Computer lab 2

Emil K Svensson

16 November, 2016

Contents

Assignment 1

1.1

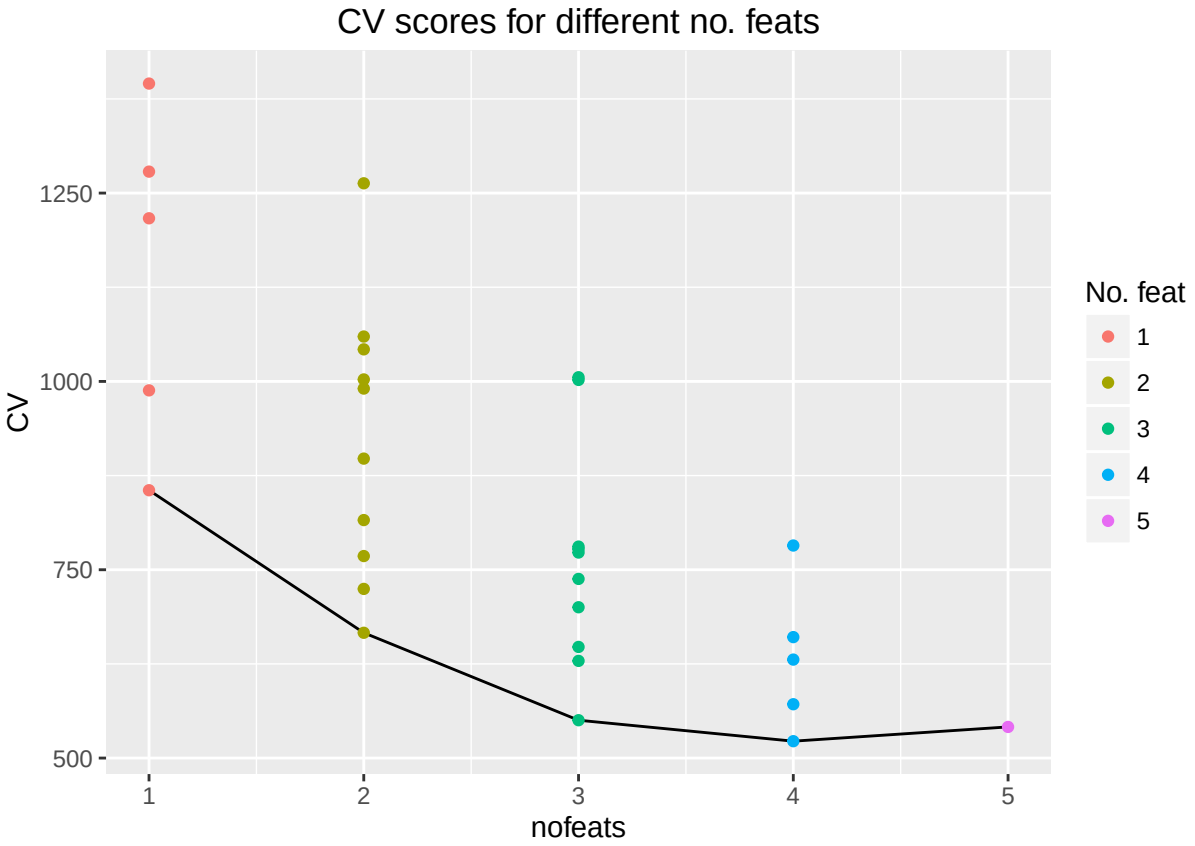
```
## 1.1

myLM <- function(Y, X, Nfolds){
  library(ggplot2)
  # DEFINING THE FUNCTION LINREG, FOR FITTING LINEAR MODELS

  linreg<-function(formula,data){
    formula <- formula(formula)
    des.mat <- model.matrix(formula , data) #Extracts the model matrix
    dep.var <- all.vars(formula)[1]         #Extracts the name of the y-variable
    dep.var <- as.matrix(data[dep.var])     #Extracts the data of the y-variable
    # and overwrites it with the data-colum

    #Calculating the beta coeffs. (X' %*% X)^-1 %*% X' %*% y
    beta.hat <- solve( t(des.mat) %*% des.mat )  %*% t(des.mat) %*% dep.var

    # Calculating the y-hat  , y_hat = X %*% beta_hat
    y.hat <- des.mat %*% beta.hat

    #Calculating the residuals e= y- y_hat
    res.err <- dep.var - y.hat

    l<-list( beta.hat = beta.hat, y.hat = y.hat, res.err = res.err)
    return(l)
  }


  #GENERATING ALL POSSIBLE PERMUTATIONS OF MODELS

  #Get the colnames for the X-variables
  q<-rep(paste0("X",c(1:5)))

  #Merge the data in to one data set and naming the columns
  myData <- cbind(Y,X)
  colnames(myData)<- c("Y",q)

  #Generating all possible combinations
  myComb<-sapply(c(1:5), FUN = combn, x = q )
  #Creating the vector that will hold all formulas
  myformula <- c(myComb[[1]])

  #Extracting the combinations of 2 and 3 X-variables and adding a + between them
  for (i in 2:3){
    for (j in 1:10){
      myformula[length(myformula)+1]<-(paste(myComb[[i]][,j],collapse = " + "))
    }
  }

  #Heres two rows that could replace the above for-loop
```

```
 #sapply(2:3, FUN = function(i) sapply(1:10, FUN = function(j)
 #paste(myComb[[i]][,j], collapse = " + " ) ) )


 #Extracting the combinations of 4 and 5 X-variables and adding a + between them
 #This is basicly a loop for the 4 combinations
 myformula <-c(myformula ,
               sapply(1:5, FUN =function(X)
                 paste(myComb[[4]][,X],collapse = " + " )
                 ),  paste(myComb[[5]],collapse = " + ")
               )

myformula<- paste("Y","~",myformula)

#### SPLITTING AND SUBSETING DATA IN TO K FOLDS

#calculatin no. rows
noobs<-dim(myData)[1]
K <- Nfolds

