Linear Regression & Feature Selection

Francisco Madrid, Toni Pardo

2023-03-30

For this lab we will use the Prostate Cancer Data from the ElemStatLearn package. (This package is deprecated but you can still find a archived version in https://cran.r-project.org/src/contrib/Archive/ElemStatLearn/) This dataset allows to examine the correlation between the level of prostate-specific antigen and a number clinical measurements in men who were about to receive a radical prostatectomy.

A data frame with 97 observations on the following 10 variables.

lcavol: log cancer volume lweight: log prostate weight age: in years lbph: log of the amount of benign prostatic hyperplasia svi: seminal vesicle invasion lcp: log of capsular penetration gleason: a numeric vector from pathohistology examinations pgg45: percent of Gleason score 4 or 5 lpsa: response. This is the quantity that we aim to predict. train: a logical vector

The last column indicates which 67 observations were used as the "training set" and which 30 as the test set. This dataset was originally published by Stamey et al.[1]

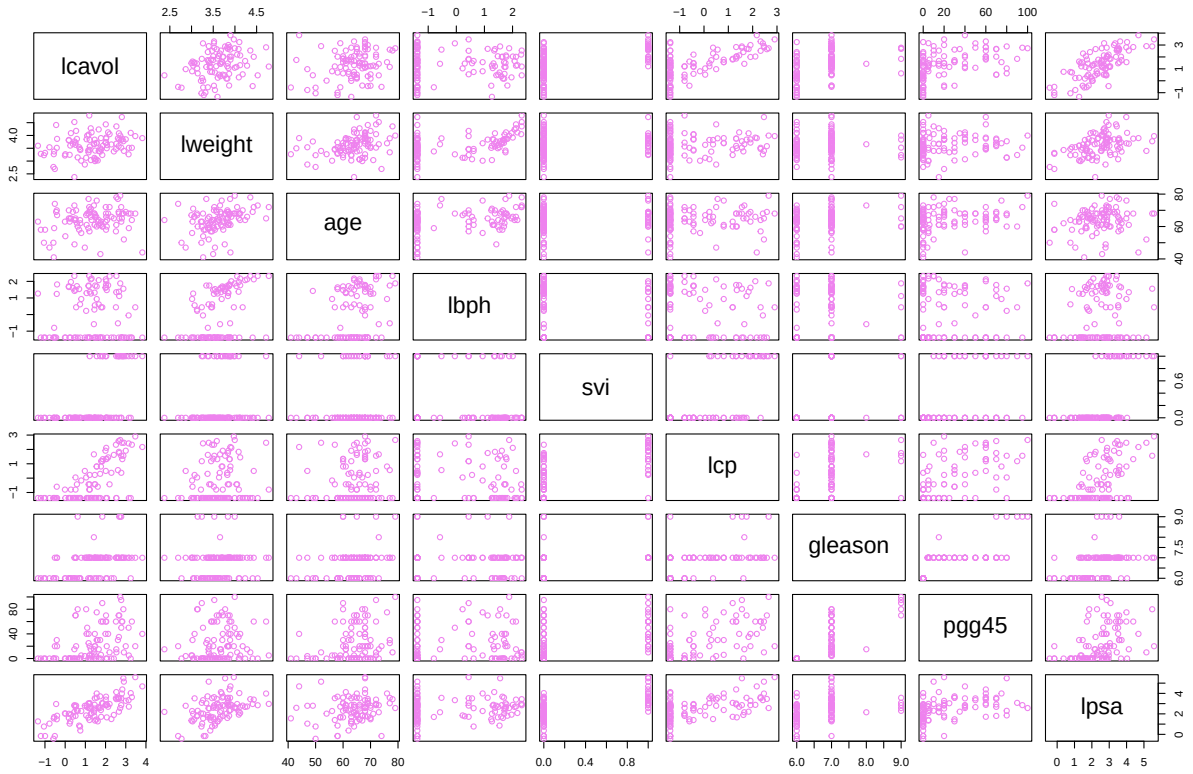
Introduction

First, install the package ElemStatLearn in order to load the prostate data with:

```
# install.packages("ElemStatLearn")

library(ElemStatLearn)
library(pracma)
library(MASS)
data(prostate)

## Do a first inspection of the data structure with:
pairs( prostate[,1:9], col="violet" )
```

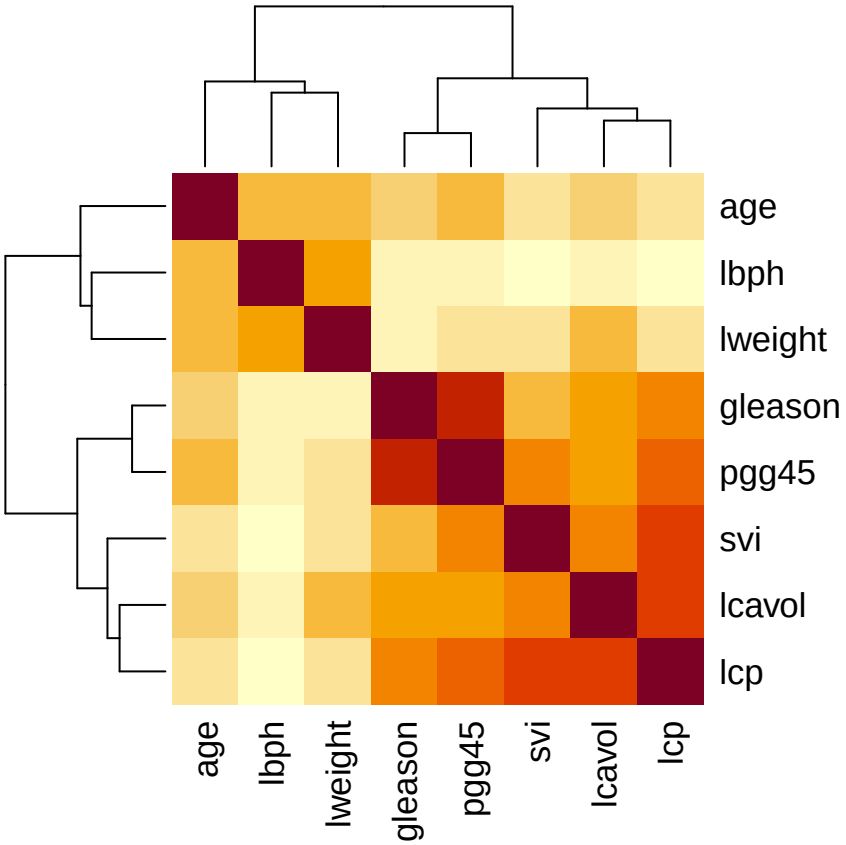


A way to inspect the relationship among the regressors is to compute their covariance matrix. You can do that using cor. Find the highest correlated regressors. In order to produce a heatmap of the correlation matrix you may use heatmap with symm=TRUE.

```
# Correlations
cor(prostate[,1:8])
```

```
##             lcavol   lweight       age         lbph          svi           lcp
## lcavol  1.0000000 0.2805214 0.2249999  0.027349703  0.53884500  0.675310484
## lweight 0.2805214 1.0000000 0.3479691  0.442264395  0.15538491  0.164537146
## age     0.2249999 0.3479691 1.0000000  0.350185896  0.11765804  0.127667752
## lbph    0.0273497 0.4422644 0.3501859  1.000000000 -0.08584324 -0.006999431
## svi     0.5388450 0.1553849 0.1176580 -0.085843238  1.00000000  0.673111185
## lcp     0.6753105 0.1645371 0.1276678 -0.006999431  0.67311118  1.000000000
## gleason 0.4324171 0.0568821 0.2688916  0.077820447  0.32041222  0.514830063
## pgg45   0.4336522 0.1073538 0.2761124  0.078460018  0.45764762  0.631528246
##             gleason      pgg45
## lcavol  0.43241706 0.43365225
## lweight 0.05688210 0.10735379
## age     0.26889160 0.27611245
## lbph    0.07782045 0.07846002
## svi     0.32041222 0.45764762
## lcp     0.51483006 0.63152825
## gleason 1.00000000 0.75190451
## pgg45   0.75190451 1.00000000
```

```
# Heatmap
heatmap(cor( prostate[,1:8] ), symm=TRUE)
```



