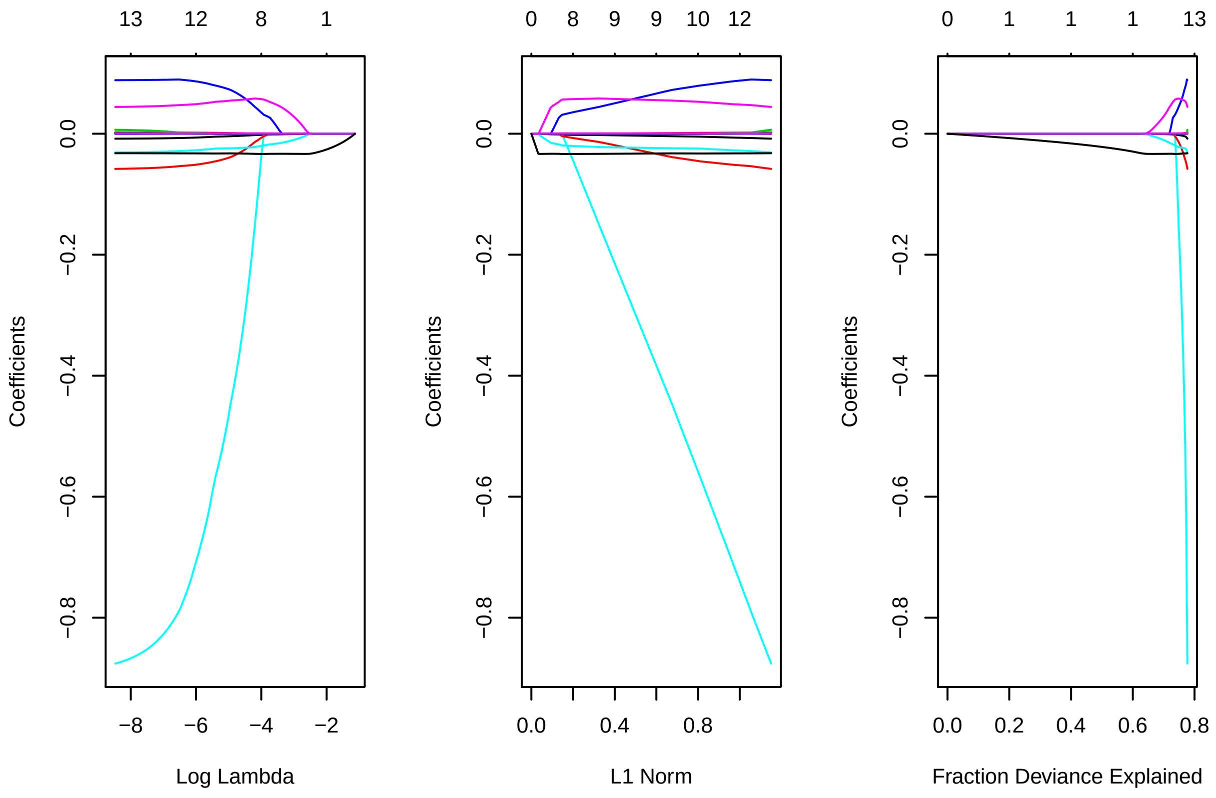
$$\mathrm{BIC} = n\log(\mathrm{err}_{\mathcal{D}}) + log(n)\mathrm{df}(\lambda)$$

$$\mathrm{mBIC} = n\log(\mathrm{err}_{\mathcal{D}}) + (\log(n) + 2log(p))\mathrm{df}(\lambda)$$

We name `lasso.min`, `lasso.1se`, `lasso.BIC` and `lasso.mBIC` the corresponding models.

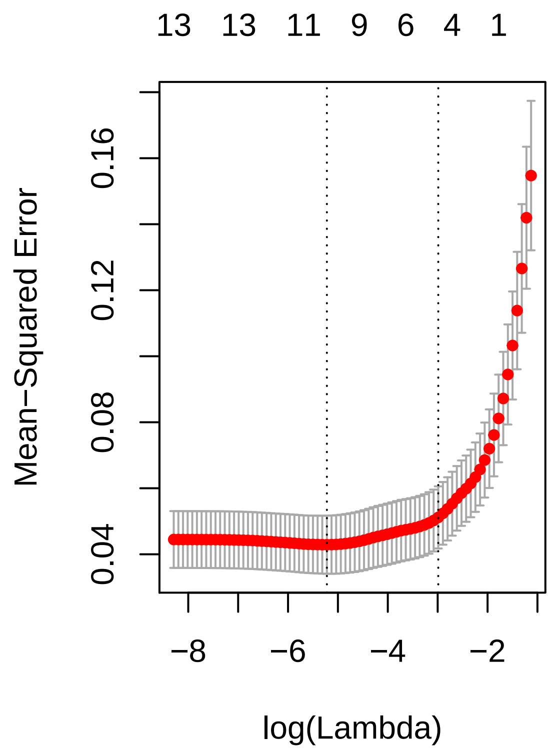
We just need to use the previous code and modulate the `alpha` parameter. Note the consecutivelly selected variables on the path towards the solution.

