Ridge and Lasso

F. Chiaromonte (33%), J. Di Iorio (33%), L. Insolia (33%), L. Testa (1%)

March 14th 2023

Contents

Introduction **1**
Libraries . . . 1
Data . . . 1

Penalized regression **2**
Ridge . . . 2
Lasso . . . 7
Unbiasing the LASSO . . . 11
A more realistic setting . . . 11

Digging in the irrepresentability condition **18**

Nonconvex penalization methods **20**

Introduction

Libraries

We are going to use a few libraries:

- glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models
- tidyverse:
- caret: Classification and Regression Training
- ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics
- corrplot: Correlation matrix plotting

```
library(glmnet)     # ridge and lasso for GLMs
library(tidyverse)  # data manipulation and visualization
library(caret)      # statistical learning techniques
library(ggplot2)    # plots
library(corrplot)   # correlation matrix plotting
```

Data

We will use the **Body Fat dataset** (which is available in the Datasets folder of our course).

The data concerns a sample of 252 men, and contains 15 variables:

- Density of the body, determined from underwater weighing

- Percentage of body fat, calculated as a function of the Density according to Siri's equation: (495/Density) – 450.
- Indicator for Age group (binary; 0: up to 45 years, 1: over 45)
- Weight (lbs)
- Height (inches)
- Neck circumference (cm)
- Chest circumference (cm)
- Abdomen circumference (cm)
- Hip circumference (cm)
- Thigh circumference (cm)
- Knee circumference (cm)
- Ankle circumference (cm)
- Biceps circumference (cm)
- Forearm circumference (cm)
- Wrist circumference (cm)

We want to understand whether we can reliably describe and predict body fat percentage on the basis of these variables, using regression methods. For age, we only have a binary indicator separating men below and above 45 years. The body measurements, on the other hand, are all continuous variables. Please see the data description file for more details.

```
df <- read.table('BODY_FAT.TXT', header=TRUE)
names(df)
```

```
##  [1] "Density"  "SiriBF."  "Over45"   "Weight"   "Height"   "NeckC"
##  [7] "ChestC"   "AbdomenC" "HipC"     "ThighC"   "KneeC"    "AnkleC"
## [13] "BicepsC"  "ForearmC" "WristC"
```

We want to predict "SiriBF." using the other features, aside from "Density". So we drop the "Density" column.

```
df <- df[,-1]
```

Penalized regression

We will perform ridge/lasso penalization through the **glmnet** package. Let us identify predictors and response variable

```
# getting the predictors
x_var <- data.matrix(df[,-1])     # NOTE: glmnet requires a matrix structure
# getting the response variable
y_var <- df[,"SiriBF."]
```

Let's have a look a the glmnet function:

```
help(glmnet)
```

Note that:

- input matrix
- response variable
- α is the elastic-net mixing parameter with range [0, 1]. Namely, $\alpha = 1$ is the lasso (default) and $\alpha = 0$ is the ridge.
- standardize is a logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE.

Ridge

To perform ridge regression, we run glmnet with $\alpha = 0$. The λ's sequence is internally computed by the package itself – although a user-defined sequence can be provided as a *lambda* argument.

```
ridge <- glmnet(x_var, y_var, alpha=0)
summary(ridge)
```

```
##           Length Class     Mode
## a0         100   -none-    numeric
## beta      1300   dgCMatrix S4
## df         100   -none-    numeric
## dim          2   -none-    numeric
## lambda     100   -none-    numeric
## dev.ratio  100   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

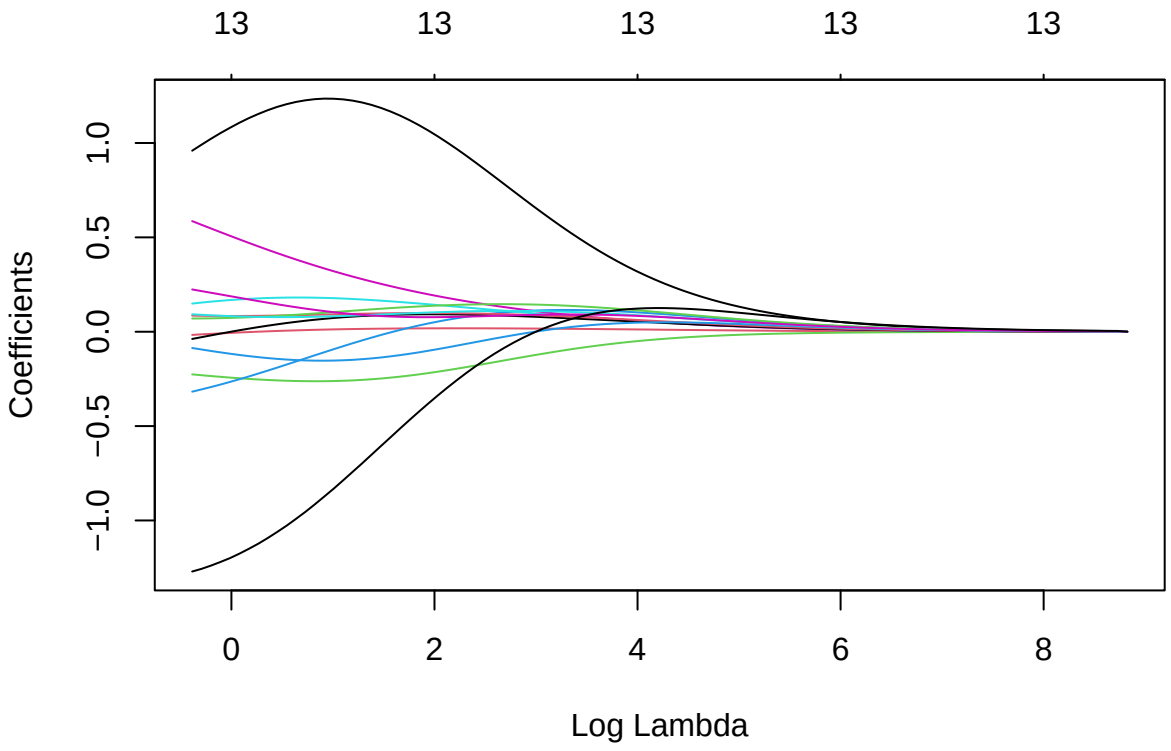
The summary is quite different than the one for linear regression, since ridge regression requires the tuning of λ. The code above fits a ridge regression for each λ value, and we have access to each of these model estimates.

We can plot the regularization path as follows:

```
dim(ridge$beta)
```

```
## [1]  13 100
```

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



We can automate the task of finding the optimal lambda value using the **cv.glmnet** function. This performs a k-fold cross-validation for glmnet, produces a plot, and returns "optimal'' λ values.

```
cv_ridge <- cv.glmnet(x_var, y_var, alpha = 0)
cv_ridge
```

```
##
## Call:  cv.glmnet(x = x_var, y = y_var, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min 0.6794   100   23.15 1.485      13
## 1se 1.0818    95   24.44 1.647      13
```

Two particular values of λ are highlighted: the minimum (min) and the largest value of lambda such that error is within 1 standard error of the minimum (1se).