Now scale columns 1 to 8 and bind the 9 column to obtain a data.frame prost.std. Follow by doing the partition in train and test using the column 10 of the original data: `data.train` and `data.test`.

```
prost <- prostate

prost.std <- data.frame(cbind(scale(prost[,1:8]),prost$lpsa))
names(prost.std)[9] <- 'lpsa'
data.train <- prost.std[prost$train,]
data.test <- prost.std[!prost$train,]
y.train<-data.train$lpsa
y.test <- data.test$lpsa
n.train <- nrow(data.train)
```

Ordinary Least Squares

Once we have the data, let us start exploring this data with ordinary least squares (OLS). For this you can use the `lm` command from basic R.

You may inspect the model with summary in order to learn about the significance of the model coefficients. What are the most significant coefficients? To do that use summary on the OLS model.

Now with the fitted model in the train data you can predict the test data using `predict`. You can calculate the squared residuals for the test data. You may store them in a variable in order to do a posterior comparison with other methods.

In order to have a better feeling how this fitting looks like, you can plot the predicted test data vs the real test response values. If you decide to add the train data as well, please use a different symbol. Add also a line with slope one and intercept zero. Now fit a straight line between the test predictions and the actual

data also using lm. Plot with a different color the actual best fit and compare with the ideal line you have plotted before. To do this plot, inspect the results of the linear model to find the fitted values. Compare the slope and the intercept to the ideal line. In order to understand better the issues with multivariate ols, let us see how the significance of a feature depends on another feature. First fir lpsa against svi and inspect the model coefficients. Now fit lpsa against svi and lcavol. How the significance of svi coefficient has changed?

```
m.ols <- lm(lpsa ~ . , data=data.train)
summary(m.ols)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = data.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.46493    0.08931  27.598  < 2e-16 ***
## lcavol       0.67953    0.12663   5.366 1.47e-06 ***
## lweight      0.26305    0.09563   2.751  0.00792 **
## age         -0.14146    0.10134  -1.396  0.16806
## lbph         0.21015    0.10222   2.056  0.04431 *
## svi          0.30520    0.12360   2.469  0.01651 *
## lcp         -0.28849    0.15453  -1.867  0.06697 .
## gleason     -0.02131    0.14525  -0.147  0.88389
## pgg45        0.26696    0.15361   1.738  0.08755 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12
```

```
y.pred.ols <- predict(m.ols,data.test)
y.predtrain.ols<-predict(m.ols,data.train)
RS.ols <- (y.pred.ols - y.test)^2
summary((y.pred.ols - y.test)^2)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000209 0.044033 0.116450 0.521274 0.444987 4.097498
```

```
mean((y.pred.ols - y.test)^2)/var(y.test)
```

```
## [1] 0.4800661
```

```
plot(y.test,y.pred.ols)
points(y.train, y.predtrain.ols,pch=8)
lines(c(0,6),c(0,6), type="line", col="red")
```

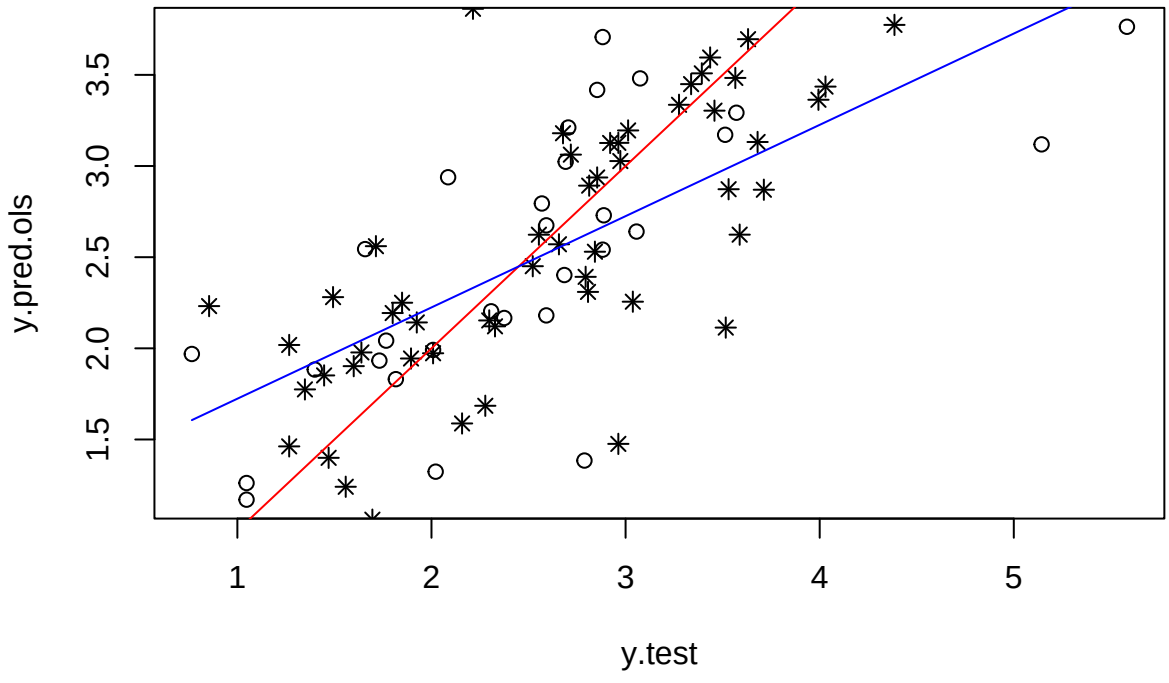
```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): plot type 'line' will be
## truncated to first character
```

```
fiteval <- data.frame(cbind(y.test,y.pred.ols))
names(fiteval)[1]<-'real'
names(fiteval)[2]<-'pred'
```

```
m.olseval<-lm(pred~real, data=fiteval)
summary(m.olseval)
```

```
##
## Call:
## lm(formula = pred ~ real, data = fiteval)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.23459 -0.25238 -0.08894  0.34346  1.04139
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.22293    0.25618   4.774 5.15e-05 ***
## real         0.50073    0.09365   5.347 1.07e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5255 on 28 degrees of freedom
## Multiple R-squared:  0.5052, Adjusted R-squared:  0.4876
## F-statistic: 28.59 on 1 and 28 DF,  p-value: 1.075e-05
```

```
lines(y.test, m.olseval$fitted.values, col="blue")
```




```
# the importance of a feature depends on other features. Compare the following. Explain (visualize).
summary(lm(lpsa~svi, data = data.train))
```

```
##
## Call:
## lm(formula = lpsa ~ svi, data = data.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.52458 -0.56828  0.01705  0.71560  1.93601
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.4405     0.1235  19.758  < 2e-16 ***
## svi           0.6630     0.1227   5.406 9.88e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.011 on 65 degrees of freedom
## Multiple R-squared:  0.3101, Adjusted R-squared:  0.2995
## F-statistic: 29.22 on 1 and 65 DF,  p-value: 9.879e-07
```

```
summary(lm(lpsa~svi+lcavol, data = data.train))
```

```
##
## Call:
## lm(formula = lpsa ~ svi + lcavol, data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6200 -0.5176  0.1609  0.5256  1.6946
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47040    0.09947  24.834  < 2e-16 ***
## svi          0.22431    0.12252   1.831   0.0718 .
## lcavol       0.71195    0.11789   6.039  8.7e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8131 on 64 degrees of freedom
## Multiple R-squared:  0.5605, Adjusted R-squared:  0.5468
## F-statistic: 40.82 on 2 and 64 DF,  p-value: 3.747e-12
```

Best subset selection: Exhaustive Search and Sequential Searches

In subset selection we aim to select the best subset of p regressors among a total of k regressors using n training samples.