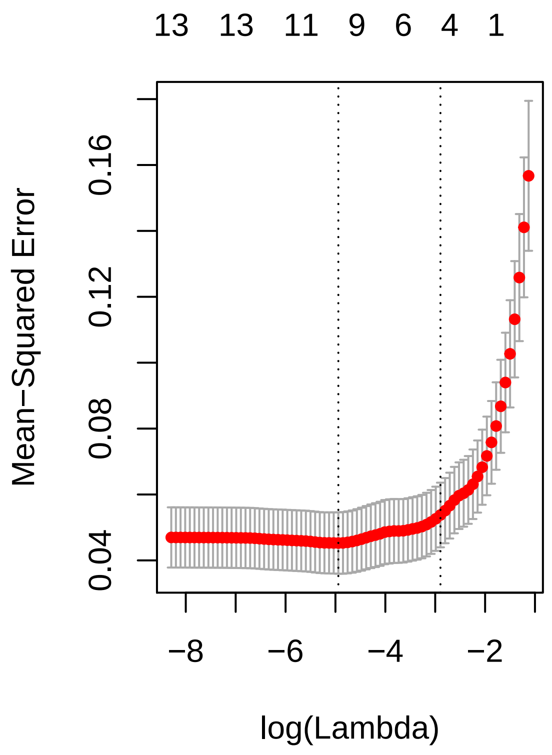
```
lasso <- glmnet(x,y)
par(mfrow=c(1,3))
plot(lasso, xvar="lambda")
plot(lasso, xvar="norm")
plot(lasso, xvar="dev")
```



Again the results remain close with both methods of cross-validation.

```
lasso.10cv <- cv.glmnet(x,y,nfolds=10, grouped=FALSE)
lasso.loo  <- cv.glmnet(x,y,nfolds=n , grouped=FALSE)
par(mfrow=c(1,2))
plot(lasso.10cv)
plot(lasso.loo)
```





We will therefore study the models that minimise the cross validation error and the most parcimonious model within 1 std of the best model.

```r
predict(lasso, x0, s=lasso.10cv$lambda.min)
```

```
##            1
## 237 3.313431
## 101 3.218219
## 149 2.512261
## 133 3.098502
## 472 3.061070
```