```
cv_ridge$lambda.min
```

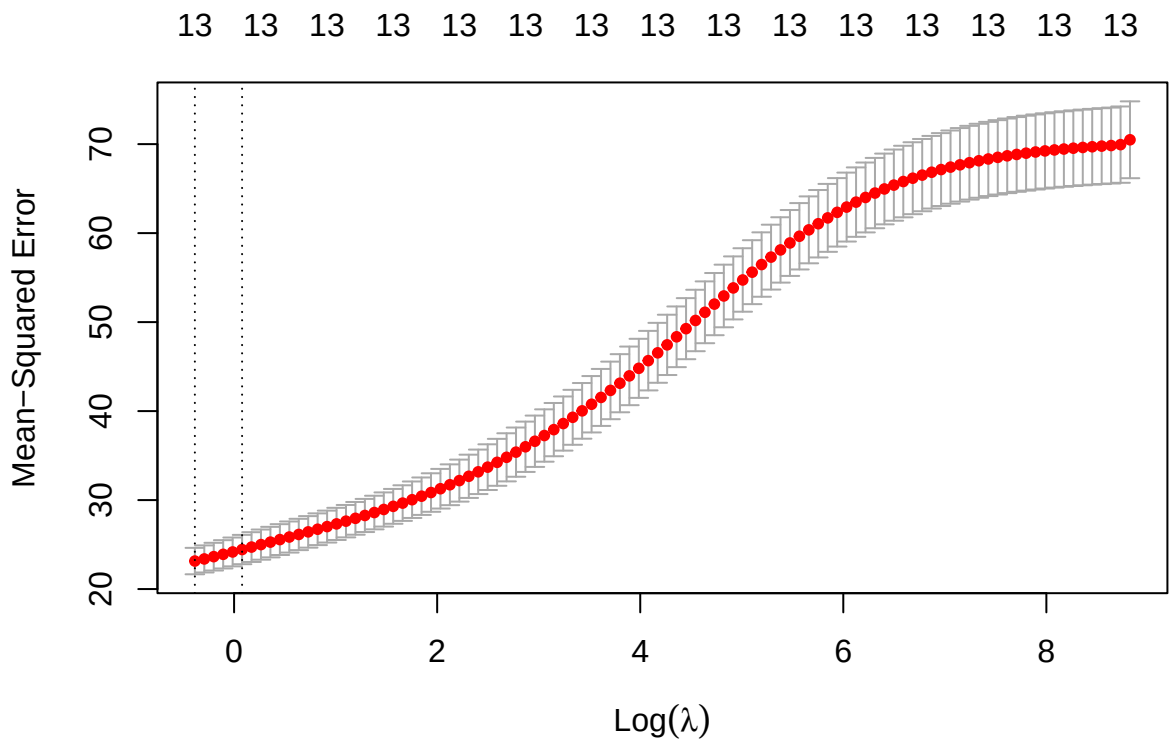
```
## [1] 0.6793883
```

```
cv_ridge$lambda.1se
```

```
## [1] 1.081778
```

We can visualize them in this way:

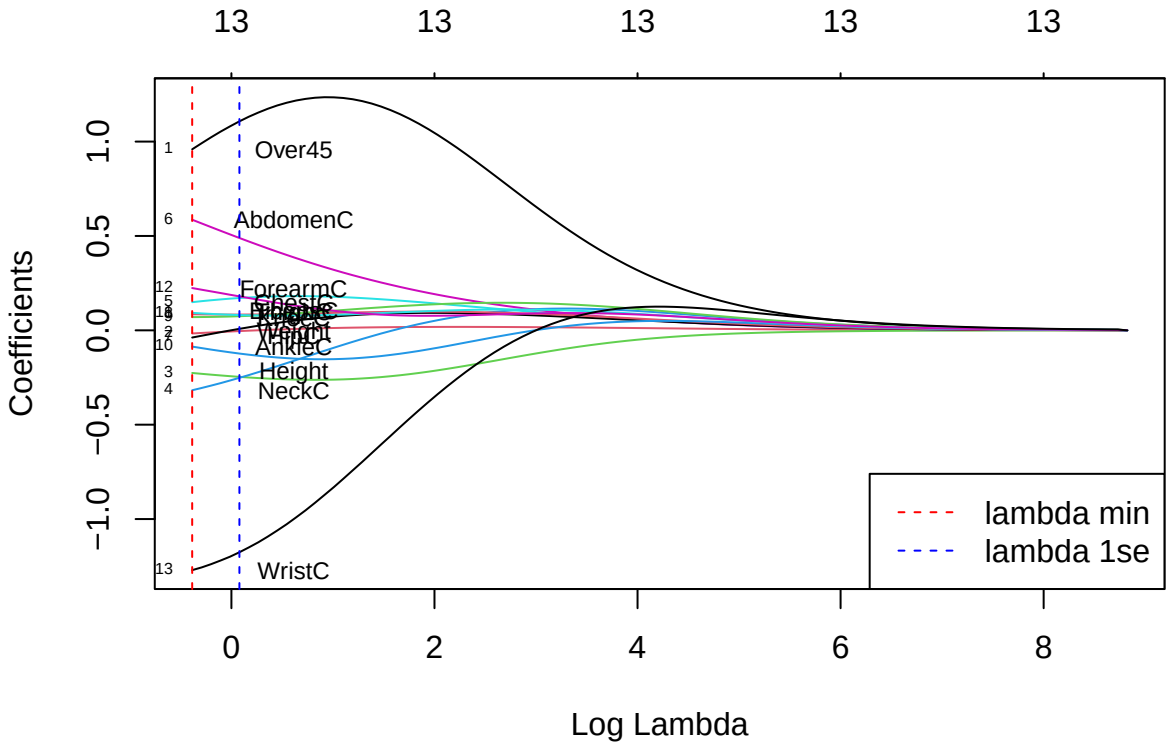
```
plot(cv_ridge)
```



Let us see again how the regression coefficients change by modifying λ, highlighting the min and 1se values:

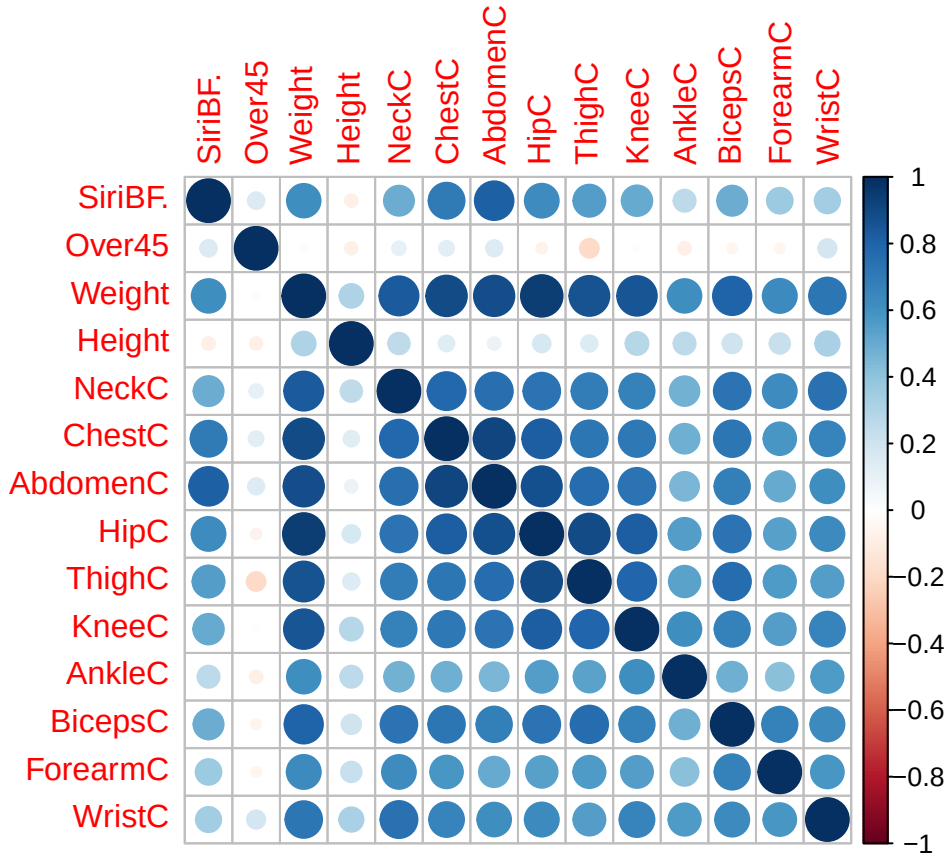
```
lbs_fun <- function(fit, offset_x=1, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])+ offset_x
```