#Use sample to randomly draw the indexes of the dataset
#and reorder the data with them in a random manner
set.seed(12345)
myData<-myData[sample(noobs),]

#Create K equal indexes that are added to the data.
cut(1:noobs,breaks=K,labels = FALSE)

myData$index <- cut(1:noobs,breaks=K,labels = FALSE)


#init a counting vector "o" useed to loop in to the data.frame "linearModels"
#and a combination index "dataKombs" used for subseting the different datasets
#used for fitting models
o <- 1
linearModels<-data.frame(CV=1,model="text",nofeats=1,stringsAsFactors = FALSE)
dataKombs <- combn(1:K,K-1)
for (m in 1:length(myformula)){
for (l in (1:K)){

  #the data of the K-folds used for the model estimation
  data<-subset(myData, myData$index %in% dataKombs[,l] )


  #the fold that was left out in the model estimation
  predmatrix<-model.matrix(formula(myformula[m]),
            subset(myData, !(myData$index %in% dataKombs[,l] )))



  #Calculating the CV score for each model. sum((Y - Y(hat))^2)
  CV<-sum(
    (
```

```
       #this is the observed Y for the left out fold.
       subset(myData, !(myData$index %in% dataKombs[,1] ))[,1] -
        #predmatrix description above.
        predmatrix %*%
        #the estimated beta-hats.
        linreg(formula = myformula[m], data = data)$beta.hat
       )^2
     )

   #inserting the results in to the linearModels data.frame
   linearModels[o,] <- c(CV,myformula[m],ncol(predmatrix) - 1)
   o <- o + 1
    }
 }
 #reforming data to numeric again.
 linearModels[,1] <- as.numeric( linearModels[,1])
 linearModels[,3] <- as.numeric( linearModels[,3])

 #The mean for the different models, each model is estimadet K times
 plotdata<-suppressWarnings(aggregate(linearModels,by = list(linearModels$model),FUN= mean)[,-3])

 #renaming a column to ease plotting
 colnames(plotdata)[1] <- "Model"

 #plotting
 engr<-ggplot() +
   geom_line(data = aggregate(plotdata,list(plotdata$nofeats),FUN = min)[,c(3,4)],aes(x=nofeats,y=CV)) +
   geom_point(data = plotdata,aes(x=nofeats,y=CV,col = factor(nofeats)) ) +
       labs(title = "CV scores for different no. feats",color = "No. feat")


#displays the plot
plot(engr)

 #Returns the models with the lowest average CV-score
 return( plotdata[min(plotdata$CV) == plotdata$CV,c(1,3,2)])

}
```