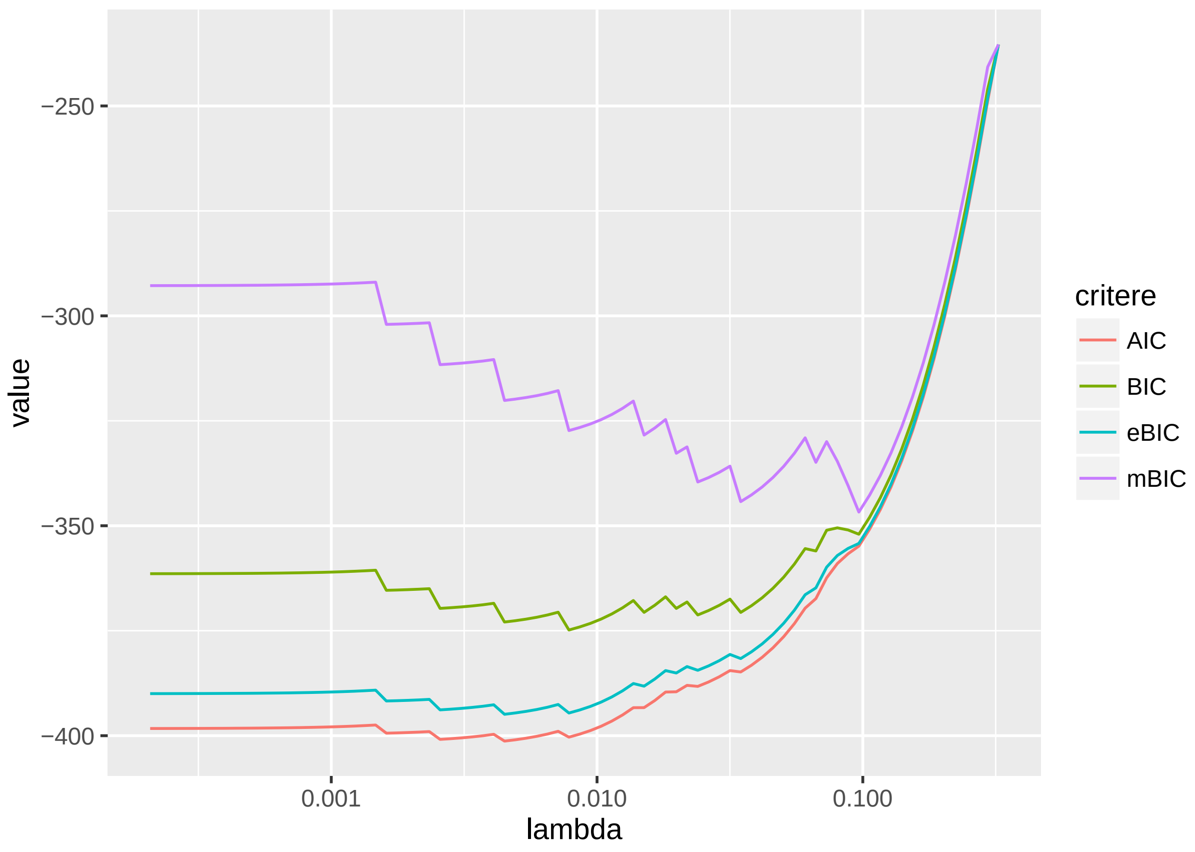
```r
predict(lasso, x0, s=lasso.10cv$lambda.1se)
```

```
##            1
## 237 3.207964
## 101 3.181633
## 149 2.559712
## 133 3.109201
## 472 3.058697
```

Here are the standard parameters for a lasso regression.

```r
n <- nrow(Boston.train)
p <- ncol(Boston.train) - 1 + 1
AIC  <- n*log(colMeans((y - predict(lasso, x))^2)) + 2 * lasso$df
BIC  <- n*log(colMeans((y - predict(lasso, x))^2)) + log(n) * lasso$df
eBIC <- n*log(colMeans((y - predict(lasso, x))^2)) + log(p) * lasso$df
mBIC <- n*log(colMeans((y - predict(lasso, x))^2)) + (log(n) + 2 *log(p)) * lasso$df
```

```
library(ggplot2)
d <- data.frame(lambda  = rep(lasso$lambda, 4),
                value   = c(AIC, BIC, eBIC, mBIC),
                critere = factor(rep(c("AIC","BIC","eBIC","mBIC"), each=length(lasso$lambda))))
ggplot(d, aes(x=lambda,y=value,colour=critere,group=critere)) + geom_line() + scale_x_log10()
```



We name `lasso.min`, `lasso.1se`, `lasso.BIC` and `lasso.mBIC` the corresponding models.

```
lambda.min.BIC  <- lasso$lambda[which.min(BIC)]
lambda.min.mBIC <- lasso$lambda[which.min(mBIC)]
predict(lasso, x0, s=lambda.min.BIC)
```

```
##            1
## 237 3.305179
## 101 3.210178
## 149 2.526363
## 133 3.093914
## 472 3.055461
```

```
predict(lasso, x0, s=lambda.min.mBIC)
```

```
##            1
## 237 3.169984
## 101 3.173758
## 149 2.579449
## 133 3.120301
## 472 3.065273
```

Forth part: Evaluating the quality of the models obtained

For each of your models, i.e. :

- The model corresponding to a constant (`null`),
- The model with all the predictors (`full`),
- The models obtained by using the stepwise methodology (`step.AIC` et `step.BIC`),
- The models obtained by using the ridge methodology (`ridge.min` et `ridge.1se`),
- The models obtained by using the lasso methodology (`lasso.min`, `lasso.1se`, `lasso.BIC` et `lasso.mBIC`),

Estimate its precition errors with the help of the test dataset. You can use the `predict` functions associated to the different objects you are manipulating.

