Feature Selection

S. Tonini, F. Chiaromonte (special thanks to J. Di Iorio, L. Insolia and L. Testa)

March 7th 2023

Contents

Introduction

Libraries

We are going to use:

- caret: Classification and Regression Training
- leaps: Regression Subset Selection

```
library(caret)  # statistical learning techniques
library(leaps)  # BSS
library(glmnet)
```

Data

We will use the **Body Fat dataset** (which is available in the Datasets folder of our course). See also the practicum material from last lecture.

The data concerns a sample of 252 men, and contains 15 different variables. We want to understand if we can reliably describe and predict body fat percentage on the basis of these variables, using regression. 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.

```
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 all other features aside from "Density". So we drop the "Density" column.

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

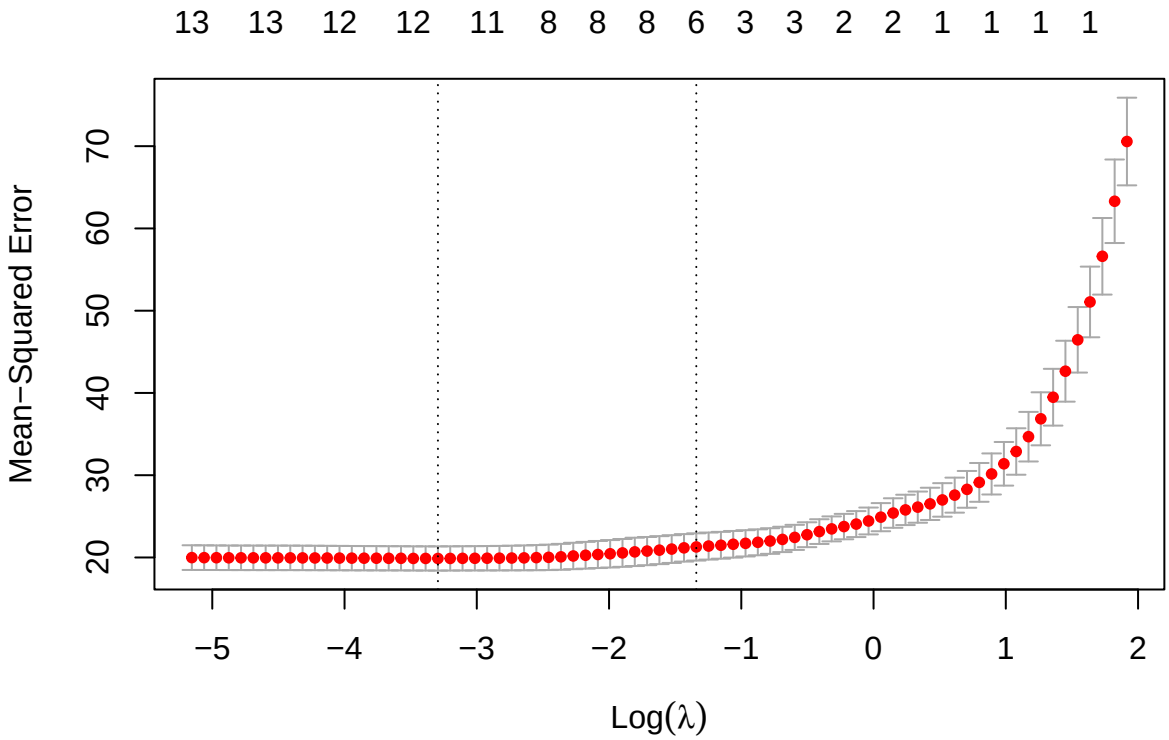
LASSO

We can start this practicum with one of the methods we discovered during the last lecture: the LASSO. As you have learnt, the LASSO implicitly performs feature selection.

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

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 0.03711    57   19.88 1.472      12
## 1se 0.26180    36   21.28 1.657       6
```

```
plot(cv_lasso)
```



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

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) -15.531418951
## Over45        0.008365606
## Weight       -0.085271056
## Height       -0.111951972
## NeckC        -0.402221616
## ChestC        .
## AbdomenC      0.966813664
## HipC         -0.171063869
## ThighC        0.077458115
## KneeC         0.050391923
## AnkleC        0.050726019
## BicepsC       0.152542394
## ForearmC      0.347466887
## WristC       -1.202707185
```

Best Subset Selection (Linear Regression)

We will use the **regsubsets()** function (part of the **leaps** library). It performs best subset selection by identifying the best model that contains a given number of predictors, where the notion of "best" is based on the in-sample RSS. No cross-validation is performed. The summary() command outputs the best set of variables for each model size.