```
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, cex=0.75, ...)
}

plot(ridge, xvar = "lambda", label=T)
lbs_fun(ridge)
abline(v=log(cv_ridge$lambda.min), col = "red", lty=2)
abline(v=log(cv_ridge$lambda.1se), col="blue", lty=2)
legend(x = "bottomright",
       legend = c("lambda min", "lambda 1se"),
       lty = c(2, 2),
       col = c("red", "blue"))
```



WristC has a very strong negative effect on the response. Why is that? We can find a partial answer by looking at the correlation matrix!

```
corrplot(cor(df))
```



Let's re-fit the model and see the estimates associated to the minimum λ.

```
min_ridge <- glmnet(x_var, y_var, alpha=0, lambda= cv_ridge$lambda.min)
coef(min_ridge)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -7.71880406
## Over45       0.95953292
## Weight      -0.01619205
## Height      -0.22630209
## NeckC       -0.31617445
## ChestC       0.14889642
## AbdomenC     0.58565111
## HipC        -0.03875703
## ThighC       0.08495378
## KneeC        0.06923480
## AnkleC      -0.08687724
## BicepsC      0.09165336
## ForearmC     0.22347815
## WristC      -1.27272327
```

We can use this model to make predictions on the training set.

```
# Make predictions on the training data
predictions <- min_ridge %>% predict(x_var) %>% as.vector()
# Model performance metrics
data.frame(
```

```
  RMSE = RMSE(predictions, y_var),
  Rsquare = R2(predictions, y_var)
)
```

```
##       RMSE  Rsquare
## 1 4.494532 0.714938
```

Be careful though! We are making predictions and assessing the goodness of fit based on training data. Is it the best choice? Do you have any other suggestions?

Lasso

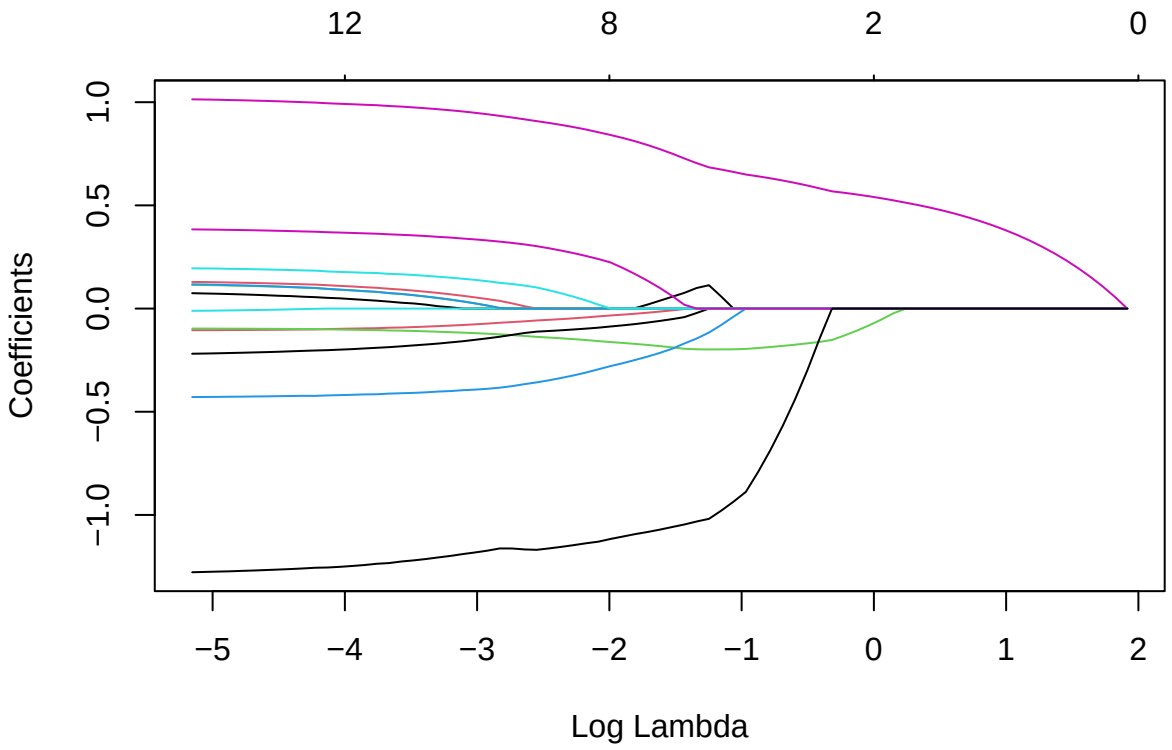
Let us now perform Lasso regression using the **glmnet** package. We follow the same approach as in Ridge regression, but set $\alpha = 1$.

```
lasso <- glmnet(x_var, y_var, alpha=1)
summary(lasso)
```

```
##           Length Class     Mode
## a0          77   -none-    numeric
## beta      1001   dgCMatrix S4
## df          77   -none-    numeric
## dim          2   -none-    numeric
## lambda      77   -none-    numeric
## dev.ratio   77   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

Let's have a look at the selection path:

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

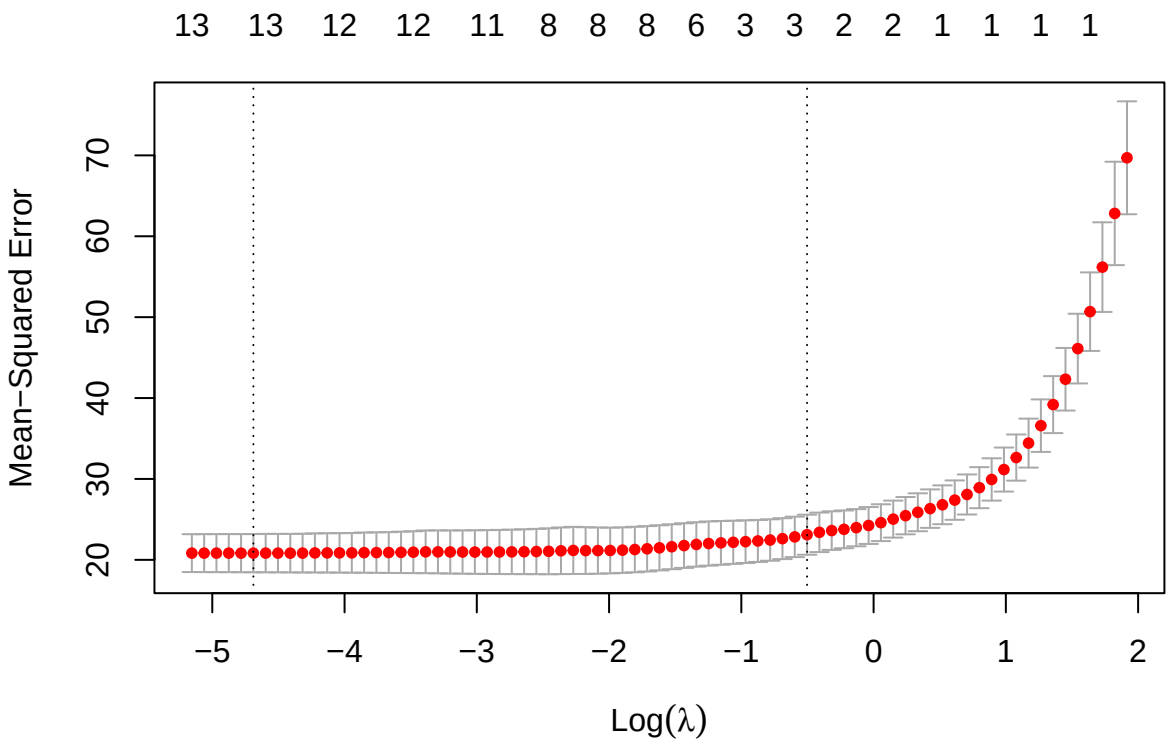


Once again, we need to tune the sparsity parameter λ. We use k-fold cross-validation through the **cv.glmnet** function.

```
cv_lasso <- cv.glmnet(x_var, y_var, alpha = 1)
cv_lasso
```

```
##
## Call:  cv.glmnet(x = x_var, y = y_var, alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min 0.0092    72   20.82 2.362      13
## 1se 0.6048    27   23.11 2.475       3
```

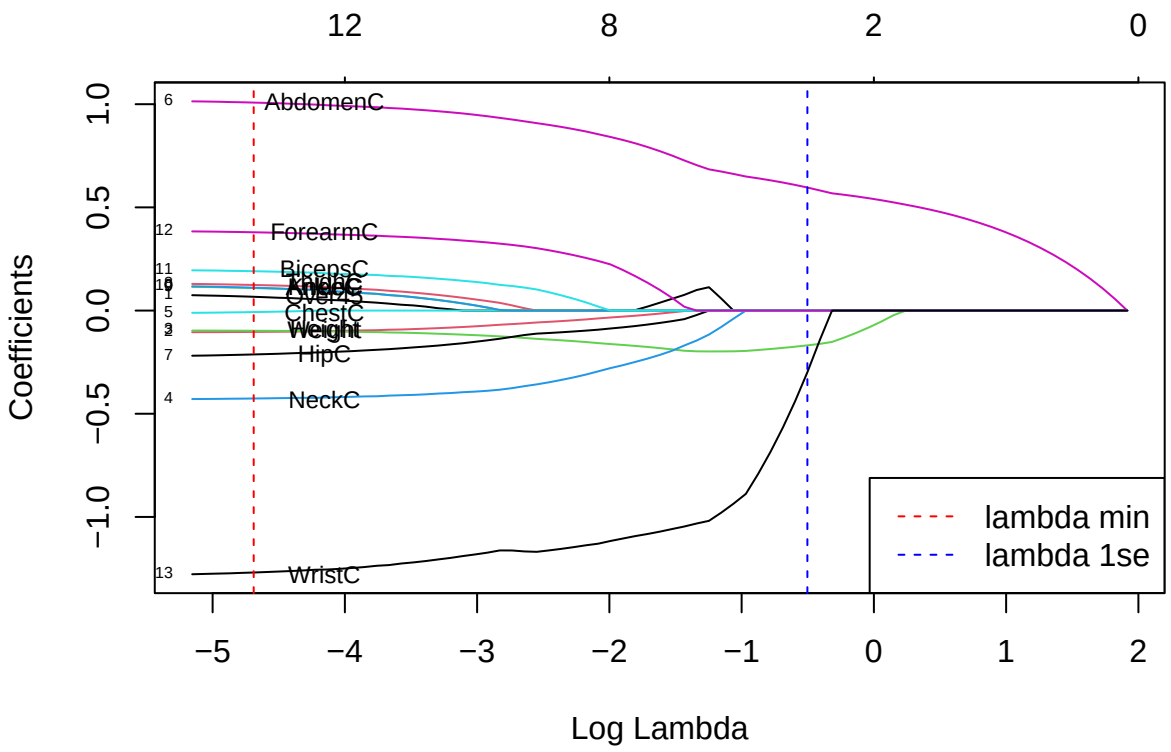
```
plot(cv_lasso)
```



Also here, it outputs the min and the 1se λ. As expected, the number of non-zero coefficients (which is printed on top of the previous plot) is lower than the one for Ridge regression.

Let us see again how the regression coefficients change by modifying λ:

```
lbs_fun <- function(fit, offset_x=1, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])+ offset_x
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, cex=0.75, ...)
}

plot(lasso, xvar = "lambda", label=T)
lbs_fun(lasso)
abline(v=log(cv_lasso$lambda.min), col = "red", lty=2)
abline(v=log(cv_lasso$lambda.1se), col="blue", lty=2)
legend(x = "bottomright",
       legend = c("lambda min", "lambda 1se"),
       lty = c(2, 2),
       col = c("red", "blue"))
```



Let us rebuild the model and compare the estimated coefficients for min and 1se λ.

```
min_lasso <- glmnet(x_var, y_var, alpha=1, lambda= cv_lasso$lambda.min)
se_lasso <- glmnet(x_var, y_var, alpha=1, lambda= cv_lasso$lambda.1se)

lasso_mat <- cbind(coef(min_lasso), coef(se_lasso))
colnames(lasso_mat) <- c("min", "1se")
lasso_mat
```

```
## 14 x 2 sparse Matrix of class "dgCMatrix"
##                         min         1se
## (Intercept) -18.424139414 -18.7474691
## Over45        0.061131213   .
## Weight       -0.102886525   .
## Height       -0.098002039  -0.1685055
## NeckC        -0.425387495   .
## ChestC       -0.008547423   .
## AbdomenC      1.009654915   0.5962716
## HipC         -0.215942023   .
## ThighC        0.124423263   .
## KneeC         0.108754746   .
## AnkleC        0.111119029   .
## BicepsC       0.190162642   .
## ForearmC      0.378702627   .
## WristC       -1.270980197  -0.3000459
```

We can use this model to make predictions on the training set.

```
# Make predictions on the training data
predictions <- se_lasso %>% predict(x_var) %>% as.vector()
# Model performance metrics
data.frame(
```

```
  RMSE = RMSE(predictions, y_var),
  Rsquare = R2(predictions, y_var)
)
```

```
##       RMSE   Rsquare
## 1 4.700132 0.6905285
```

Unbiasing the LASSO

You have learnt that LASSO estimates are biased downwards, i.e. the coefficients are shrunk towards 0. An approach to debias them is the following:

- run a LASSO on the original data set;
- run a OLS on the set of variables selected by the LASSO in the previous step.