1.2

```
##1.2

myLM(Y = swiss[,1],
     X = swiss[ , 2:ncol(swiss) ],
     Nfolds = 5 )
```




```
##                     Model nofeats       CV
## 12 Y ~ X1 + X3 + X4 + X5       4 522.4022
```

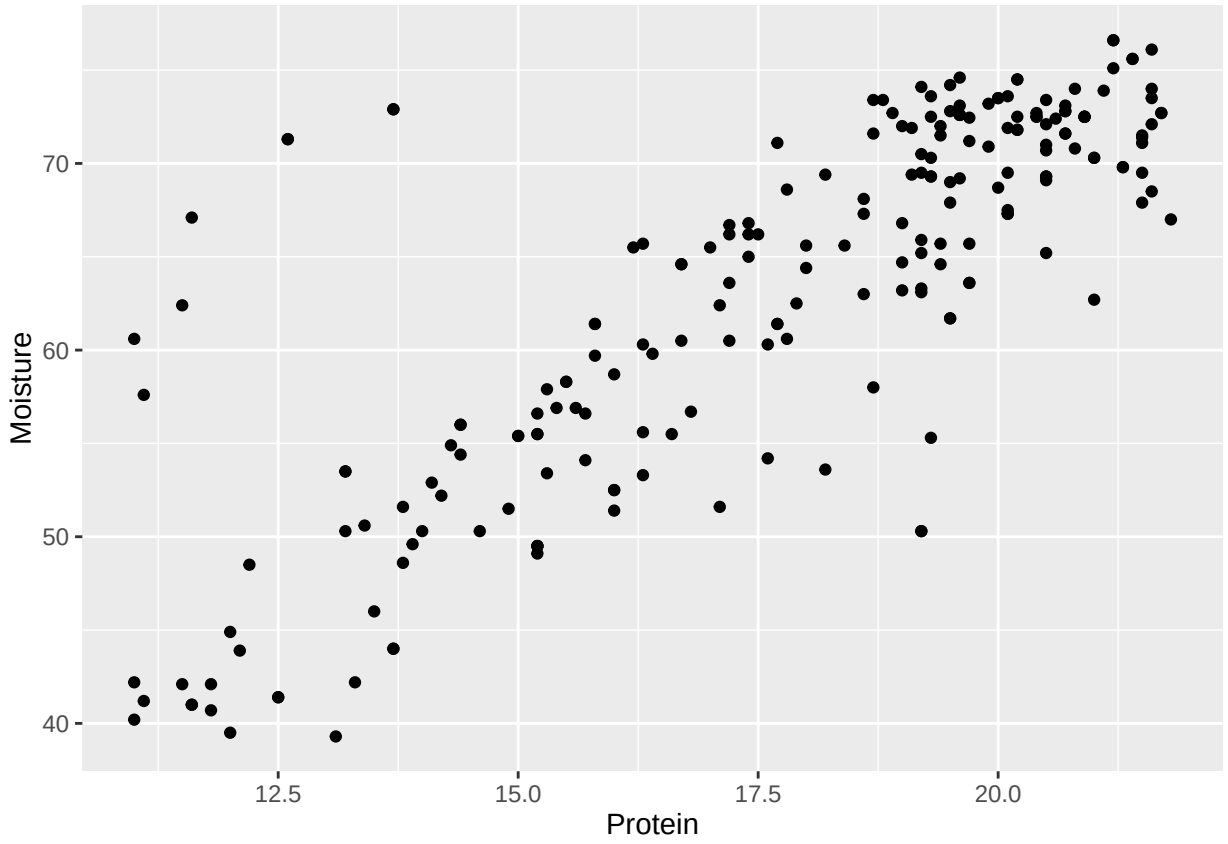
The CV-plot shows the cross validation-score (CV-score) for each of the models, each score corresponds to a model fitted against the response. When no. features (X-variables) is one each dot corresponds to the CV-score that is the average SSE of K (five in this case) models with one variable fitted against the respond. The plot shows how both the variation and mean decreases when the number of features increases. The lowest variation and mean for the CV-score (disregarding no. feats = 5) is models with 4 features. The three best models are one with 3, 4 and 5 features whereas the model with 4 features has the lowest CV-score and the most appropriate model.

The best subset to predict Fertility is Agriculture, Education, Catholic and Infant Mortality. It seems reasonable that these have impact on the Fertility rate as Agriculture, Education and Infant Mortality probably serves as good indicators for well-being in the town. The share of Catholics in the town could be explained by their belief that various methods of birth-control is some kind of a sin or that they don't belive in sex before marrige. The percentage of draftees receiving highest mark on army examination is the one variable that is left out and it doesn't seem weird, as it doesn't have any direct empirical connections to the fertility rate.