For best subset selection we will be using the leaps library. Leaps implements several criteria to decide on the best model using only the training data. We will not go deep on the theory, but mostly they are a combination of the Residuals Sum of Squares (RSS) and some penalty term that depends on the number of coefficients and the number of training data. One of them is the *Mallows' Cp* statistics [2].

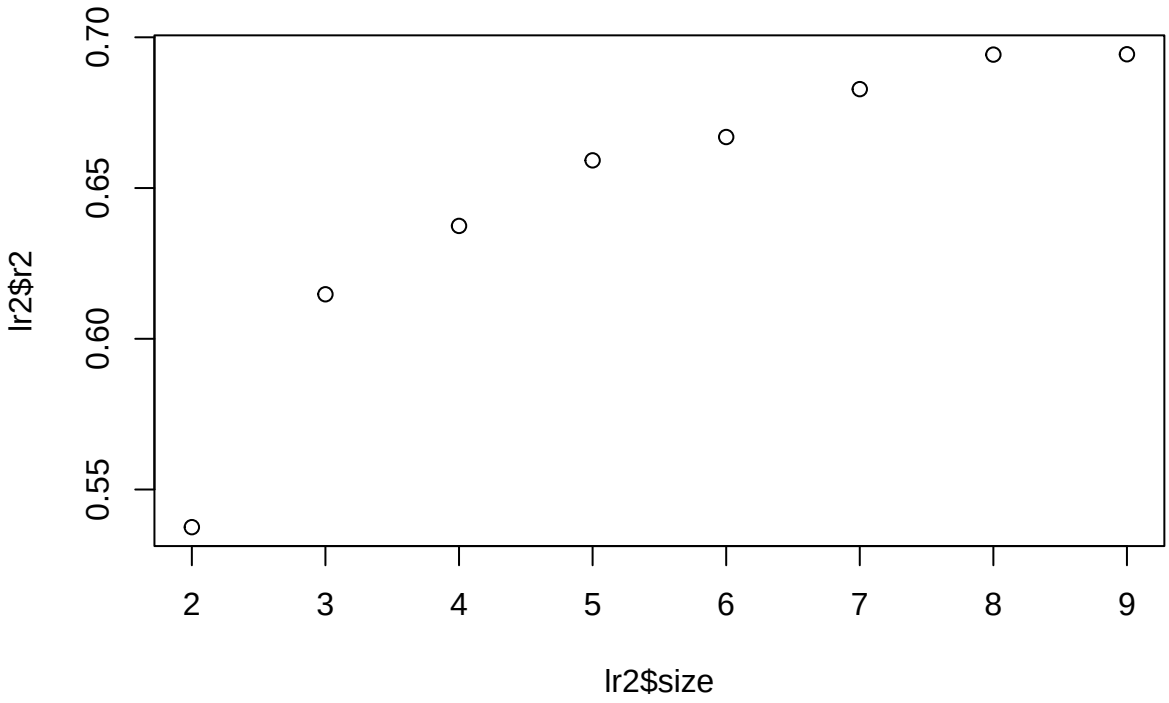
The command leaps does an exhaustive search for the best subsets using a branch and bound algorithm. The branch and bound algorithm is able to skip some of the cases because RSSp decreases monolithically with p. (Look for the branch and bound algorithm for more details).[3]

```
library(leaps)
lr2 <- leaps(data.train[,1:8],data.train[,9],method='r2',nbest=1)
lcp <- leaps(data.train[,1:8],data.train[,9],method='Cp', nbest=1)
```

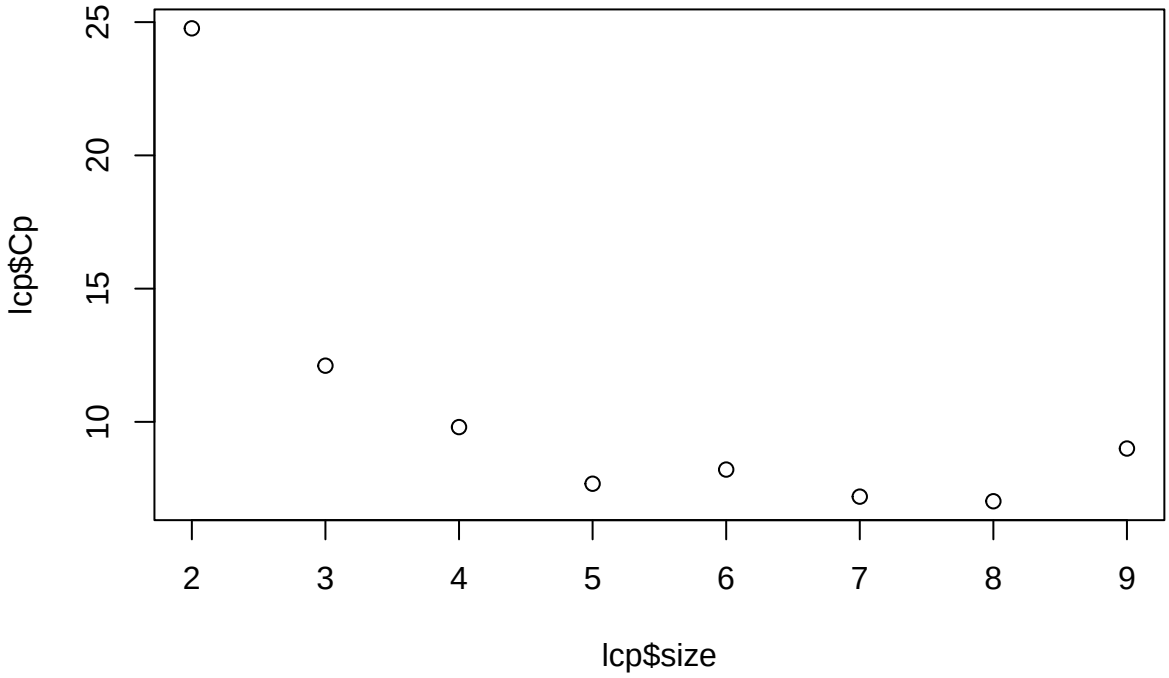
the argument `nbest = 1` ensures that it only returns the best model for each number of variables. Please interpret the information contained in lr2 and look how the variables are added to the model.‘

Plot the adjusted R2 against the size of the subset. Look at the contents of lr2. Do the same with the Mallows' Cp. Try to think what is the minimum subset that may give the best results according these criteria.

```
plot(lr2$size,lr2$r2)
```




```
plot(lcp$size,lcp$Cp)
```



One way to select the best model is to find the absolute minimum of Cp, though due to the parsimony criteria a model with less coefficients is probably better. Select the one with the minimum Cp. To do that you may use which.min to select the best row. Can you see which variables have been selected?

```
# Select best model according to Cp
bestfeat <- lcp$which[which.min(lcp$Cp),]
```

Fit a linear model only with the selected variables and do the same diagnostics as we did in the previous section with the OLS model. In particular, find the prediction for the test data and the squared residuals.

```
# Train and test the model on the best subset
m.bestsubset <- lm(lpsa ~ .,data=data.train[,bestfeat])
summary(m.bestsubset)
```