```
regfit.full = regsubsets(SiriBF. ~ ., data = df,  nvmax = 13, method="exhaustive")
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(SiriBF. ~ ., data = df, nvmax = 13, method = "exhaustive")
## 13 Variables  (and intercept)
##          Forced in Forced out
## Over45       FALSE      FALSE
## Weight       FALSE      FALSE
## Height       FALSE      FALSE
## NeckC        FALSE      FALSE
## ChestC       FALSE      FALSE
## AbdomenC     FALSE      FALSE
## HipC         FALSE      FALSE
## ThighC       FALSE      FALSE
## KneeC        FALSE      FALSE
## AnkleC       FALSE      FALSE
## BicepsC      FALSE      FALSE
## ForearmC     FALSE      FALSE
## WristC       FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: exhaustive
##           Over45 Weight Height NeckC ChestC AbdomenC HipC ThighC KneeC AnkleC
## 1  ( 1 )  " "    " "    " "    " "   " "    "*"      " "  " "    " "   " "
## 2  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 3  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
```

```
## 4  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 5  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  " "    " "   " "
## 6  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  " "    " "   " "
## 7  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  "*"    " "   " "
## 8  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  "*"    " "   " "
## 9  ( 1 )  " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   " "
## 10  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   "*"
## 11  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    "*"   "*"
## 12  ( 1 ) " "    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
## 13  ( 1 ) "*"    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
##           BicepsC ForearmC WristC
## 1  ( 1 )  " "     " "      " "
## 2  ( 1 )  " "     " "      " "
## 3  ( 1 )  " "     " "      "*"
## 4  ( 1 )  " "     "*"      "*"
## 5  ( 1 )  " "     "*"      "*"
## 6  ( 1 )  "*"     "*"      "*"
## 7  ( 1 )  " "     "*"      "*"
## 8  ( 1 )  "*"     "*"      "*"
## 9  ( 1 )  "*"     "*"      "*"
## 10  ( 1 ) "*"     "*"      "*"
## 11  ( 1 ) "*"     "*"      "*"
## 12  ( 1 ) "*"     "*"      "*"
## 13  ( 1 ) "*"     "*"      "*"
```

The summary() function also returns R^2, RSS R^2_{adj}, C_p, and BIC. We can examine these to try to select the best overall model.

```
names(summary(regfit.full))
```

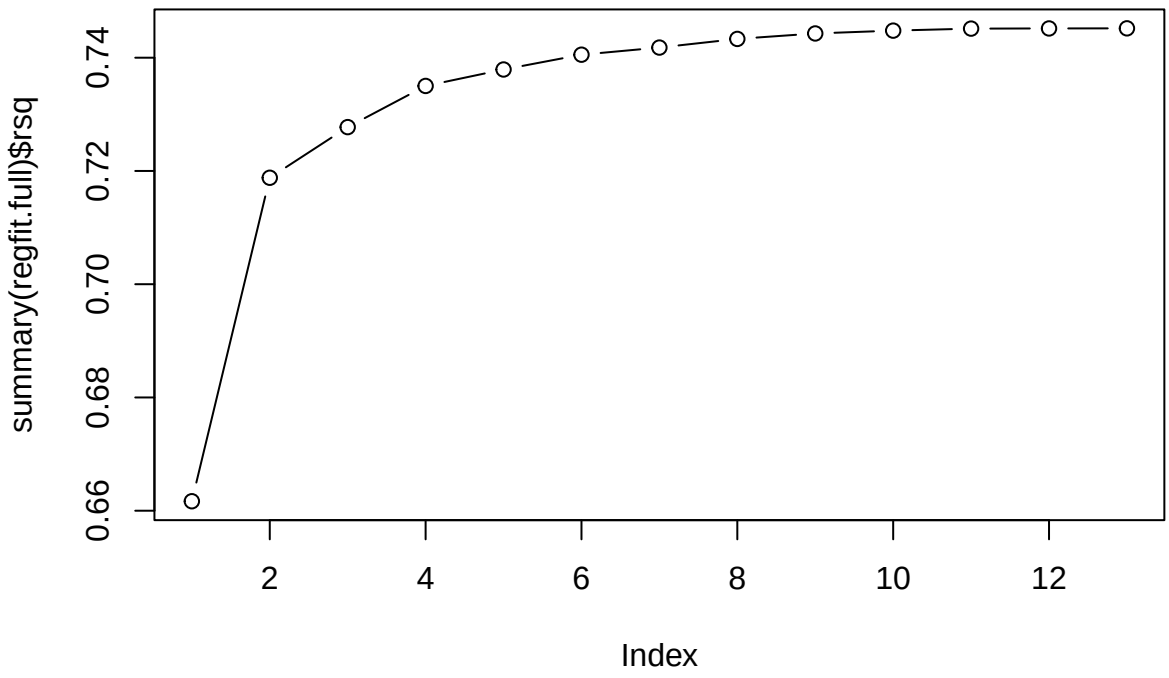
```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