Assignment 2

2.1

```
## 2.1

setwd("/Users/EmilsHem/Documents/732A95/lab2")
TC<- read.csv2("tecator.csv")

ggplot(data = TC, aes(x=Protein,Moisture)) + geom_point()
```



A linear model with Protein as a predictor would be a good fit to the variable Moisture. Except for some outliers to the left of the plot and some low observations for Moisture when Protein is observed between 17 - 20.

2.2

$$p(M_i) = p(y|\mathbf{w}, x) = N(w_0 + \sum_1^i (x_1^i w_i), \sigma^2)$$

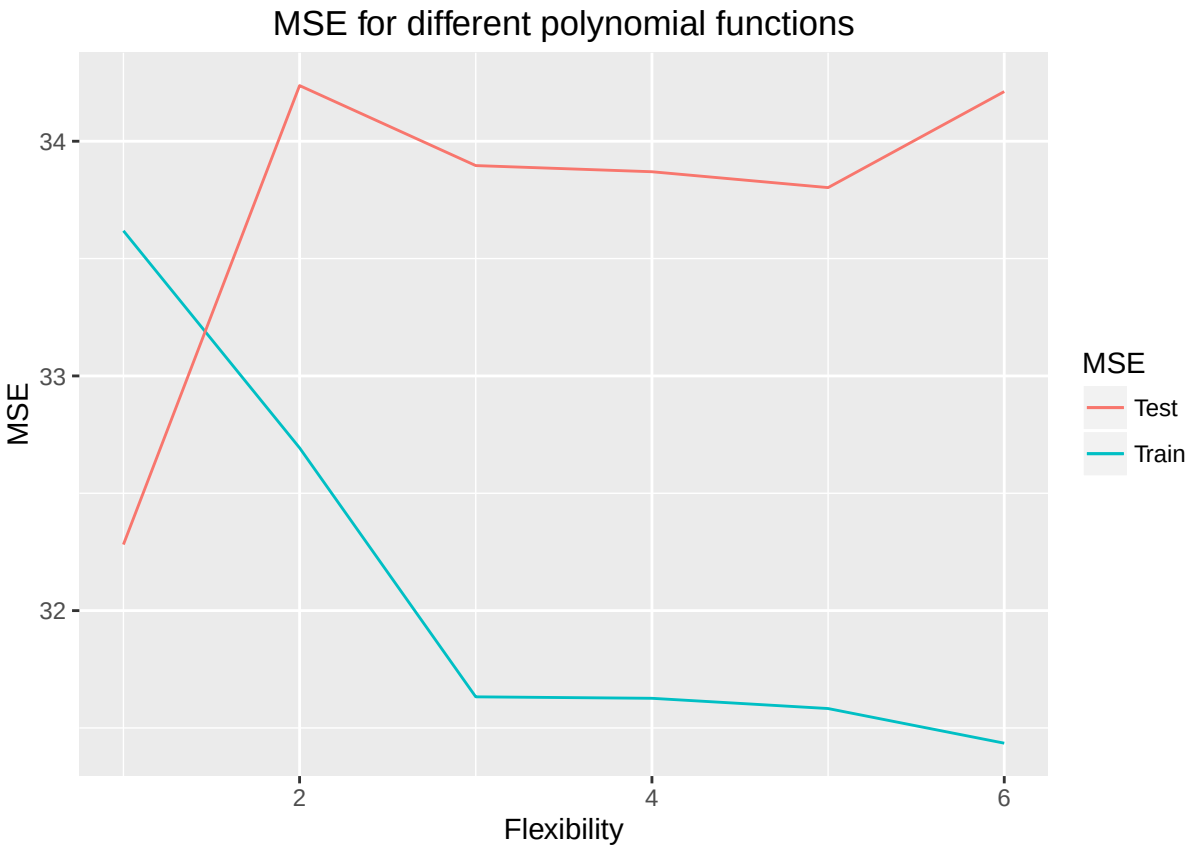
Where sigma-squared can be written as follows for each M_i

$$\epsilon \sim N(0, \sigma^2)$$

The σ^2 is the expected variance. As long as the normality assumption (where the error terms has a mean of zero and a variance of sigma square) holds it is appropriate to assume that the minimized MSE from the training data as it is an unbiased estimator. Although in reality the model with the lowest MSE isn't the most appropriate due to over fitting of data.