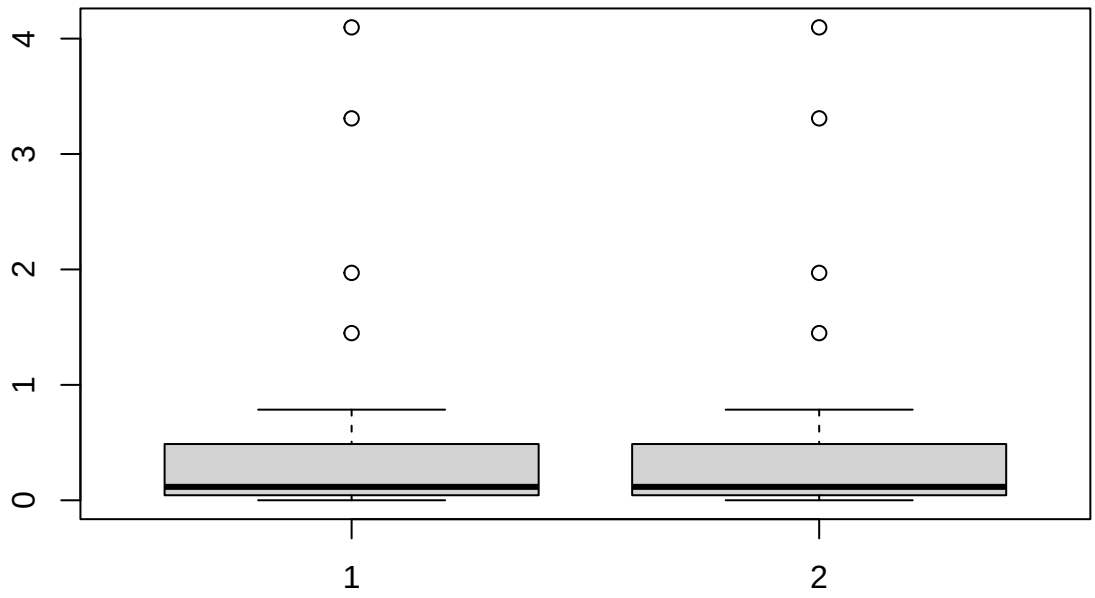
```
##
## Call:
## lm(formula = lpsa ~ ., data = data.train[, bestfeat])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.65425 -0.34471 -0.05615  0.44380  1.48952
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.46687    0.08760  28.161  < 2e-16 ***
## lcavol       0.67645    0.12384   5.462 9.88e-07 ***
## lweight      0.26528    0.09363   2.833   0.0063 **
## age         -0.14503    0.09757  -1.486   0.1425
```

```
## lbph         0.20953    0.10128   2.069   0.0430 *
## svi          0.30709    0.12190   2.519   0.0145 *
## lcp         -0.28722    0.15300  -1.877   0.0654 .
## pgg45        0.25228    0.11562   2.182   0.0331 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7064 on 59 degrees of freedom
## Multiple R-squared:  0.6943, Adjusted R-squared:  0.658
## F-statistic: 19.14 on 7 and 59 DF,  p-value: 4.496e-13
```

```
y.pred.bestsubset <- predict(m.bestsubset,data.test[,bestfeat])
RS.leaps.cp<-(y.pred.ols - y.test)^2
summary((y.pred.bestsubset - y.test)^2)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000575 0.038005 0.122430 0.516513 0.449268 4.002000
```

```
boxplot(RS.ols,RS.leaps.cp)
```



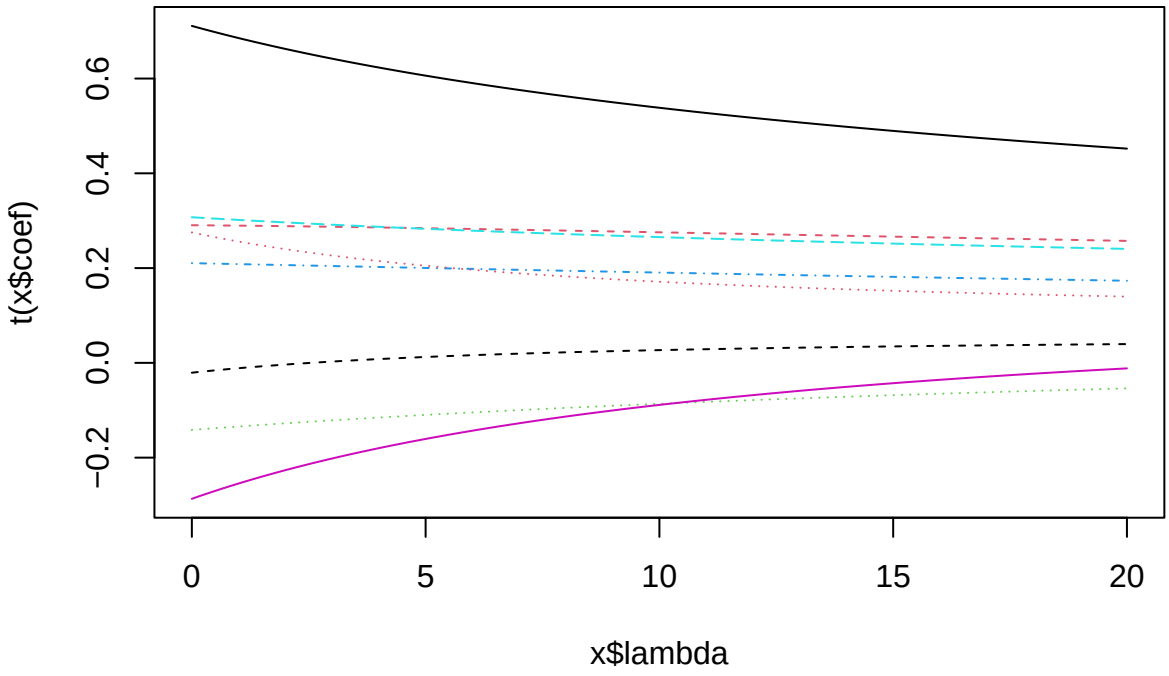
Interpret the result of the algorithm. Explain with your own words which combinations of variables is the algorithm exploring at each step. Compare the obtained result with the previous result by SFS or SBS.

Ridge Regression

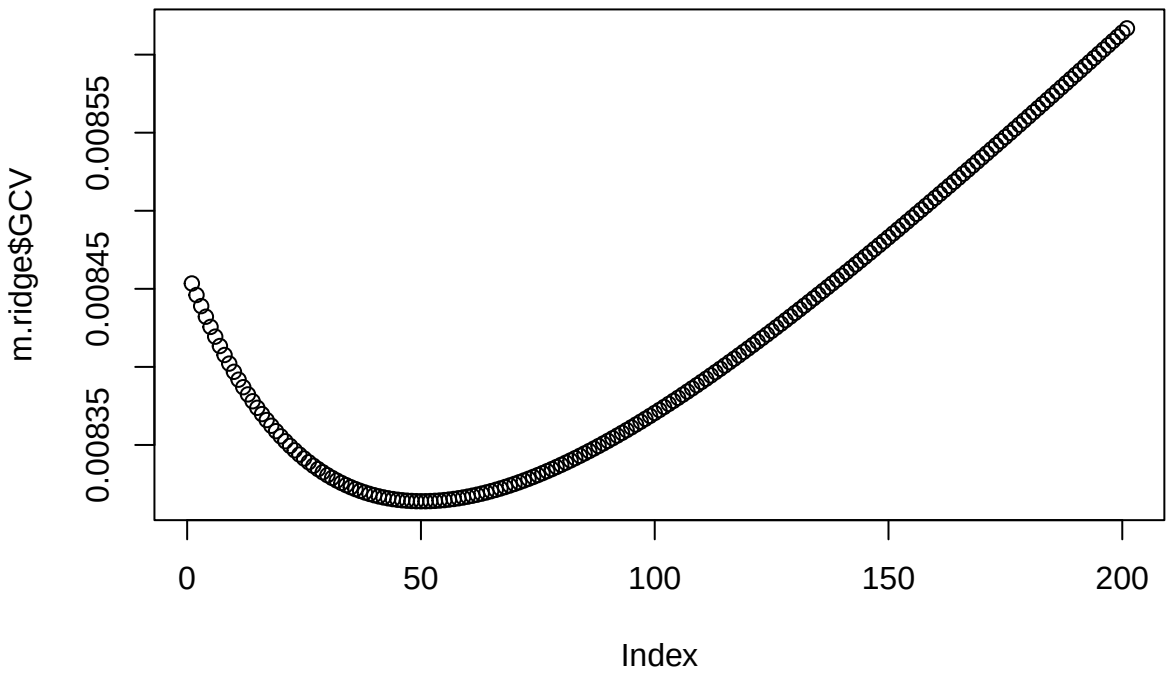
Now, let us consider regularization techniques starting with Ridge Regression. This technique is implemented in the MASS package with the function `lm.ridge`. Look at the example and fit a model while scanning