As expected, the R^2 statistic increases monotonically as more variables are included into the model.

```
summary(regfit.full)$rsq
```

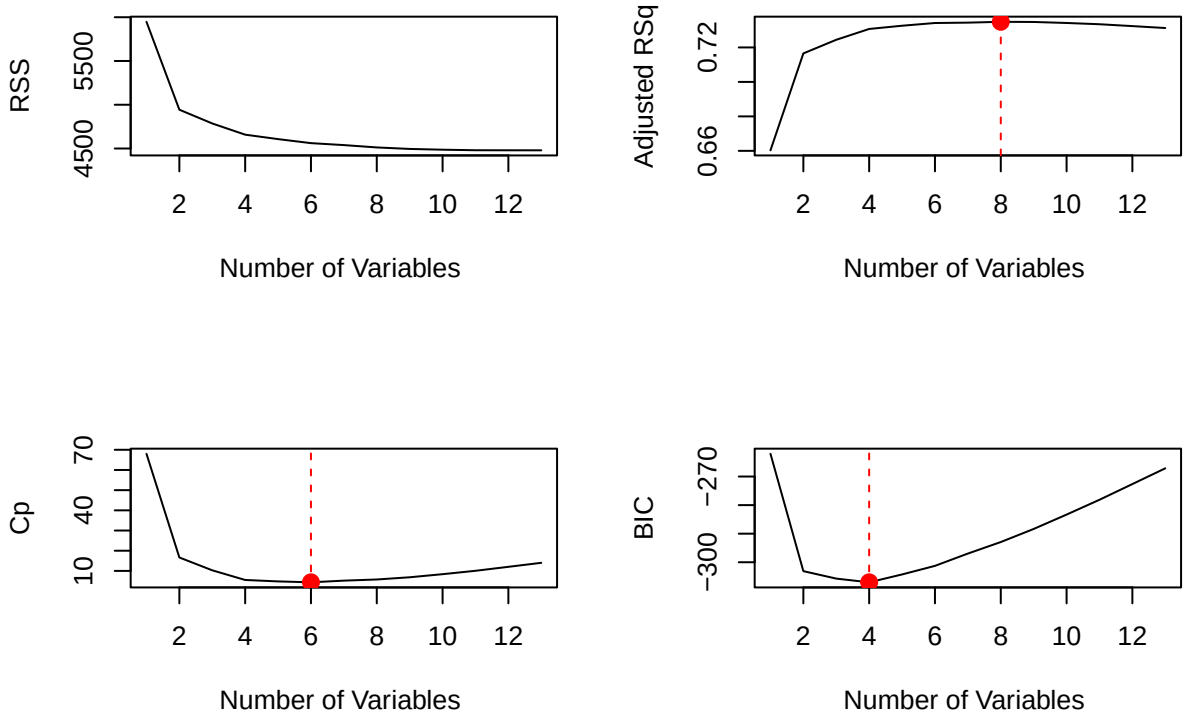
```
##  [1] 0.6616721 0.7187981 0.7277401 0.7350112 0.7379161 0.7405288 0.7417892
##  [8] 0.7433160 0.7442807 0.7447809 0.7451348 0.7451669 0.7451821
```

```
#plot rss
plot(summary(regfit.full)$rsq, type="b")
```



Let us plot also the other indexes.