Perhaps, compare the following two regression methods:

```
predictions_bias <- min_lasso %>% predict(x_var) %>% as.vector()

coefs = which(coef(min_lasso)!=0) - 1
coefs = coefs[2:length(coefs)]
prediction_2step <- lm(y_var ~ ., data = as.data.frame(x_var[, coefs])) %>% predict(as.data.frame(x_var

# Model performance metrics
data.frame(
  RMSE_bias = RMSE(predictions_bias, y_var),
  Rsquare_bias = R2(predictions_bias, y_var),
  RMSE_2step = RMSE(prediction_2step, y_var),
  Rsquare_2step = R2(prediction_2step, y_var)
)
```

```
##   RMSE_bias Rsquare_bias RMSE_2step Rsquare_2step
## 1  4.217001    0.7450832   4.216107     0.7451821
```

A more realistic setting

Let's try to overfit and look at the MSE increase again!

```
# take all pairwise interactions
dim(x_var)
```

```
## [1] 252  13
```

```
newx <- as.data.frame(x_var)
newx <- model.matrix(~ .^2, data=newx)
dim(newx)
```

```
## [1] 252  92
```

```
colnames(newx)
```

```
##  [1] "(Intercept)"       "Over45"            "Weight"
##  [4] "Height"            "NeckC"             "ChestC"
##  [7] "AbdomenC"          "HipC"              "ThighC"
## [10] "KneeC"             "AnkleC"            "BicepsC"
## [13] "ForearmC"          "WristC"            "Over45:Weight"
## [16] "Over45:Height"     "Over45:NeckC"      "Over45:ChestC"
## [19] "Over45:AbdomenC"   "Over45:HipC"       "Over45:ThighC"
```

```
## [22] "Over45:KneeC"      "Over45:AnkleC"     "Over45:BicepsC"
## [25] "Over45:ForearmC"   "Over45:WristC"     "Weight:Height"
## [28] "Weight:NeckC"      "Weight:ChestC"     "Weight:AbdomenC"
## [31] "Weight:HipC"       "Weight:ThighC"     "Weight:KneeC"
## [34] "Weight:AnkleC"     "Weight:BicepsC"    "Weight:ForearmC"
## [37] "Weight:WristC"     "Height:NeckC"      "Height:ChestC"
## [40] "Height:AbdomenC"   "Height:HipC"       "Height:ThighC"
## [43] "Height:KneeC"      "Height:AnkleC"     "Height:BicepsC"
## [46] "Height:ForearmC"   "Height:WristC"     "NeckC:ChestC"
## [49] "NeckC:AbdomenC"    "NeckC:HipC"        "NeckC:ThighC"
## [52] "NeckC:KneeC"       "NeckC:AnkleC"      "NeckC:BicepsC"
## [55] "NeckC:ForearmC"    "NeckC:WristC"      "ChestC:AbdomenC"
## [58] "ChestC:HipC"       "ChestC:ThighC"     "ChestC:KneeC"
## [61] "ChestC:AnkleC"     "ChestC:BicepsC"    "ChestC:ForearmC"
## [64] "ChestC:WristC"     "AbdomenC:HipC"     "AbdomenC:ThighC"
## [67] "AbdomenC:KneeC"    "AbdomenC:AnkleC"   "AbdomenC:BicepsC"
## [70] "AbdomenC:ForearmC" "AbdomenC:WristC"   "HipC:ThighC"
## [73] "HipC:KneeC"        "HipC:AnkleC"       "HipC:BicepsC"
## [76] "HipC:ForearmC"     "HipC:WristC"       "ThighC:KneeC"
## [79] "ThighC:AnkleC"     "ThighC:BicepsC"    "ThighC:ForearmC"
## [82] "ThighC:WristC"     "KneeC:AnkleC"      "KneeC:BicepsC"
## [85] "KneeC:ForearmC"    "KneeC:WristC"      "AnkleC:BicepsC"
## [88] "AnkleC:ForearmC"   "AnkleC:WristC"     "BicepsC:ForearmC"
## [91] "BicepsC:WristC"    "ForearmC:WristC"
```

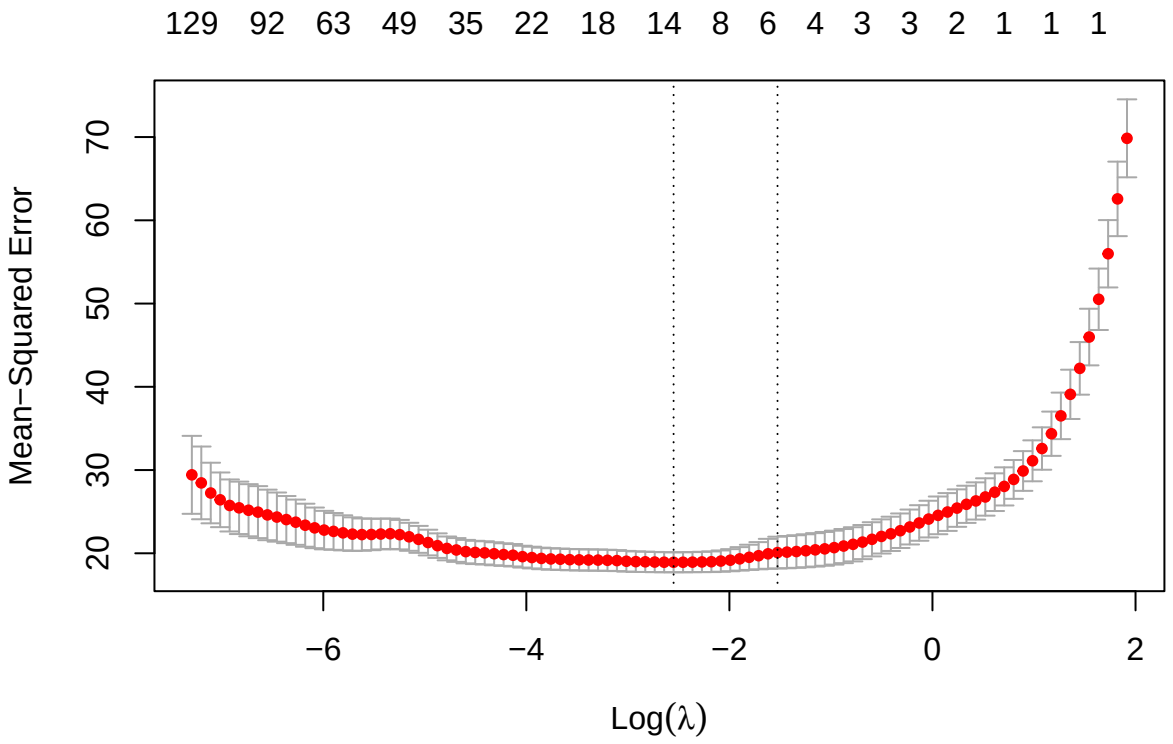
```
# take their squared terms too!
newx <- as.matrix(newx)
newx <- cbind(newx, newx[, 2:ncol(newx)]^2)
dim(newx)
```

```
## [1] 252 183
```

```
cv_lasso <- cv.glmnet(newx, y_var, alpha = 1)
cv_lasso
```

```
##
## Call:  cv.glmnet(x = newx, y = y_var, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure    SE Nonzero
## min 0.07811    49   18.92 1.201      16
## 1se 0.21735    38   20.09 1.908       6
```

```
plot(cv_lasso)
```




```
coef(cv_lasso)
```

```
## 184 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)       -4.281864e+01
## (Intercept)        .
## Over45             .
## Weight             .
## Height             .
## NeckC              .
## ChestC             .
## AbdomenC           7.923206e-01
## HipC               .
## ThighC             .
## KneeC              .
## AnkleC             .
## BicepsC            .
## ForearmC           .
## WristC             .
## Over45:Weight      .
## Over45:Height      .
## Over45:NeckC       .
## Over45:ChestC      .
## Over45:AbdomenC    .
## Over45:HipC        .
## Over45:ThighC      .
## Over45:KneeC       .
## Over45:AnkleC      .
## Over45:BicepsC     .
## Over45:ForearmC    .
## Over45:WristC      .
```