the lambda from 0 to 20 in steps of 0.1. This function computes the Generalized Cross-Validation (GCV). GCV was proposed by Golub et al. [4] as an approximation of the LOO CV estimator with the advantage of avoiding the use of test data. Now plot m.ridge$GCV where m.ridge is the regression model as a function of lambda. Find the regularization parameter with the lowest GCV and the corresponding regression coefficients. In this package, the function predict is not implemented for ridge regression models so I suggest you to implement the prediction yourselves.

```
library(MASS)
m.ridge <- lm.ridge(lpsa ~ .,data=data.train, lambda = seq(0,20,0.1))
plot(m.ridge)
```




```
# select parameter by minimum GCV
plot(m.ridge$GCV)
```




```
# Predict is not implemented so we need to do it ourselves
y.pred.ridge = scale(data.test[,1:8],center = F, scale = m.ridge$scales)%*% m.ridge$coef[,which.min(m.r
summary((y.pred.ridge - y.test)^2)
```

```
##        V1
##  Min.   :0.000166
##  1st Qu.:0.038770
##  Median :0.136406
##  Mean   :0.495704
##  3rd Qu.:0.454224
##  Max.   :3.866447
```

LASSO

The LASSO algorithm for sparse multilinear regression is implemented in the lars package. Since lars executes different variants of LASSO, specify that you want to use `type="lasso"` which is the basic LASSO algorithm. Caution: Data Inputs should be in matrix form. Once you have fitted the model use plot directly on the resulting model to see the evolution of the model parameters when the regularization parameter changes.

You may see in which order the parameters are forced to zero value. Note that in the x-axis you have the L1 modulus of the coefficient vector compared to the maximum L1 modulus of the coefficient vector corresponding to the OLS solution. L1 modulus is calculated as the sum of the absolute values of the coefficients.

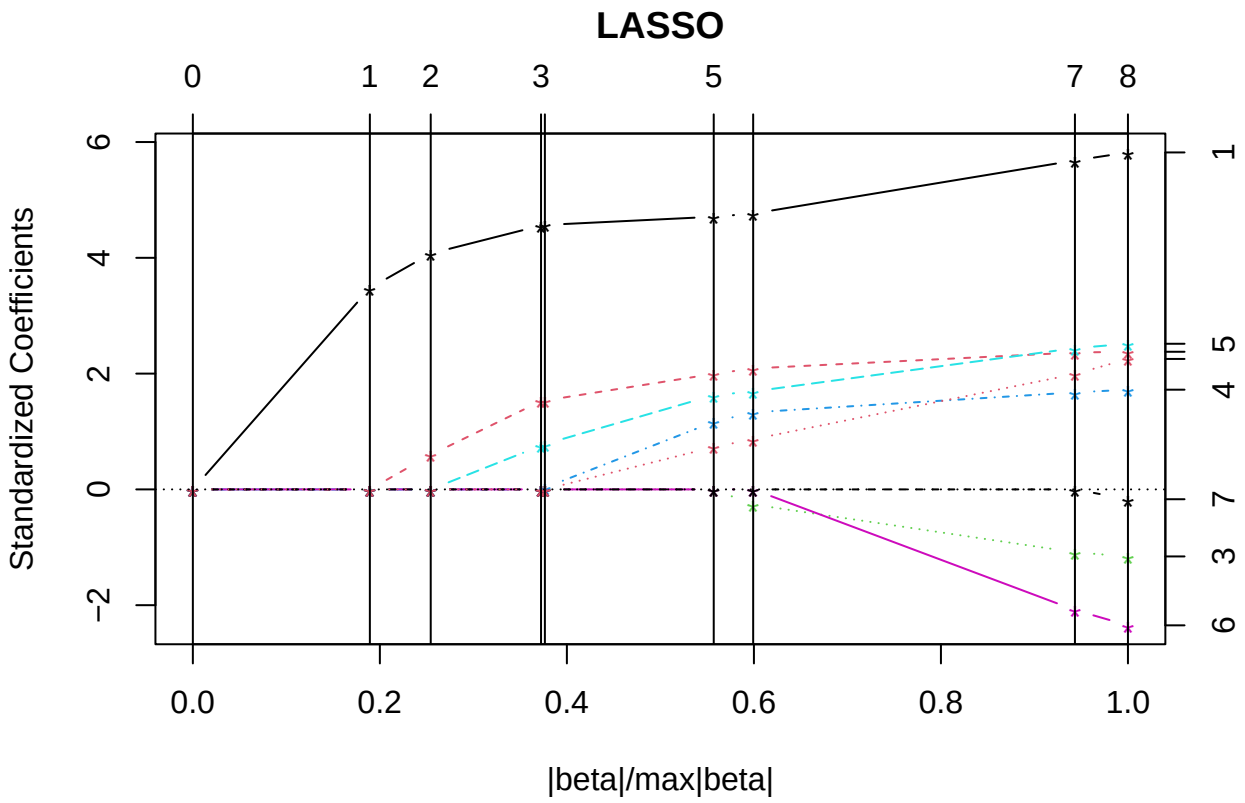
In order to determine the best regularization parameter we can use cross-validation. Specifically k-fold. For that purpose you can use`cv.lars`. In the help you will see that `cv.lars` uses `k = 10` as default. You may force `cv.lars` to produce a plot with the standard deviations of the MSE over the folds. Inspect the resulting plot and taking into account the parsimony principle think of the best model you may select. Beyond that

we can take the absolute minimum of the CV plot in order to select the regularization parameter. Do you think this is the simplest model that provides similar MSE. Propose an additional criteria to select the regularization parameter.