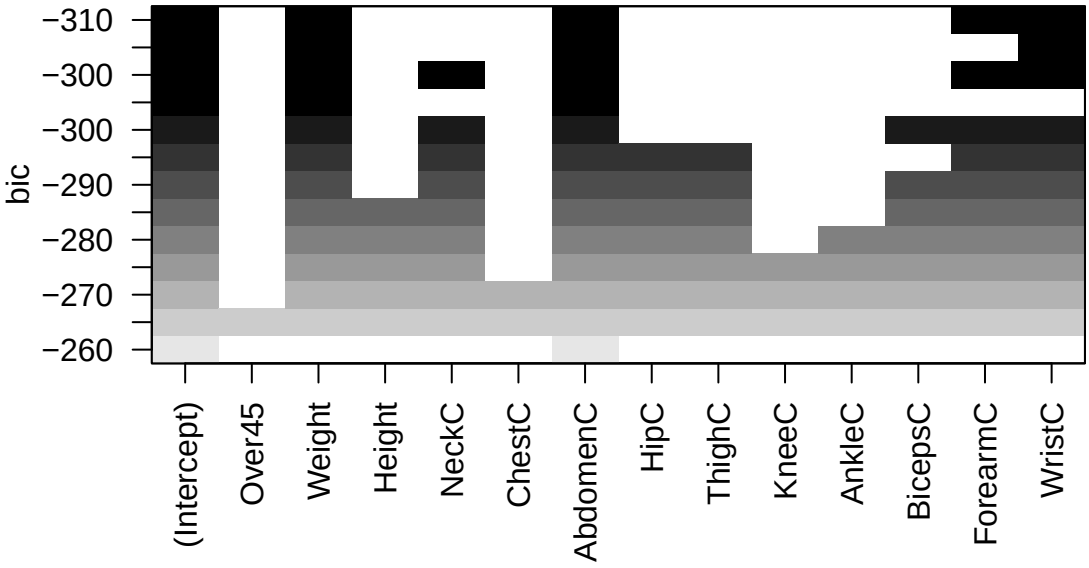
```
reg.summary <- summary(regfit.full)
par(mfrow=c(2,2))
plot(reg.summary$rss ,xlab="Number of Variables ",ylab="RSS",type="l")


plot(reg.summary$adjr2 ,xlab="Number of Variables ", ylab="Adjusted RSq",type="l")
max_adjr2 <- which.max(reg.summary$adjr2)
abline(v=max_adjr2, col="red", lty=2)
points(max_adjr2,reg.summary$adjr2[max_adjr2], col="red",cex=2,pch=20)

plot(reg.summary$cp ,xlab="Number of Variables ",ylab="Cp", type='l')
min_cp <- which.min(reg.summary$cp )
points(min_cp, reg.summary$cp[min_cp],col="red",cex=2,pch=20)
abline(v=min_cp, col="red", lty=2)

plot(reg.summary$bic ,xlab="Number of Variables ",ylab="BIC",type='l')
min_bic <- which.min(reg.summary$bic)
points(min_bic,reg.summary$bic[min_bic],col="red",cex=2,pch=20)
abline(v=min_bic, col="red", lty=2)
```



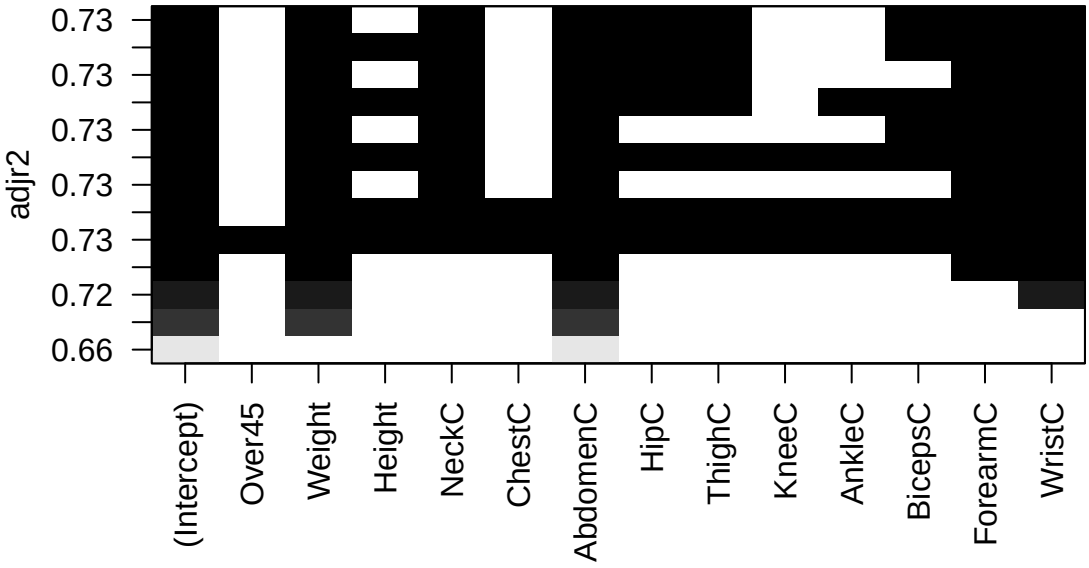
Note that the use of different criteria leads to different decisions.