2.3

```
## 2.3

set.seed(12345)
obs   <- dim(TC)[1]
id    <- sample(1:obs, floor(obs*0.5))
train <- TC[id,]
valid <- TC[-id,]


  dataMSE <- data.frame(trainMSE = 1, testMSE = 1, noPoly = 1)
  i <-1

for (modell in 1:6) {

  linj<-lm(formula(paste("Moisture ~",paste("I(Protein^",1:modell,")", collapse = " + "))),data = train
  trainMSE<-mean(linj$residuals^2)
  testMSE<-mean( (predict(linj,newdata = valid) - valid$Moisture)^2)
  dataMSE[i,] <- c(trainMSE,testMSE,i)
  i <- i + 1

}

 aplot<- ggplot(data = dataMSE) +
    geom_line(aes(x = noPoly ,y = trainMSE, col = "Train")) +
    geom_line(aes(x = noPoly ,y = testMSE, col = "Test")) +
    labs(title="MSE for different polynomial functions",
         color ="MSE", x="Flexibility", y = "MSE")
  plot(aplot)
```



In the beginning of the plot the training data has a higher error rate than the test data which is probably a coincidence where the outliers were mostly placed in the traning data set. But it definitly seems like a linear model is most apropriate as it has the lowest MSE for both test and traning set.

When the training MSE decreases rapidly when the flexibility (no. parameters) increases. At the same time the MSE for increases. This is a statment of the overfitting as the traning model says it get better (lower MSE) but in reality it only improves the fit for the data given and not for any new observations.

2.4

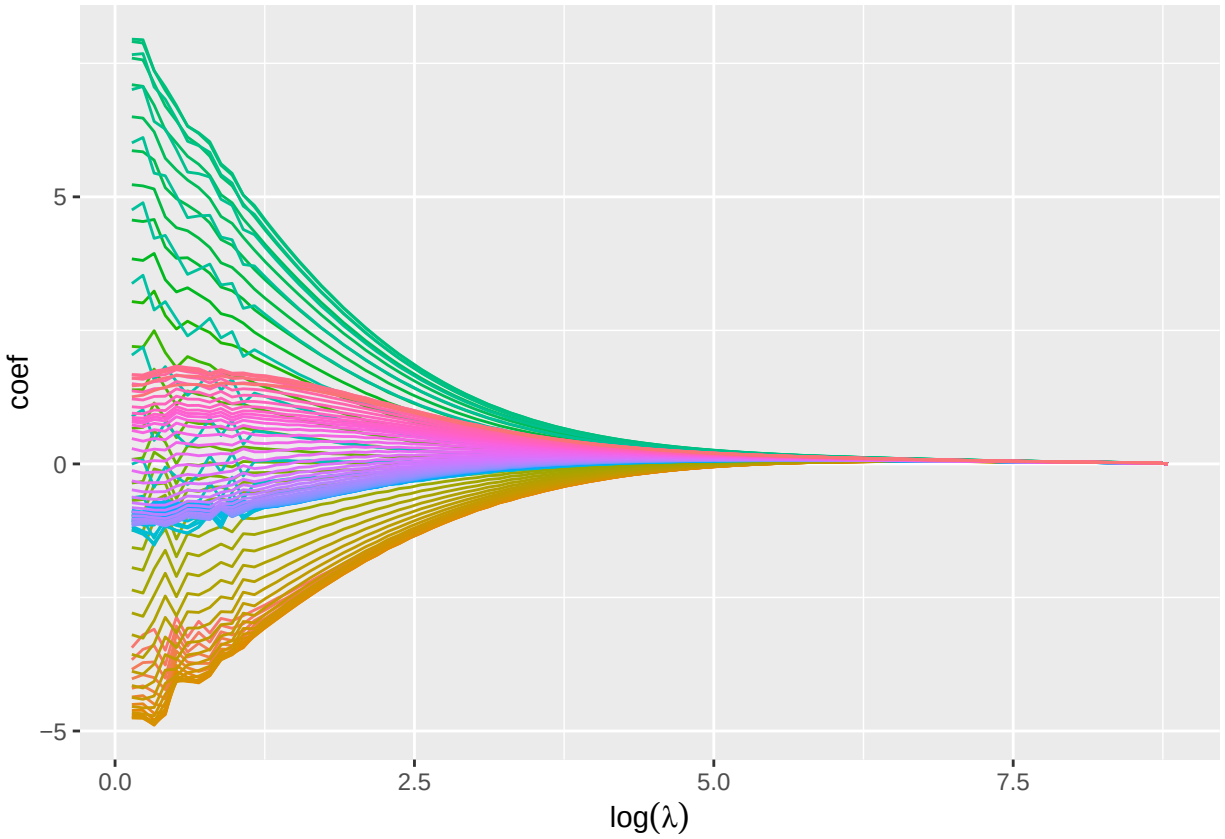
```
##2.4

library(MASS)
varSelect<-lm(Fat ~ . , data = TC[,2:102])
stepRes<-stepAIC(varSelect,trace = FALSE)
cat(length(stepRes$coefficients) - 1)
```

```
## 63
```

The stepAIC variable selection function kept 63 of the 100, in total it removed 37 of the variables.