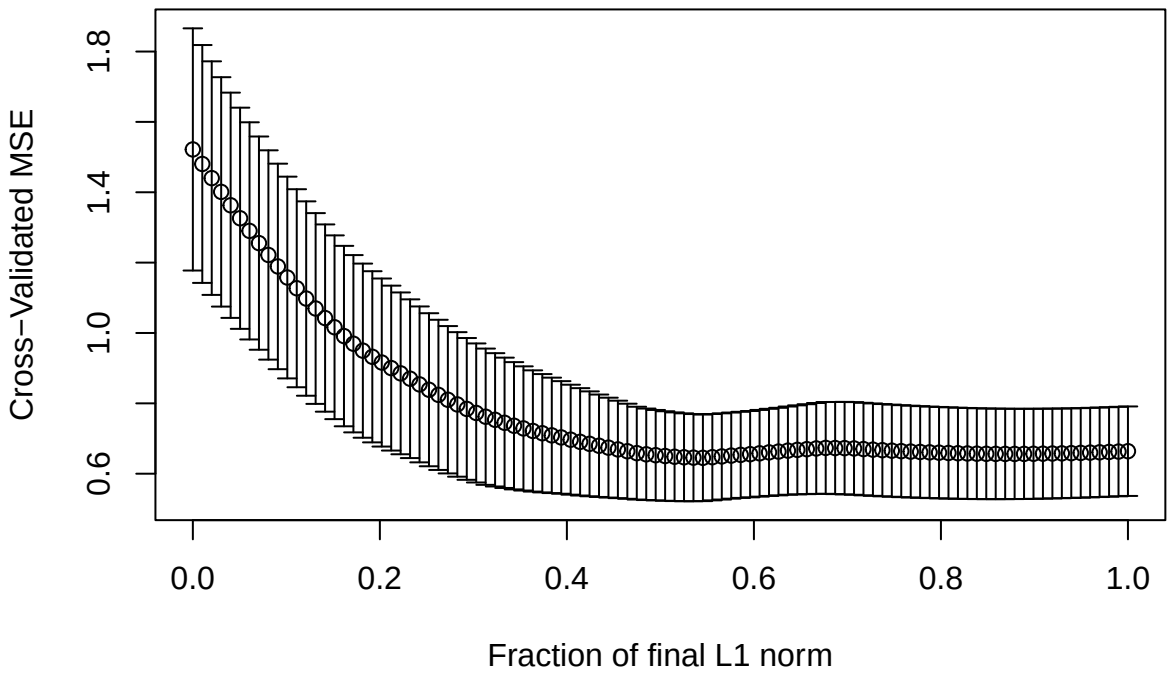
```
library(lars)
```

```
## Loaded lars 1.3
```

```
m.lasso <- lars(as.matrix(data.train[,1:8]),data.train$lpsa, type="lasso")
plot(m.lasso)
```




```
# Cross-validation
r <- cv.lars(as.matrix(data.train[,1:8]),data.train$lpsa, type = "lasso", plot.it = TRUE, se = TRUE)
```




```
##### Note 5/8/11: in the newer versions of lars package (> 0.9-8), the field r$fraction seems to have
bestfraction <- r$index[which.min(r$cv)]

# Observe coefficients
coef.lasso <- predict(m.lasso,as.matrix(data.test[,1:8]), s = bestfraction, type = "coefficient", mode="
coef.lasso
```

```
## $s
## [1] 0.5454545
##
## $fraction
## [1] 0.5454545
##
## $mode
## [1] "fraction"
##
## $coefficients
##     lcavol    lweight        age       lbph        svi        lcp    gleason
## 0.54878455 0.21751879 0.00000000 0.13333976 0.19085833 0.00000000 0.00000000
##      pgg45
## 0.08114078
```

```
coef.lasso2 <- predict(m.lasso,as.matrix(data.test[,1:8]), s = 0.55, type = "coefficient", mode = "frac
coef.lasso2
```

```
## $s
## [1] 0.55
```

```
##
## $fraction
## [1] 0.55
##
## $mode
## [1] "fraction"
##
## $coefficients
##     lcavol    lweight        age       lbph        svi        lcp    gleason
## 0.54920263 0.21880039 0.00000000 0.13683094 0.19343978 0.00000000 0.00000000
##      pgg45
## 0.08332334
```

```
# Prediction
y.pred.lasso <- predict(m.lasso,as.matrix(data.test[,1:8]), s = bestfraction, type = "fit", mode = "fra

# Test error
summary((y.pred.lasso - y.test)^2)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000001 0.063457 0.134963 0.456933 0.404173 3.565778
```

```
# Prediction parsimonious model
y.pred.lasso2 <- predict(m.lasso,as.matrix(data.test[,1:8]), s = 0.55, type = "fit", mode="fraction")$fi

# Test error
summary((y.pred.lasso2 - y.test)^2)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000001 0.060436 0.131433 0.457808 0.405377 3.565966
```

MUVR

MUVR is an algorithm to improve predictive performance and minimize overfitting and false positives in multivariate analysis. In the MUVR algorithm, minimal variable selection is achieved by performing **recursive variable elimination in a repeated double crossvalidation (rdCV) procedure**. The algorithm supports partial least squares and random forest modelling, and simultaneously identifies minimal-optimal and all-relevant variable sets for regression, classification and multilevel analyses.

By averaging variable ranks over the inner segments before variable reduction in each **recursive backwards elimination step**, potential overfitting that may occur during model training and variable ranking is minimized.

```
library(MUVR)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
cl <- parallel::makeCluster(parallel::detectCores()-1)
doParallel::registerDoParallel(cl)

model_PLS <- MUVR::MUVR(prost.std[ ,-9], prost.std[ , "lpsa"],
                        ML = F,
                        method ='PLS',
```

```
                          nRep = 20,
                          nOuter = 5,
                          varRatio = 0.8)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

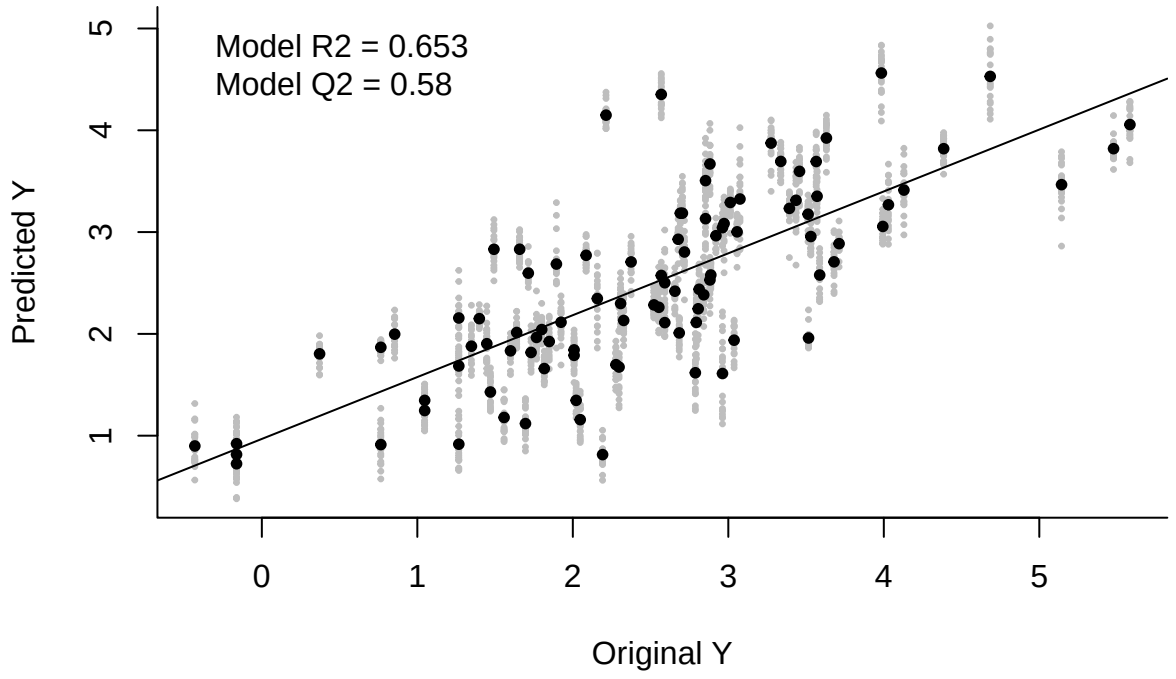
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
##
## Missing ID -> Assume all unique (i.e. sample independence)
## Missing fitness -> RMSEP
##  Elapsed time 0.1556667 mins
```

```
stopCluster(cl)
```

For final estimation of fitness and model predictive ability, Q2 is used for regression analysis, facilitating interpretation of modelling fitness, regardless of the scale of the original dependent variable (upper bounded by 1 for perfect prediction). This is in contrast to the inner validation loop, where RMSEP estimates fitness in the original response scale and is thus suitable for averaging.

```
plotMV(model_PLS)
```




```
cl <- parallel::makeCluster(parallel::detectCores()-1)
doParallel::registerDoParallel(cl)
```