Here is another visualization tool which is available in leaps and that you may find useful:

```
plot(regfit.full,scale="bic") #for "bic"
```




```
plot(regfit.full,scale="adjr2") #for "adjr2"
```



Tuning strategies

To select the best model we are going to use cross-validation.

For instance, we can start by splitting the observations into a training set and a test set (of size 80 vs 20%).

```
dim(df)
```

```
## [1] 252  14
```

```
set.seed(1)
train = sample(1:nrow(df), round(nrow(df)*0.8), rep=F)
test = which(!(1:nrow(df) %in% train))

length(test)
```

```
## [1] 50
```

```
length(train)
```

```
## [1] 202
```

Now, we apply regsubsets() on the training set:

```
regfit.best <- regsubsets(SiriBF. ~ ., data = df[train,],  nvmax = 13, method="exhaustive")
```

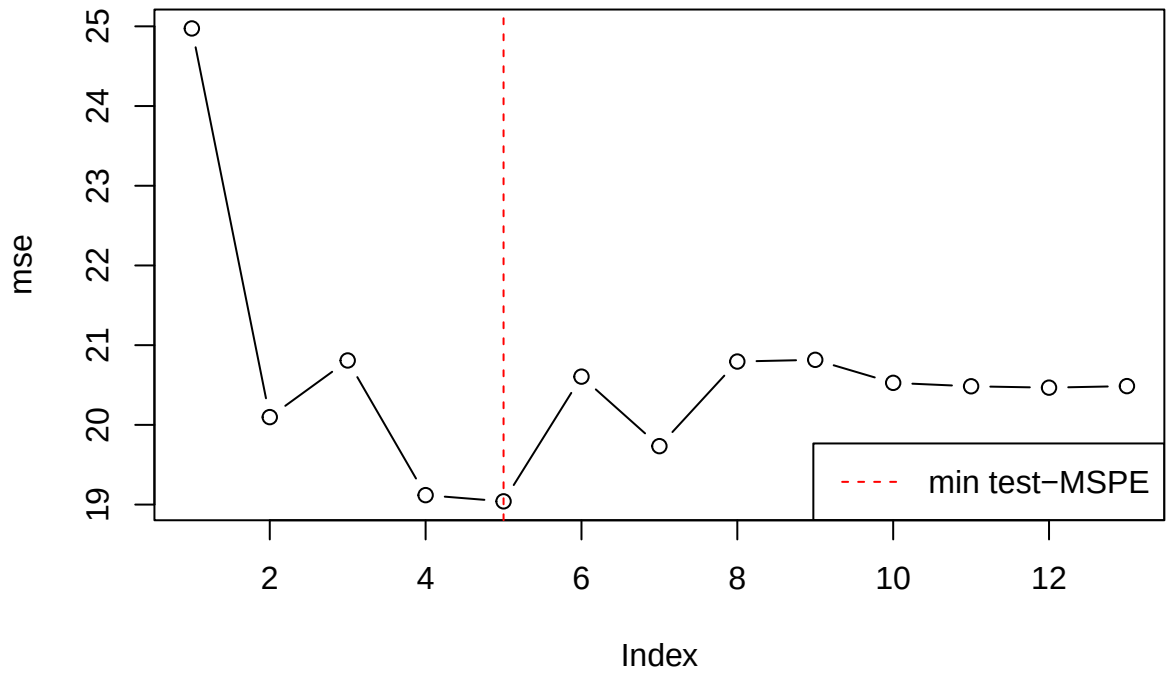
We then compute the test set error for the best model of each size. We need to compute all of this by hand, since the leaps library does not provide a predict function (as it is customary for R packages). Thus, we first make a model matrix containing test data:

```
test.mat = model.matrix(SiriBF. ~ ., data = df[test,])
dim(test.mat)
```

```
## [1] 50 14
```

Within a for loop, for each size i, we extract the coefficients for the best model of that size. We generate the predictions by multiplying them into the appropriate columns of the test model matrix. Then, we compute the MSPE on test data.

```
p = (ncol(df)-1) # number of predictors
mse <- rep(NA, p) # out-of-sample MSE
for(i in 1:p){
  coefi=coef(regfit.best,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  mse[i]=mean((df$SiriBF.[test]-pred)^2)
}

plot(mse, type='b')
abline(v=which.min(mse), col="red", lty=2)
legend(x = "bottomright",
       legend = "min test-MSPE",
       lty = 2,
       col = "red")
```