2.5

```
## 2.5

library(glmnet)
library(reshape2)
wideWeg <- glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 0)
mycoeffs <- coefficients(wideWeg)[-1,]
colnames(mycoeffs) <- wideWeg$lambda

myPlot<-melt(as.matrix(mycoeffs),id = rownames,c("lambda","coef"))

colnames(myPlot) <- c("features","lambda","coef")


ggplot(myPlot,aes(x=log(lambda),y = coef,color = features)) + geom_line(show.legend = FALSE) + labs(x =
```

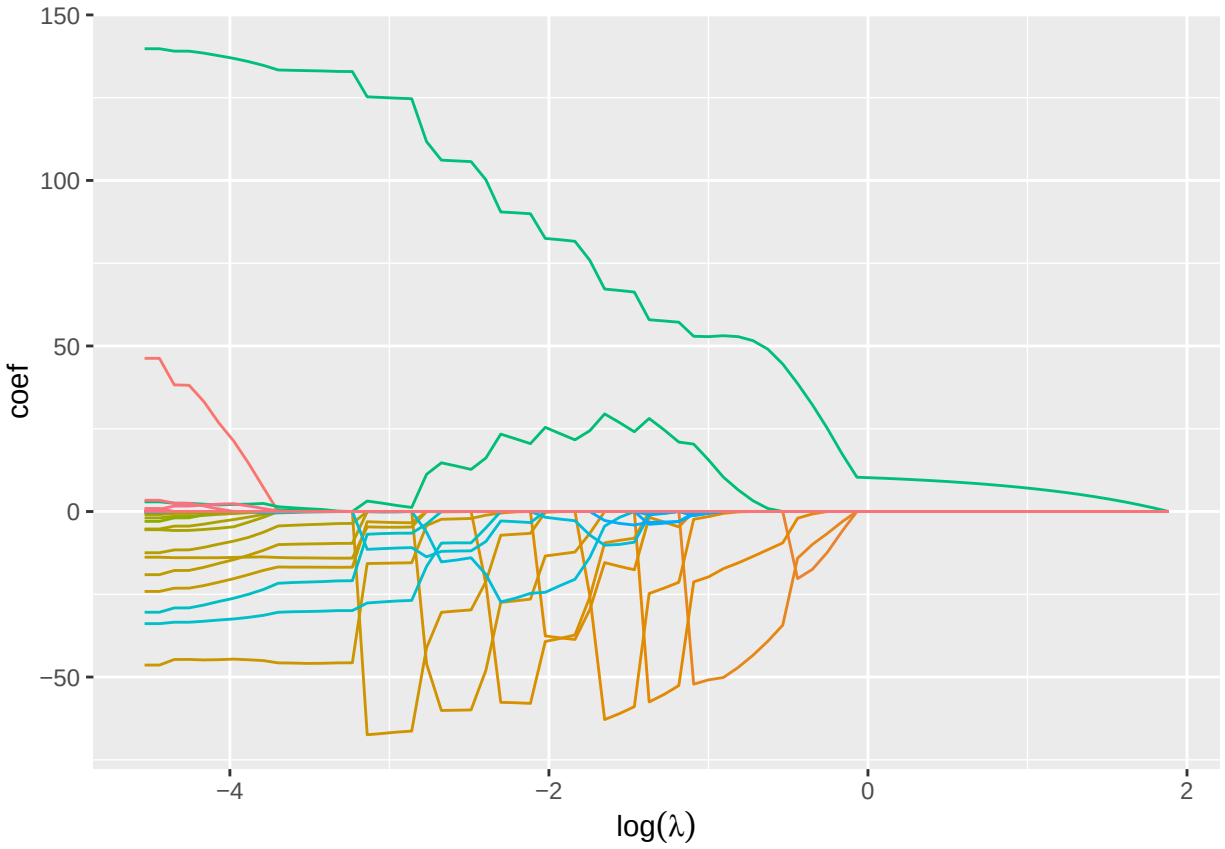


In this plot each line represents a variable ChannelX and how these are penalized by the lambda-value. When log(λ) increases the more each of the variables get penalized and the coefficients move closer to zero depending on it's value to the model. When a line is further away from the line where y = 0 the more impact to the model it brings. So the green lines at the top of the lines has the most relevance in the plot.