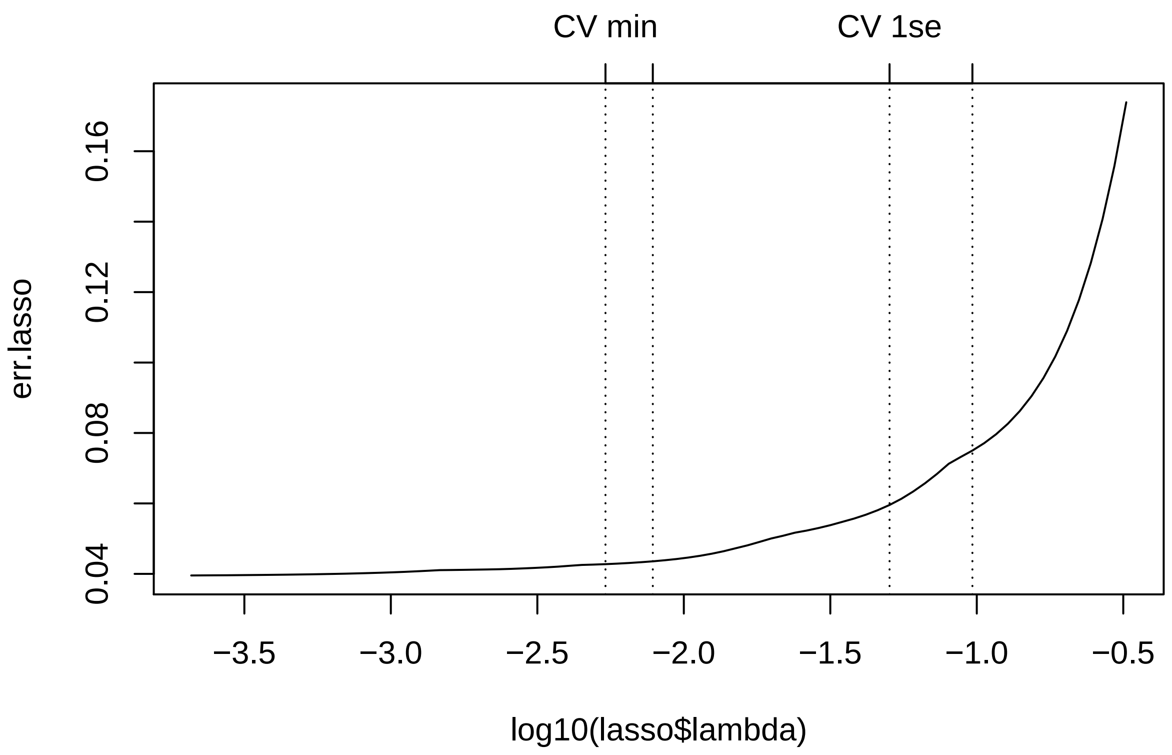
```r
y.test <- log(Boston.test$medv)
x.test <- as.matrix(Boston.test[, -14])

err.null <- mean((y.test - predict(null, Boston.test))^2)
err.full <- mean((y.test - predict(full, Boston.test))^2)
err.sAIC <- mean((y.test - predict(step.AIC, Boston.test))^2)
err.sBIC <- mean((y.test - predict(step.BIC, Boston.test))^2)
err.ridge.min  <- mean((y.test - predict(ridge, newx=x.test, s=ridge.10cv$lambda.min))^2)
err.ridge.1se  <- mean((y.test - predict(ridge, newx=x.test, s=ridge.10cv$lambda.min))^2)
err.lasso.min  <- mean((y.test - predict(lasso, newx=x.test, s=lasso.10cv$lambda.min))^2)
err.lasso.1se  <- mean((y.test - predict(lasso, newx=x.test, s=lasso.10cv$lambda.min))^2)
err.lasso.BIC  <- mean((y.test - predict(lasso, newx=x.test, s=lambda.min.BIC))^2)
err.lasso.mBIC <- mean((y.test - predict(lasso, newx=x.test, s=lambda.min.mBIC))^2)

res <- data.frame(modele = c("null", "full", "step.AIC", "step.BIC", "ridge.CVmin",
                             "ridge.CV1se", "lasso.CVmin", "lasso.CV1se", "lasso.BIC", "lasso.mBIC"),
                  erreur = c(err.null, err.full, err.sAIC, err.sBIC, err.ridge.min, err.ridge.1se,
                             err.lasso.min, err.lasso.1se, err.lasso.BIC, err.lasso.mBIC))
print(res)
```

```
##         modele     erreur
## 1         null 0.17390082
## 2         full 0.03932112
## 3     step.AIC 0.04183688
## 4     step.BIC 0.04690176
## 5  ridge.CVmin 0.04119122
## 6  ridge.CV1se 0.04119122
## 7  lasso.CVmin 0.04276014
## 8  lasso.CV1se 0.04276014
## 9    lasso.BIC 0.04356138
## 10  lasso.mBIC 0.07501286
```