The best model is the one with 5 parameters (plus the intercept term). Let us see the coefficients:

```
coef(regfit.best, which.min(mse))
```

```
## (Intercept)      Weight       NeckC    AbdomenC    ForearmC      WristC
## -32.8474919  -0.1255555  -0.3901280   0.9896630   0.6028652  -1.0754292
```

We now perform best subset selection on the full data set and select the best 5 variables.

```
regfit.best.full <- regsubsets(SiriBF. ~ ., data=df ,nvmax=13)
coef(regfit.best.full, which.min(mse))
```

```
## (Intercept)      Weight       NeckC    AbdomenC    ForearmC      WristC
## -30.6535775  -0.1227999  -0.3656842   1.0078446   0.5270335  -1.2462846
```

Focusing on the full dataset, we do not necessarily retrieve the same 5 variables that were obtained from the training set alone. However, in this example we retrieve the same set of selected features.

```
rescompare <- rbind(coef(regfit.best, which.min(mse)),
                    coef(regfit.best.full, which.min(mse)))
rownames(rescompare) <- c("train", "full")
rescompare
```

```
##       (Intercept)     Weight      NeckC AbdomenC  ForearmC    WristC
## train   -32.84749 -0.1255555 -0.3901280 0.989663 0.6028652 -1.075429
## full    -30.65358 -0.1227999 -0.3656842 1.007845 0.5270335 -1.246285
```

We now consider a k-fold CV (with 10-folds). Let's create 10 folds through the caret package:

```
k = 10
set.seed(123)

# folds
folds <- createFolds(1:nrow(df), k = 10, list = TRUE, returnTrain = T)
fold <- matrix(NA, nrow(df), k)
for (i in 1:k) {
  fold[, i] <- (1:nrow(df) %in% folds[[i]])
}
head(fold, 10)
```

```
##        [,1] [,2]  [,3]  [,4]  [,5] [,6] [,7] [,8]  [,9] [,10]
##  [1,]  TRUE TRUE  TRUE  TRUE  TRUE TRUE TRUE TRUE  TRUE FALSE
##  [2,]  TRUE TRUE  TRUE  TRUE  TRUE TRUE TRUE TRUE FALSE  TRUE
##  [3,] FALSE TRUE  TRUE  TRUE  TRUE TRUE TRUE TRUE  TRUE  TRUE
##  [4,]  TRUE TRUE  TRUE  TRUE  TRUE TRUE TRUE TRUE  TRUE FALSE
##  [5,]  TRUE TRUE FALSE  TRUE  TRUE TRUE TRUE TRUE  TRUE  TRUE
##  [6,]  TRUE TRUE  TRUE FALSE  TRUE TRUE TRUE TRUE  TRUE  TRUE
##  [7,]  TRUE TRUE FALSE  TRUE  TRUE TRUE TRUE TRUE  TRUE  TRUE
##  [8,]  TRUE TRUE FALSE  TRUE  TRUE TRUE TRUE TRUE  TRUE  TRUE
##  [9,] FALSE TRUE  TRUE  TRUE  TRUE TRUE TRUE TRUE  TRUE  TRUE
## [10,]  TRUE TRUE  TRUE  TRUE FALSE TRUE TRUE TRUE  TRUE  TRUE
```

```
# initialize an empty matrix to contain test errors
cv.errors=matrix(NA, k, # num of folds
                 p,     # num of variables
                 dimnames=list(NULL, paste(1:p)))
```

Now we write a for loop that performs cross-validation. We make our predictions for each model size, compute the test errors on the appropriate subset, and store them in the appropriate slot in the matrix cv.errors. Considering the absence of a predict() function we create our own function where objects would be the result of regsubset(), newdata comprising the test set, and id as the number of parameters in the models obtained by regsubset().