Around $log(lambda) = 1.25$ the irregular patterns in the lines seem to stop and assume a more smoothed pattern. At a log(λ) value around 5 almost all coefficients have converged at near zero and have no or little effect to the model.

2.6

```
## 2.6

lassoReg <- glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 1)
mycoeffs <- coefficients(lassoReg)[-1,]
colnames(mycoeffs) <- lassoReg$lambda

myLasso<-melt(as.matrix(mycoeffs),id = rownames,c("lambda","coef"))

colnames(myLasso) <- c("features","lambda","coef")


ggplot(myLasso,aes(x=log(lambda),y = coef,color = features)) +
  geom_line(show.legend = FALSE) + labs(x = expression(log(lambda)) )
```



As in the previous plot a line closer to zero mean a less useful model. A difference here is that the Lasso-model actually zero and removed and not kept very close to zero as in the Ridge-regression. This gives Lasso-models in general easier enterpretations as the number of variables are kept down.

One variable seem to have a large impact regardless of the $\log(\lambda)$ value, it's the one with highest values on coef.

When $\log(\lambda)$ is between -5 and -3 there are a number of variables that still are non-zero in the model. After that the number of variables decrease but some variables get a big importance boost at various rates of $\log(\lambda)$ which is shown by the up and down pattern in the mustard-yellow and orange lines. When $\log(\lambda)$ is 2 every variable except the green line has lost it influence on the model and their coefficients are equal to zero.

Compared to the Ridge regression the Lasso model seem be a bit more nuanced in its way of selecting variables that have impact on the model. The Lasso also don't remove all variables importance when lambda gets higher.

2.7

```
## 2.7
set.seed(12345)
cvLASSO<-cv.glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 1)

nocoeff<-matrix(coef(cvLASSO,s ="lambda.min"))
nocoeff[nocoeff != 0]
```

```
##  [1]  2.975525e+01 -4.588819e+01 -1.679962e+01 -1.400883e+01 -9.799177e+00
##  [6] -3.972331e+00 -2.241199e-05  2.552947e-01  8.459322e-01  1.331640e+02
## [11] -4.737202e-03 -1.452002e-01 -2.130415e+01 -3.020948e+01 -4.986143e-02
```