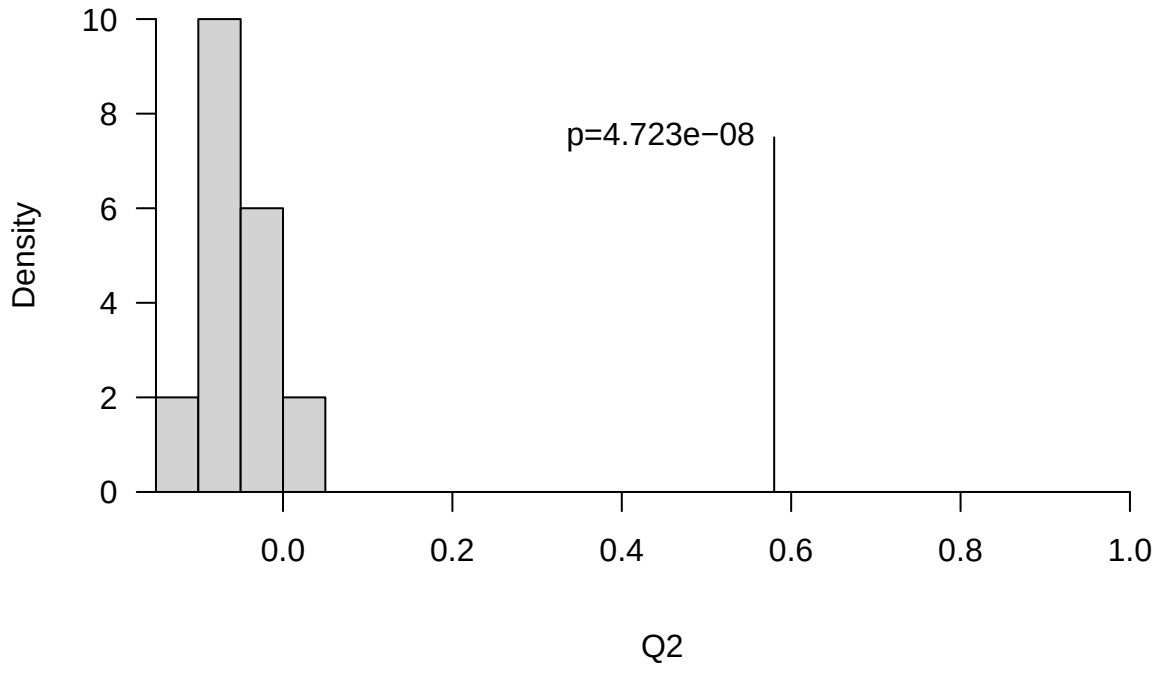
```r
perm <- MUVR::permutations(model_PLS, nPerm = 10)
```

```
##
## "model_PLS" permutation 1 of 10
##
##  Elapsed time 0.1106667 mins
##
## Estimated time left: 0.996 mins
##
##
## "model_PLS" permutation 2 of 10
##
##  Elapsed time 0.0945 mins
##
## Estimated time left: 0.8206667 mins
##
##
## "model_PLS" permutation 3 of 10
##
##  Elapsed time 0.09083333 mins
##
## Estimated time left: 0.6906667 mins
##
##
## "model_PLS" permutation 4 of 10
##
##  Elapsed time 0.09066667 mins
##
## Estimated time left: 0.58 mins
##
##
## "model_PLS" permutation 5 of 10
##
##  Elapsed time 0.09166667 mins
##
## Estimated time left: 0.4783333 mins
##
##
## "model_PLS" permutation 6 of 10
##
##  Elapsed time 0.1038333 mins
##
## Estimated time left: 0.3881111 mins
##
##
## "model_PLS" permutation 7 of 10
##
##  Elapsed time 0.1518333 mins
##
## Estimated time left: 0.3145714 mins
##
##
## "model_PLS" permutation 8 of 10
##
```

```
##  Elapsed time 0.1643333 mins
##
## Estimated time left: 0.2245833 mins
##
##
## "model_PLS" permutation 9 of 10
##
##  Elapsed time 0.1056667 mins
##
## Estimated time left: 0.1115926 mins
##
##
## "model_PLS" permutation 10 of 10
##
##  Elapsed time 0.105 mins
##
## Estimated time left: 0 mins
```

```
stopCluster(cl)
```

```
MUVR::permutationPlot(model_PLS, perm)
```




```
MUVR::getVIP(model_PLS)
```

```
##         order    name   rank
## lcavol      1  lcavol 1.0000
## lcp         2     lcp 2.5800
```

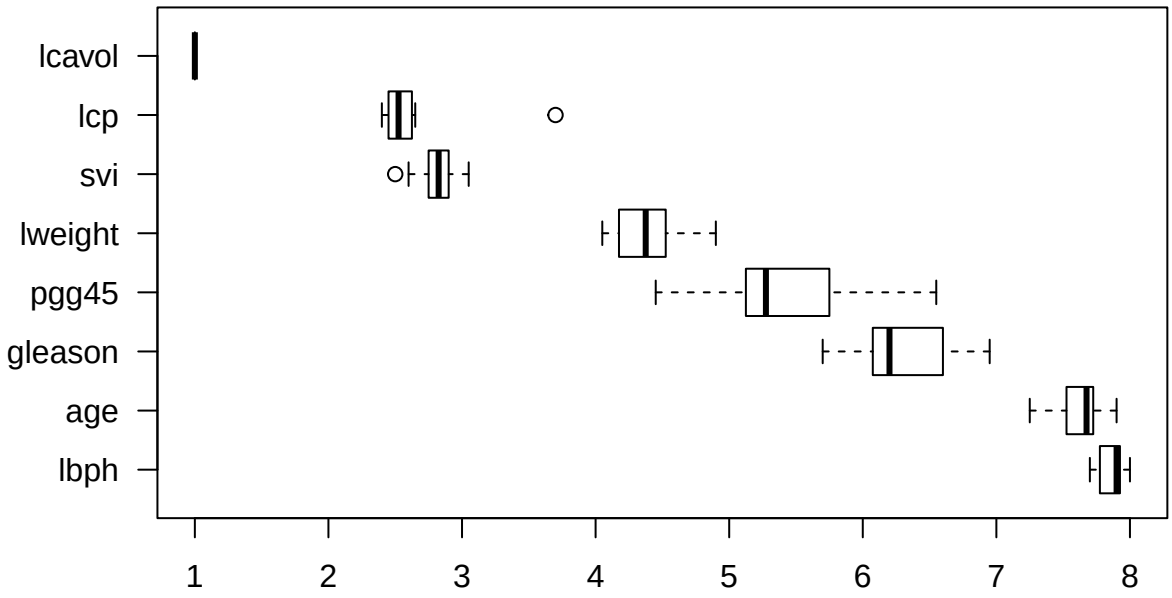
```
## svi         3     svi 2.8100
## lweight     4 lweight 4.3875
## pgg45       5   pgg45 5.3875
## gleason     6 gleason 6.2950
## age         7     age 7.6350
## lbph        8    lbph 7.8650
```

```
summary(m.bestsubset) # from OLS
```