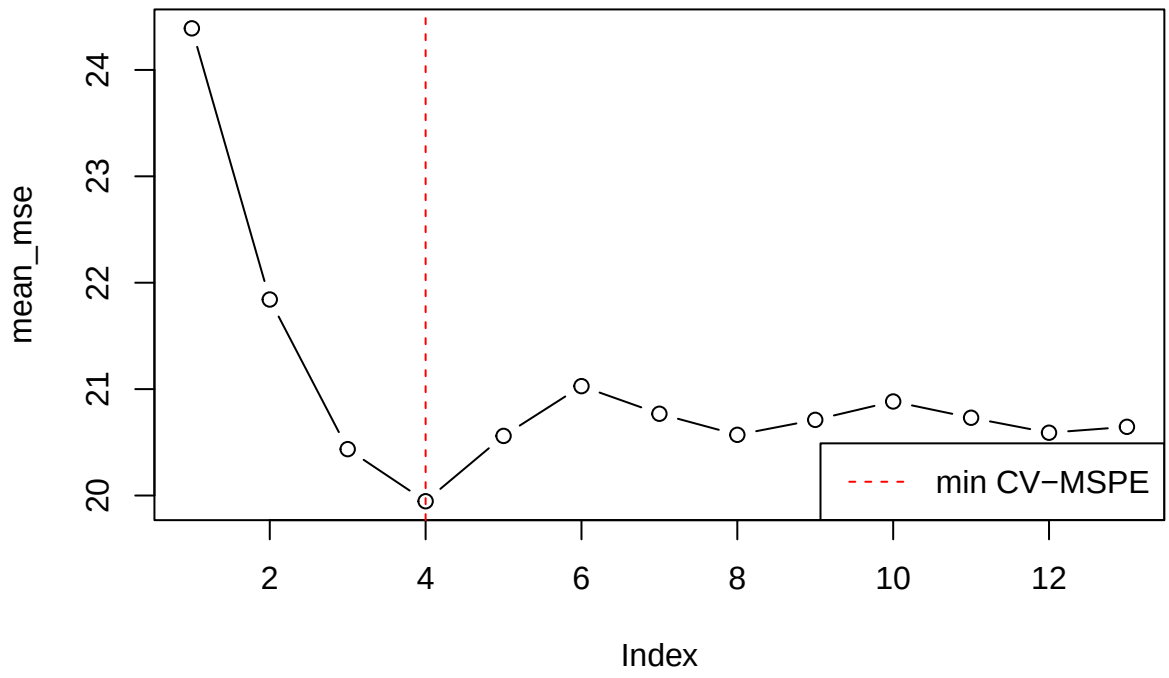
```r
predict.regsubsets = function (object, newdata, id, ...){
  form = as.formula(object$call[[2]])
  mat = model.matrix(form,newdata)
  coefi = coef(object ,id=id)
  xvars = names(coefi)
  mat[,xvars] %*% coefi
}
```

We are ready!

```r
# loop for each fold
for(j in 1:k){
  best.fit = regsubsets(SiriBF. ~ . , data=df[fold[,j], ], nvmax = p)

  # for each best model
  for (i in 1:p){
    pred = predict.regsubsets(best.fit, df[!fold[,j], ], id = i)
    cv.errors[j, i] = mean((df$SiriBF.[!fold[,j]] - pred)^2)
  }
}
```

The result is stored in the cv.errors matrix having on the rows the folds and on the columns the number of variables of the model. Each cell contains the MSPE. Now we compute column-wise averages.

```r
mean_mse <- colMeans(cv.errors)
plot(mean_mse, type='b')
abline(v=which.min(mean_mse), col="red", lty=2)
legend(x = "bottomright",
       legend = "min CV-MSPE",
       lty = 2,
       col = "red")
```



Using this approach we should retain 4 variables (plus the intercept):

```
reg.best  <- regsubsets (SiriBF. ~ ., data=df, nvmax=p)
coef(reg.best, which.min(mean_mse))
```

```
## (Intercept)      Weight    AbdomenC    ForearmC      WristC
## -34.8540743  -0.1356315   0.9957513   0.4729284  -1.5055620
```

Scalable BSS

Have a look at the **best-subset** package if you need to perform best subset selection on a large number of features.

```
# library(devtools)
# install_github(repo="ryantibs/best-subset", subdir="bestsubset")
```