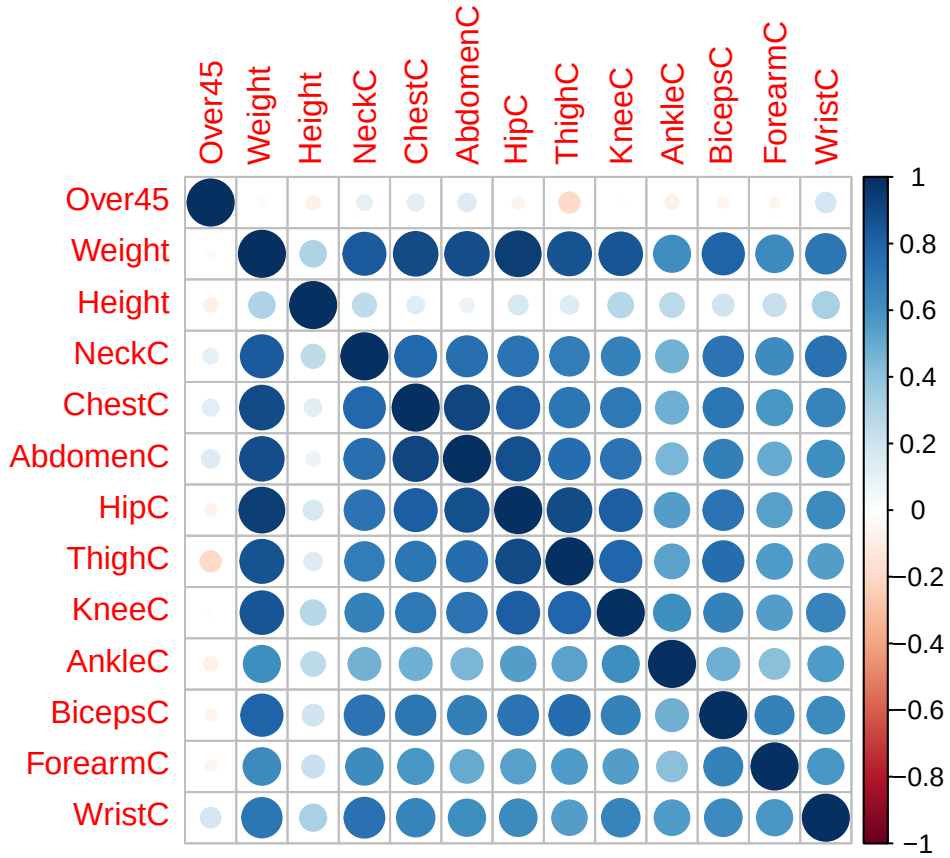
```
## Weight:Height      .
## Weight:NeckC       .
## Weight:ChestC      .
## Weight:AbdomenC    .
## Weight:HipC        .
## Weight:ThighC      .
## Weight:KneeC       .
## Weight:AnkleC      .
## Weight:BicepsC     .
## Weight:ForearmC    .
## Weight:WristC      .
## Height:NeckC       .
## Height:ChestC      .
## Height:AbdomenC    .
## Height:HipC        .
## Height:ThighC      .
## Height:KneeC       .
## Height:AnkleC      .
## Height:BicepsC     .
## Height:ForearmC    .
## Height:WristC      .
## NeckC:ChestC       .
## NeckC:AbdomenC     .
## NeckC:HipC         .
## NeckC:ThighC       .
## NeckC:KneeC        .
## NeckC:AnkleC       .
## NeckC:BicepsC      .
## NeckC:ForearmC     .
## NeckC:WristC       .
## ChestC:AbdomenC    .
## ChestC:HipC        .
## ChestC:ThighC      .
## ChestC:KneeC       .
## ChestC:AnkleC      .
## ChestC:BicepsC     .
## ChestC:ForearmC    .
## ChestC:WristC      .
## AbdomenC:HipC      .
## AbdomenC:ThighC    .
## AbdomenC:KneeC     .
## AbdomenC:AnkleC    .
## AbdomenC:BicepsC   .
## AbdomenC:ForearmC  3.351450e-04
## AbdomenC:WristC    .
## HipC:ThighC        .
## HipC:KneeC         .
## HipC:AnkleC        .
## HipC:BicepsC       .
## HipC:ForearmC      .
## HipC:WristC        .
## ThighC:KneeC       .
## ThighC:AnkleC      .
## ThighC:BicepsC     .
```

```
## ThighC:ForearmC    .
## ThighC:WristC      .
## KneeC:AnkleC       .
## KneeC:BicepsC      .
## KneeC:ForearmC     .
## KneeC:WristC       .
## AnkleC:BicepsC     .
## AnkleC:ForearmC    .
## AnkleC:WristC      .
## BicepsC:ForearmC   .
## BicepsC:WristC     .
## ForearmC:WristC    .
## Over45             .
## Weight             .
## Height             .
## NeckC              .
## ChestC             .
## AbdomenC           .
## HipC               .
## ThighC             .
## KneeC              .
## AnkleC             .
## BicepsC            .
## ForearmC           .
## WristC             .
## Over45:Weight      .
## Over45:Height      .
## Over45:NeckC       .
## Over45:ChestC      .
## Over45:AbdomenC    .
## Over45:HipC        .
## Over45:ThighC      .
## Over45:KneeC       .
## Over45:AnkleC      .
## Over45:BicepsC     .
## Over45:ForearmC    .
## Over45:WristC      .
## Weight:Height      .
## Weight:NeckC       .
## Weight:ChestC      .
## Weight:AbdomenC    .
## Weight:HipC       -7.037316e-09
## Weight:ThighC      .
## Weight:KneeC       .
## Weight:AnkleC      .
## Weight:BicepsC     .
## Weight:ForearmC    .
## Weight:WristC      .
## Height:NeckC      -1.576541e-07
## Height:ChestC      .
## Height:AbdomenC    .
## Height:HipC        .
## Height:ThighC      .
## Height:KneeC       .
```