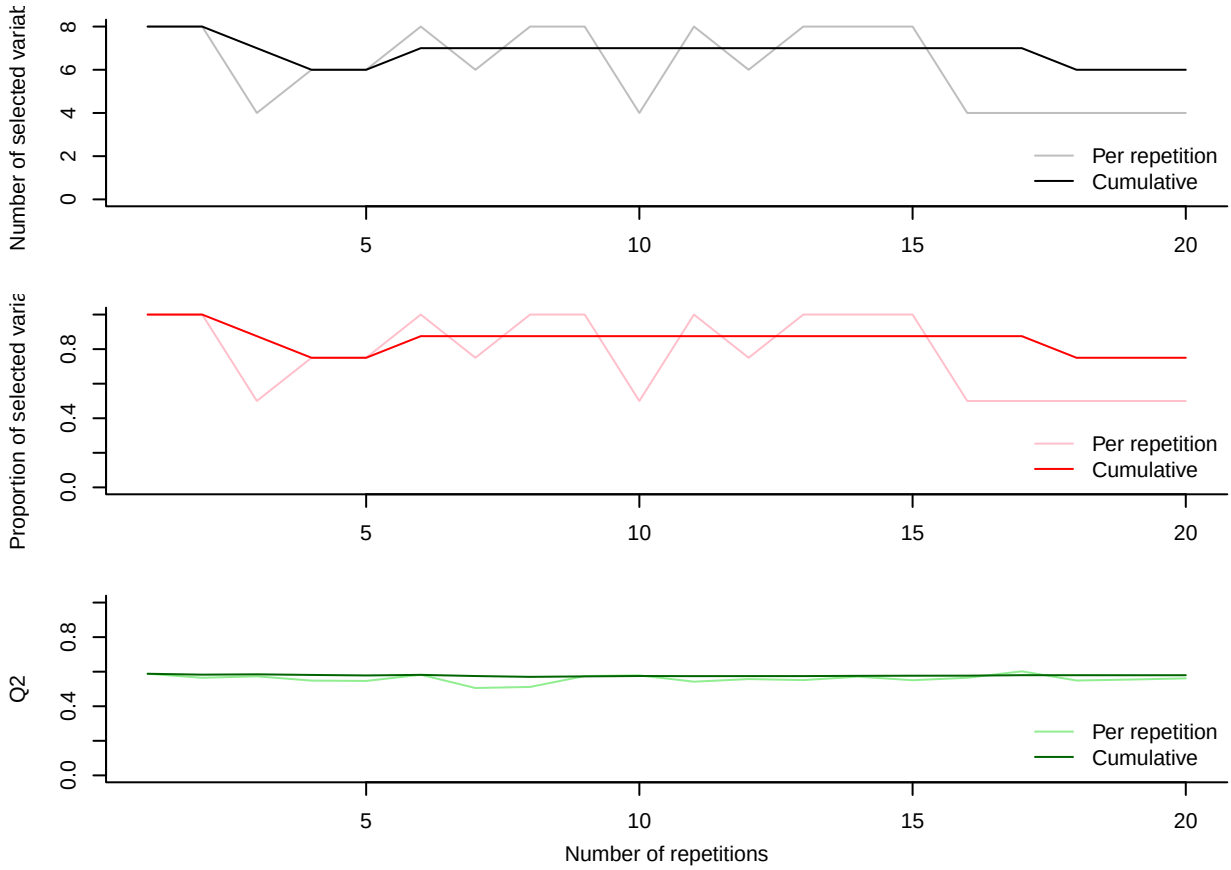
```
##
## Call:
## lm(formula = lpsa ~ ., data = data.train[, bestfeat])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.65425 -0.34471 -0.05615  0.44380  1.48952
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.46687    0.08760  28.161  < 2e-16 ***
## lcavol       0.67645    0.12384   5.462 9.88e-07 ***
## lweight      0.26528    0.09363   2.833   0.0063 **
## age         -0.14503    0.09757  -1.486   0.1425
## lbph         0.20953    0.10128   2.069   0.0430 *
## svi          0.30709    0.12190   2.519   0.0145 *
## lcp         -0.28722    0.15300  -1.877   0.0654 .
## pgg45        0.25228    0.11562   2.182   0.0331 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7064 on 59 degrees of freedom
## Multiple R-squared:  0.6943, Adjusted R-squared:  0.658
## F-statistic: 19.14 on 7 and 59 DF,  p-value: 4.496e-13
```

Variable ranking and selection are performed in the inner validation loop and final model performance is then assessed using observations in the test segment that is never used for model training or tuning. In each of the inner training models, variables are ranked by de facto standard techniques, i.e. variable importance of projection (VIP) for PLS analysis (Mehmood et al., 2011) and mean decrease in Gini index (classification) or mean decrease in accuracy (regression) for RF analysis (Strobl et al., 2007). For each iteration of the variable tuning, variable ranks are averaged between the inner models.

```
MUVR::plotVIP(model_PLS)
```

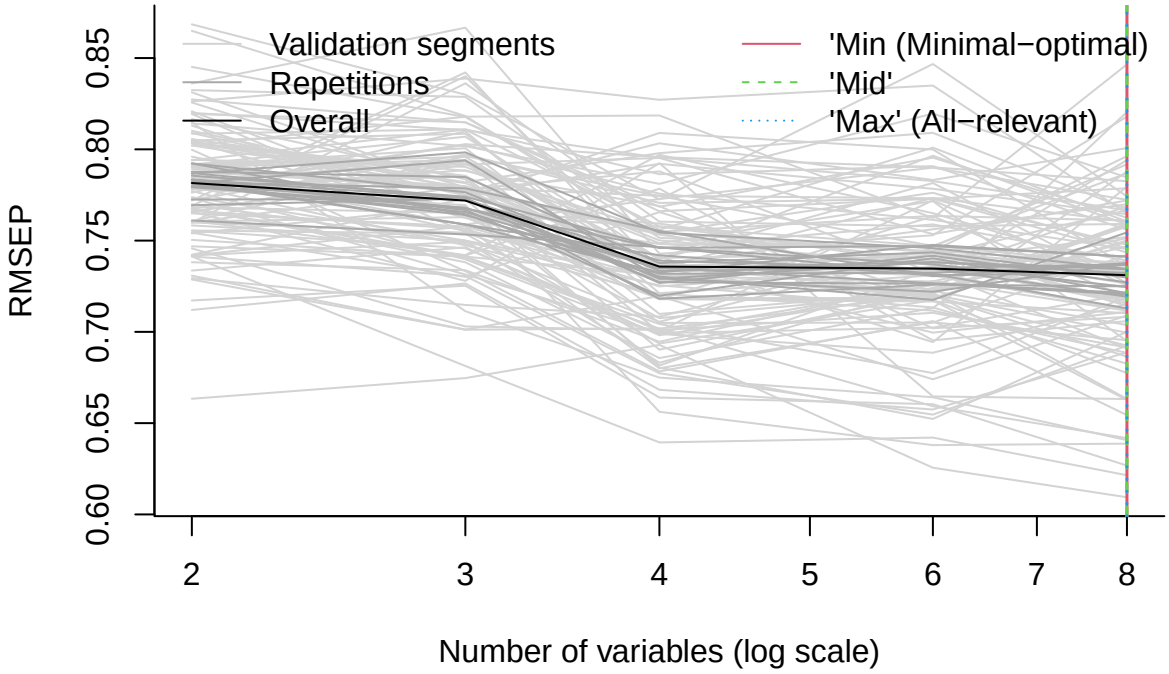



```
MUVR::plotStability(model_PLS)
```



Arbitration of model performance in variable tuning within the inner validation loop is performed using different fitness functions specifically adapted to the problem type: Root mean square error of prediction (RMSEP) for regression and number of misclassifications (MISS) for multilevel or general classification analysis (two or more classes). The area under the receiver operation characteristics curve (AUROC) and balanced error rate (BER) are supported as optional fitness metrics for classification.

```
MUVR::plotVAL(model_PLS)
```


RMSEP
0.60
0.65
0.70
0.75
0.80
0.85
2
3
4
5
6
7
8
Number of variables (log scale)
Validation segments
Repetitions
Overall
'Min (Minimal−optimal)
'Mid'
'Max' (All−relevant)