Forward and Backward Stepwise Selection

The regsubsets() function can be used to perform forward stepwise or backward stepwise selection. In order to do so, we need to set the argument method="forward" or method="backward" (as opposed to "exhaustive").

```
regfit.fwd = regsubsets(SiriBF. ~. , data=df,nvmax=13, method ="forward")
regfit.bwd = regsubsets(SiriBF. ~. , data=df,nvmax=13, method ="backward")
summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(SiriBF. ~ ., data = df, nvmax = 13, method = "forward")
## 13 Variables  (and intercept)
```

```
##          Forced in Forced out
## Over45       FALSE      FALSE
## Weight       FALSE      FALSE
## Height       FALSE      FALSE
## NeckC        FALSE      FALSE
## ChestC       FALSE      FALSE
## AbdomenC     FALSE      FALSE
## HipC         FALSE      FALSE
## ThighC       FALSE      FALSE
## KneeC        FALSE      FALSE
## AnkleC       FALSE      FALSE
## BicepsC      FALSE      FALSE
## ForearmC     FALSE      FALSE
## WristC       FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: forward
##           Over45 Weight Height NeckC ChestC AbdomenC HipC ThighC KneeC AnkleC
## 1  ( 1 )  " "    " "    " "    " "   " "    "*"      " "  " "    " "   " "
## 2  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 3  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 4  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 5  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  " "    " "   " "
## 6  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  " "    " "   " "
## 7  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  " "    " "   " "
## 8  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  "*"    " "   " "
## 9  ( 1 )  " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   " "
## 10  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   "*"
## 11  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    "*"   "*"
## 12  ( 1 ) " "    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
## 13  ( 1 ) "*"    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
##           BicepsC ForearmC WristC
## 1  ( 1 )  " "     " "      " "
## 2  ( 1 )  " "     " "      " "
## 3  ( 1 )  " "     " "      "*"
## 4  ( 1 )  " "     "*"      "*"
## 5  ( 1 )  " "     "*"      "*"
## 6  ( 1 )  "*"     "*"      "*"
## 7  ( 1 )  "*"     "*"      "*"
## 8  ( 1 )  "*"     "*"      "*"
## 9  ( 1 )  "*"     "*"      "*"
## 10  ( 1 ) "*"     "*"      "*"
## 11  ( 1 ) "*"     "*"      "*"
## 12  ( 1 ) "*"     "*"      "*"
## 13  ( 1 ) "*"     "*"      "*"
```

```
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(SiriBF. ~ ., data = df, nvmax = 13, method = "backward")
## 13 Variables  (and intercept)
##          Forced in Forced out
## Over45       FALSE      FALSE
## Weight       FALSE      FALSE
## Height       FALSE      FALSE
## NeckC        FALSE      FALSE
```

```
## ChestC       FALSE      FALSE
## AbdomenC     FALSE      FALSE
## HipC         FALSE      FALSE
## ThighC       FALSE      FALSE
## KneeC        FALSE      FALSE
## AnkleC       FALSE      FALSE
## BicepsC      FALSE      FALSE
## ForearmC     FALSE      FALSE
## WristC       FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: backward
##           Over45 Weight Height NeckC ChestC AbdomenC HipC ThighC KneeC AnkleC
## 1  ( 1 )  " "    " "    " "    " "   " "    "*"      " "  " "    " "   " "
## 2  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 3  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 4  ( 1 )  " "    "*"    " "    " "   " "    "*"      " "  " "    " "   " "
## 5  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  " "    " "   " "
## 6  ( 1 )  " "    "*"    " "    "*"   " "    "*"      " "  "*"    " "   " "
## 7  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  "*"    " "   " "
## 8  ( 1 )  " "    "*"    " "    "*"   " "    "*"      "*"  "*"    " "   " "
## 9  ( 1 )  " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   " "
## 10  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    " "   "*"
## 11  ( 1 ) " "    "*"    "*"    "*"   " "    "*"      "*"  "*"    "*"   "*"
## 12  ( 1 ) " "    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
## 13  ( 1 ) "*"    "*"    "*"    "*"   "*"    "*"      "*"  "*"    "*"   "*"
##           BicepsC ForearmC WristC
## 1  ( 1 )  " "     " "      " "
## 2  ( 1 )  " "     " "      " "
## 3  ( 1 )  " "     " "      "*"
## 4  ( 1 )  " "     "*"      "*"
## 5  ( 1 )  " "     "*"      "*"
## 6  ( 1 )  " "     "*"      "*"
## 7  ( 1 )  " "     "*"      "*"
## 8  ( 1 )  "*"     "*"      "*"
## 9  ( 1 )  "*"     "*"      "*"
## 10  ( 1 ) "*"     "*"      "*"
## 11  ( 1 ) "*"     "*"      "*"
## 12  ( 1 ) "*"     "*"      "*"
## 13  ( 1 ) "*"     "*"      "*"
```

We can notice that the best k-variables models can be different according to the stepwise procedure.

GLMs

In the case of GLM (see, for instance, logistic regression), we can use the **bestglm** library. It performs best subset selection for GLMs and computes AIC, BIC, EBIC, BICq or Cross-Validation. It also calls the leaps library when performing linear regression.

See also the **glmulti** library.