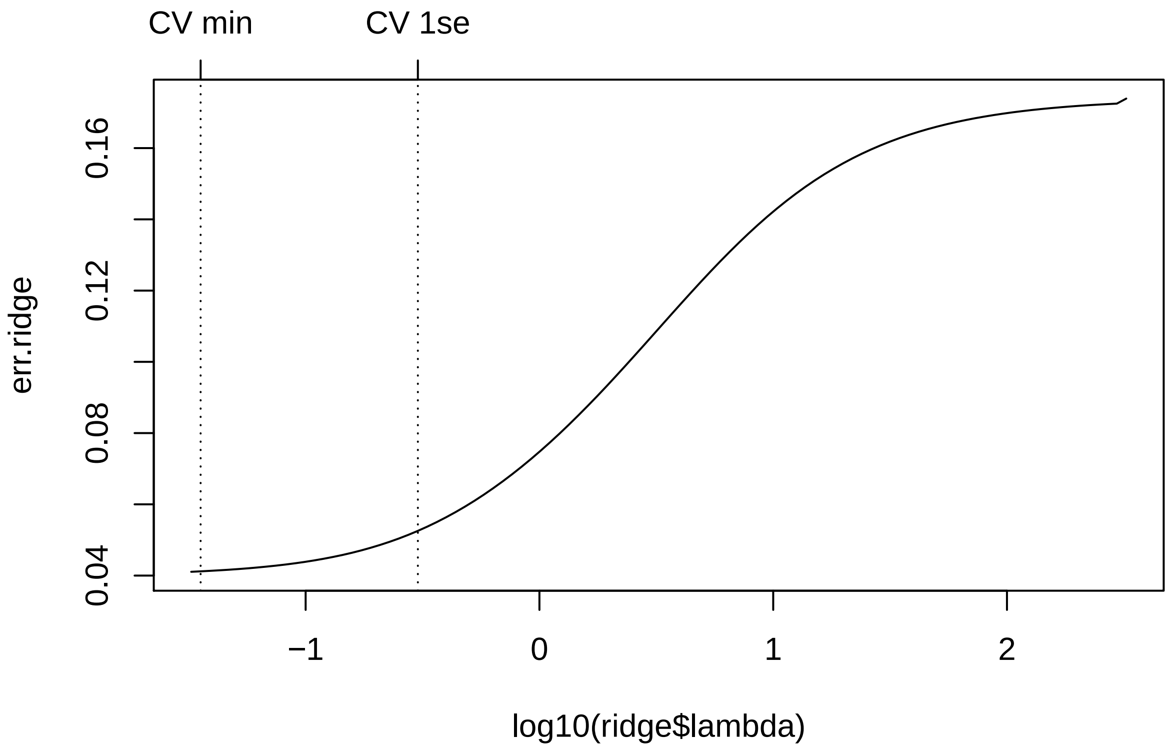
For the ridge and lasso regressions, plot the evolution of the prediction error calculated over the test ensemble versus the values of the regularization parameters (use a logarithmic scale). Highlight the λ values corresponding to the cross-validation and penalization criteria.

```r
err.lasso  <- colMeans((y.test - predict(lasso, newx=x.test))^2)
err.ridge  <- colMeans((y.test - predict(ridge, newx=x.test))^2)
```

```r
plot(log10(lasso$lambda), err.lasso,  type="l")
axis(3, at=c(log10(lasso.10cv$lambda.min), log10(lasso.10cv$lambda.1se),
            log10(lambda.min.BIC), log10(lambda.min.mBIC)),
        labels = c("CV min", "CV 1se", "BIC", "mBIC"))
abline(v=c(log10(lasso.10cv$lambda.min), log10(lasso.10cv$lambda.1se),
          log10(lambda.min.BIC), log10(lambda.min.mBIC)), lty=3)
```



```r
plot(log10(ridge$lambda), err.ridge, type="l")
axis(3, at=c(log10(ridge.10cv$lambda.min), log10(ridge.10cv$lambda.1se)),
        labels = c("CV min", "CV 1se"))
abline(v=c(log10(ridge.10cv$lambda.min), log10(ridge.10cv$lambda.1se)), lty=3)
```


CV min
CV 1se
err.ridge
0.04
0.08
0.12
0.16
−1
0
1
2
log10(ridge$lambda)