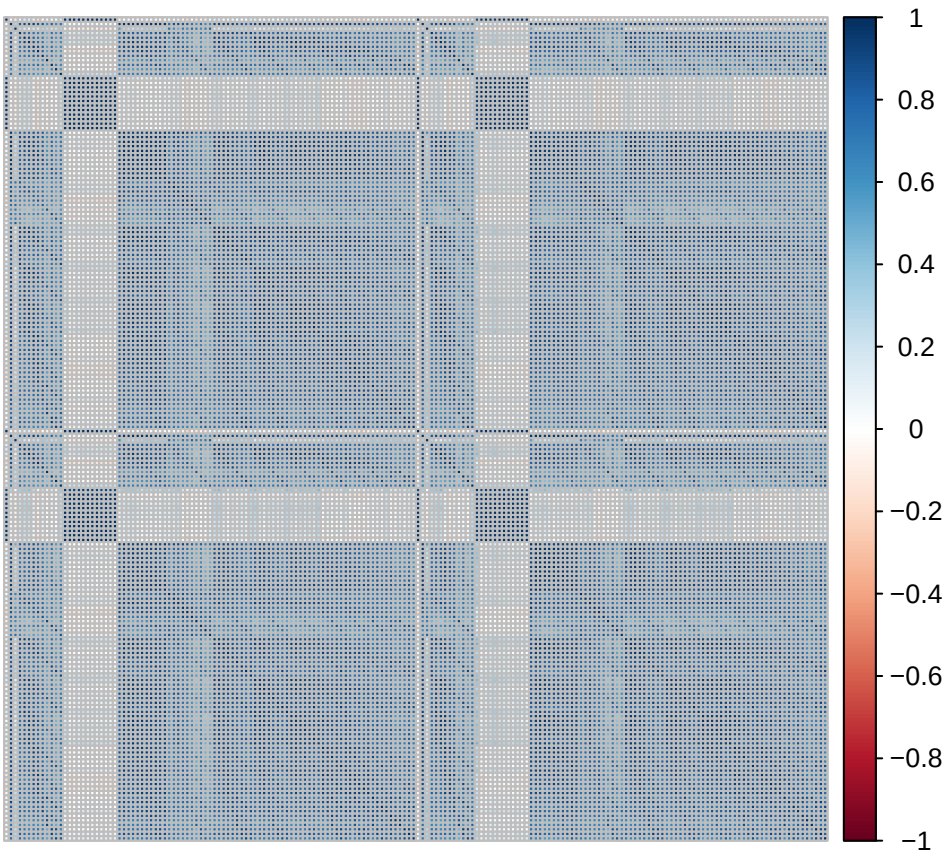
```
## Height:AnkleC      .
## Height:BicepsC     .
## Height:ForearmC    .
## Height:WristC     -4.701473e-06
## NeckC:ChestC       .
## NeckC:AbdomenC     .
## NeckC:HipC         .
## NeckC:ThighC       .
## NeckC:KneeC        .
## NeckC:AnkleC       .
## NeckC:BicepsC      .
## NeckC:ForearmC     .
## NeckC:WristC      -1.912811e-06
## ChestC:AbdomenC    .
## ChestC:HipC        .
## ChestC:ThighC      .
## ChestC:KneeC       .
## ChestC:AnkleC      .
## ChestC:BicepsC     .
## ChestC:ForearmC    .
## ChestC:WristC      .
## AbdomenC:HipC      .
## AbdomenC:ThighC    .
## AbdomenC:KneeC     .
## AbdomenC:AnkleC    .
## AbdomenC:BicepsC   .
## AbdomenC:ForearmC  .
## AbdomenC:WristC    .
## HipC:ThighC        .
## HipC:KneeC         .
## HipC:AnkleC        .
## HipC:BicepsC       .
## HipC:ForearmC      .
## HipC:WristC        .
## ThighC:KneeC       .
## ThighC:AnkleC      .
## ThighC:BicepsC     .
## ThighC:ForearmC    .
## ThighC:WristC      .
## KneeC:AnkleC       .
## KneeC:BicepsC      .
## KneeC:ForearmC     .
## KneeC:WristC       .
## AnkleC:BicepsC     .
## AnkleC:ForearmC    .
## AnkleC:WristC      .
## BicepsC:ForearmC   .
## BicepsC:WristC     .
## ForearmC:WristC    .
```

However, lasso assumptions are unlikely to hold due to strong correlations between predictors.

```
# original data
corrplot(cor(x_var))
```




```
# transformed data
newx = as.data.frame(newx[, 2:ncol(newx)])
corrplot(cor(newx), tl.pos='n')
```



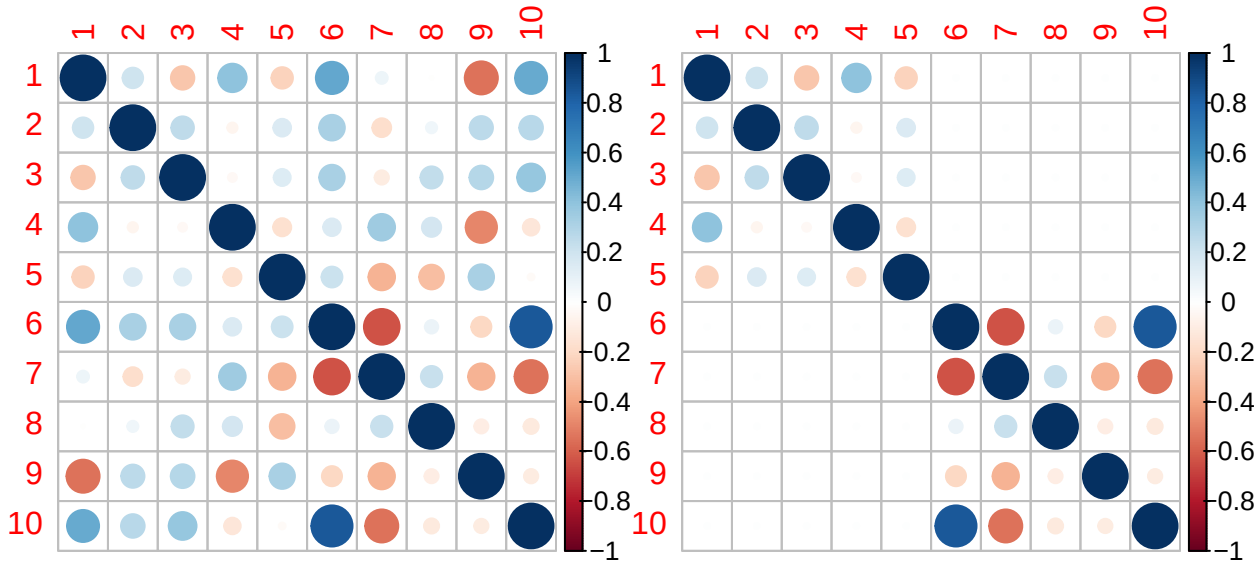
Digging in the irrepresentability condition

As you have seen before, the variance-covariance matrix can tell us whether LASSO will work (at least asymptotically) on the data at our disposal. Let's have a look at an example (available here).

```
set.seed(12345) # Seed for replication

library(mvtnorm) # Sampling from a multivariate Normal
library(clusterGeneration) # Random matrix generation

p = 10 # = Number of Candidate Variables
k = 5 # = Number of Relevant Variables
n = 500 # = Number of observations

betas = (-1)^(1:k) # = Values for beta

sigma1 = genPositiveDefMat(p,"unifcorrmat")$Sigma # sigma1 violates irc

sigma2 = sigma1 # sigma2 satisfies irc
sigma2[(k+1):p,1:k]=0 # removing correlation among active and nonactive variables
sigma2[1:k,(k+1):p]=0

# Verify irrepresentable condition
irc1 = sort(abs(sigma1[(k+1):p,1:k] %*% solve(sigma1[1:k,1:k]) %*% sign(betas)))
irc2 = sort(abs(sigma2[(k+1):p,1:k] %*% solve(sigma2[1:k,1:k]) %*% sign(betas)))
c(max(irc1),max(irc2))
```

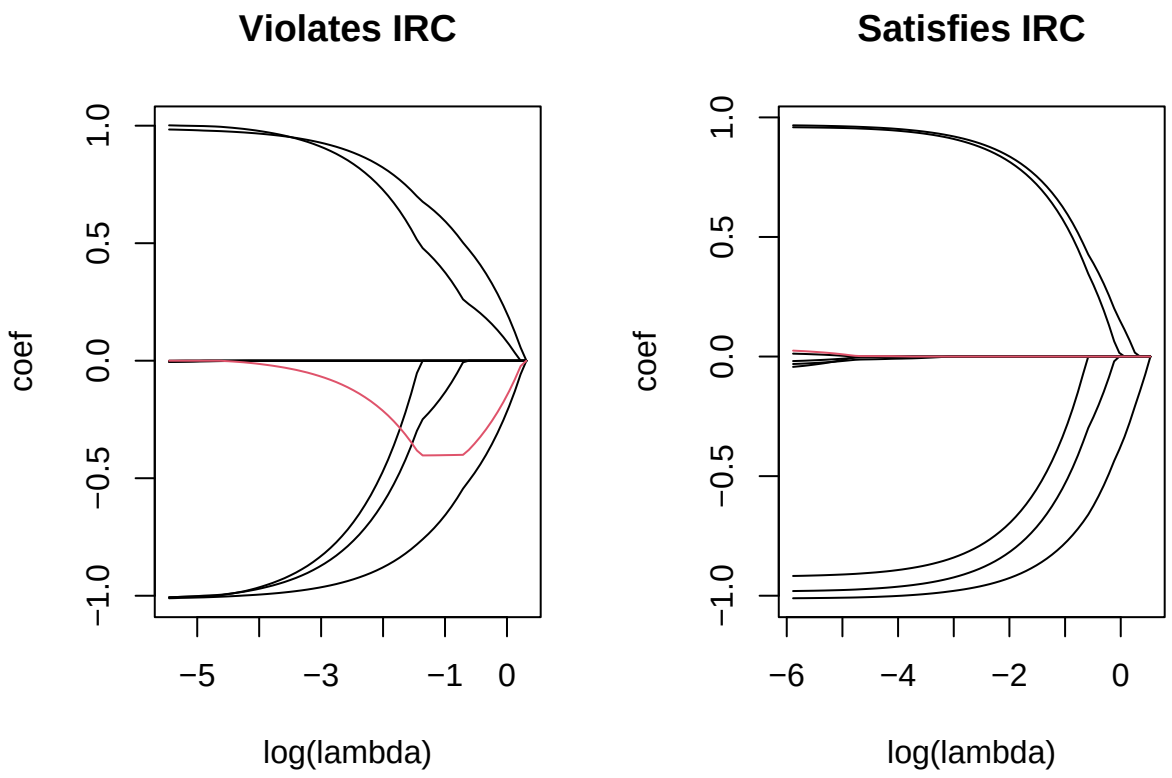
```
## [1] 3.222599 0.000000
```

```
# = Have a look at the correlation matrices
par(mfrow=c(1,2))
corrplot(cov2cor(sigma1))
corrplot(cov2cor(sigma2))
```



The first variance-covariance matrix does not satisfy IRC, while the second does. Let's check it "visually", by exploring the regularization paths!

```
X1 = rmvnorm(n,sigma = sigma1) # Variables violating IRC
X2 = rmvnorm(n,sigma = sigma2) # Variables satisfying IRC
e = rnorm(n) # Error from Standard Normal
y1 = X1[,1:k]%*%betas+e # Generate y for design 1
y2 = X2[,1:k]%*%betas+e # Generate y for design 2
lasso1 = glmnet(X1,y1,nlambda = 100) # Estimation for design 1
lasso2 = glmnet(X2,y2,nlambda = 100) # Estimation for design 2

# Regularization paths
par(mfrow=c(1,2))
l1=log(lasso1$lambda)
matplot(as.matrix(l1),t(coef(lasso1)[-1,]),type="l",lty=1,col=c(rep(1,9),2),ylab="coef",xlab="log(lambda
l2=log(lasso2$lambda)
matplot(as.matrix(l2),t(coef(lasso2)[-1,]),type="l",lty=1,col=c(rep(1,9),2),ylab="coef",xlab="log(lambda
```



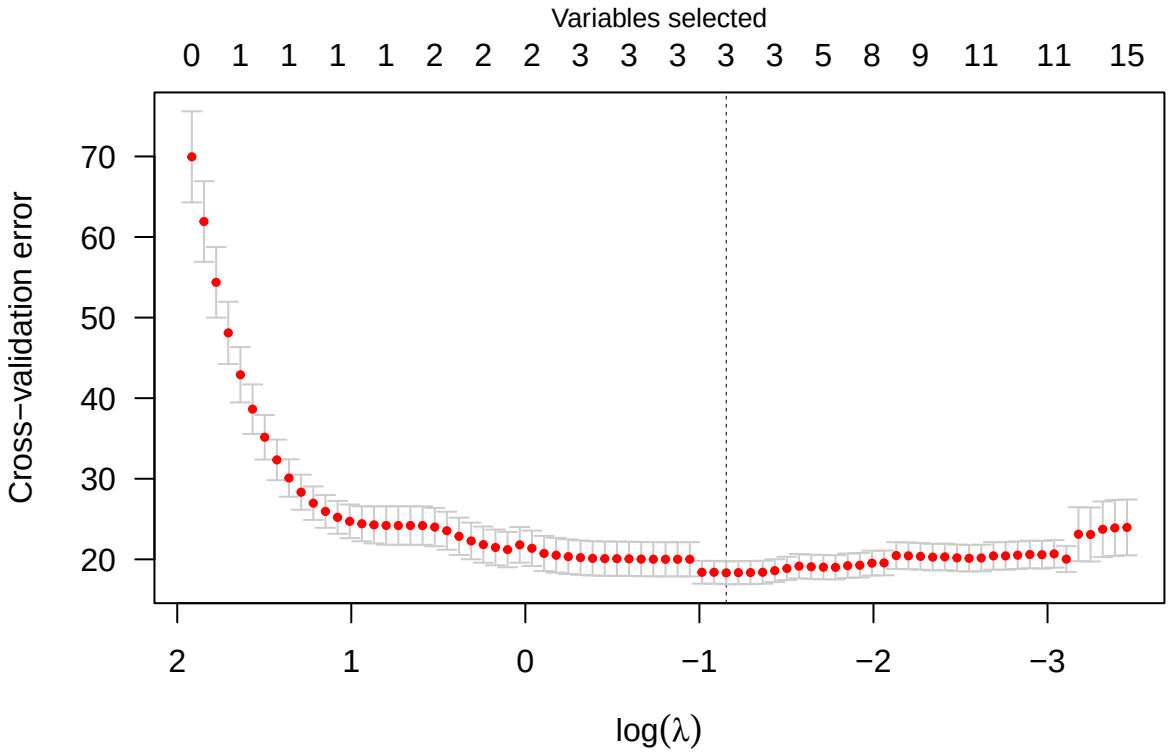
Nonconvex penalization methods

If you want to try nonconvex penalization methods, have a look at the **ncvreg** package! It has a very similar syntax than glmnet (check the paper here).

```
library(ncvreg)

dim(newx)
```

```
## [1] 252 182
```

```
cv_scad <- cv.ncvreg(as.matrix(newx), y_var)
plot(cv_scad)
```


Variables selected
0 1 1 1 1 2 2 2 3 3 3 3 3 5 8 9 11 11 15
Cross−validation error
70
60
50
40
30
20
2
1
0
−1
−2
−3
log(λ)