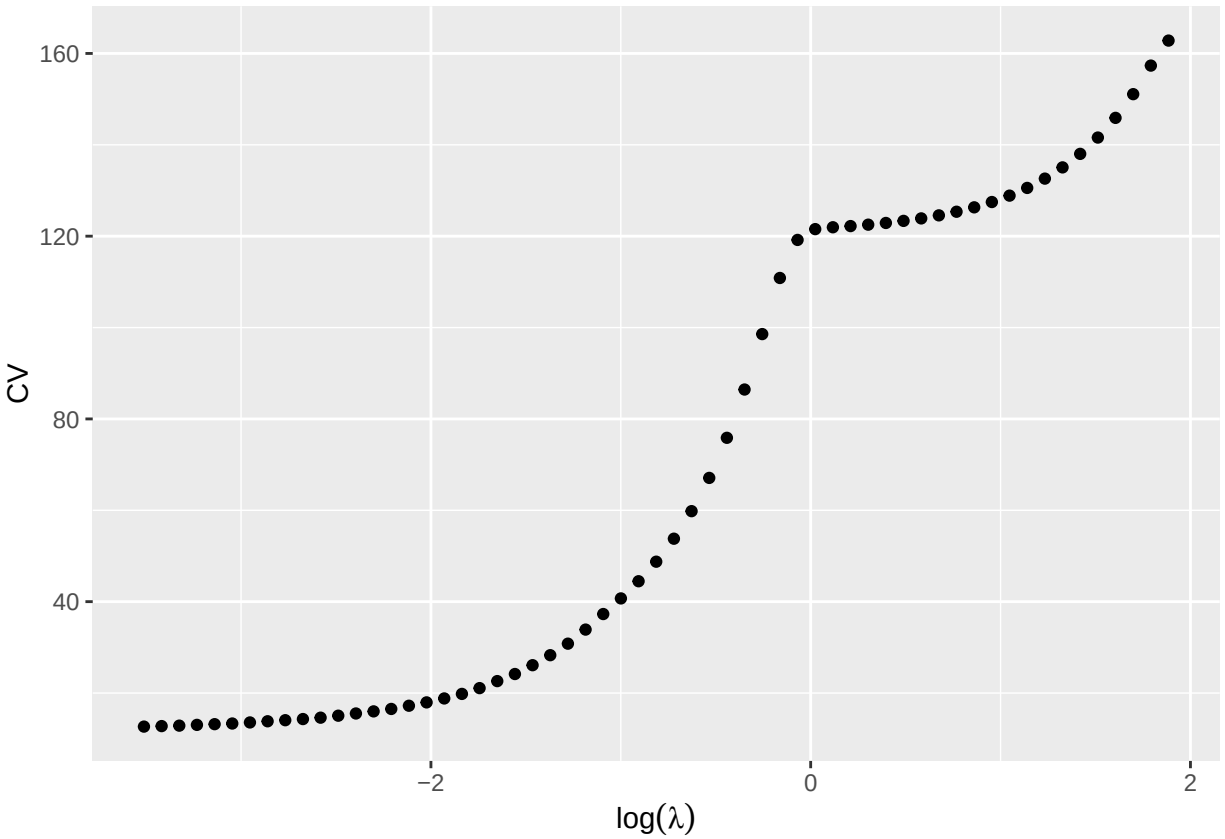
```
cat(paste("The no. of variables included in the optimal model was: ",length(nocoeff[nocoeff != 0]) - 1))
```

```
## The no. of variables included in the optimal model was:  14
```

```
cat(paste("The lambda of the optimal model was: ",cvLASSO$lambda.min))#antalet coeff exklusive intercep
```

```
## The lambda of the optimal model was:  0.0298560506070932
```

```
plotLasso<- data.frame(CV=cvLASSO$cvm,lambda= cvLASSO$lambda)

ggplot(data=plotLasso,aes(x = log(lambda), y = CV)) + geom_point() +
  labs(x = expression(log(lambda)) )
```



The CV score seems to increase as the $\log(\lambda)$ increases. So as we increase lambda we increase the MSE which would imply that we remove information (i.e. under fit) as we increase lambda. In this case when $\log(\lambda)$ reaches 0 it can not remove any more variables as there is only one left and instead start decreasing the last coefficient until it can't go any lower.

2.8

The Cross Validation in Lasso has 12.65335 a MSE for the optimal model and uses only 14 parameters compared to the stepAIC which removed 36 variables. The LASSO-model seem to penalize complex models in a better way than the stepAIC which stops with a model one could assume still is a too complex model to be considerd a good fit.

Code

```
knitr::opts_chunk$set(echo = TRUE)
## 1.1

myLM <- function(Y, X, Nfolds){
  library(ggplot2)
  # DEFINING THE FUNCTION LINREG, FOR FITTING LINEAR MODELS

  linreg<-function(formula,data){
    formula <- formula(formula)
    des.mat <- model.matrix(formula , data) #Extracts the model matrix
    dep.var <- all.vars(formula)[1]         #Extracts the name of the y-variable
    dep.var <- as.matrix(data[dep.var])     #Extracts the data of the y-variable
    # and overwrites it with the data-colum

    #Calculating the beta coeffs. (X' %*% X)^-1 %*% X' %*% y
    beta.hat <- solve( t(des.mat) %*% des.mat )  %*% t(des.mat) %*% dep.var

    # Calculating the y-hat  , y_hat = X %*% beta_hat
    y.hat <- des.mat %*% beta.hat

    #Calculating the residuals e= y- y_hat
    res.err <- dep.var - y.hat

    l<-list( beta.hat = beta.hat, y.hat = y.hat, res.err = res.err)
    return(l)
  }


  #GENERATING ALL POSSIBLE PERMUTATIONS OF MODELS

  #Get the colnames for the X-variables
  q<-rep(paste0("X",c(1:5)))

  #Merge the data in to one data set and naming the columns
  myData <- cbind(Y,X)
  colnames(myData)<- c("Y",q)

  #Generating all possible combinations
  myComb<-sapply(c(1:5), FUN = combn, x = q )
  #Creating the vector that will hold all formulas
  myformula <- c(myComb[[1]])

  #Extracting the combinations of 2 and 3 X-variables and adding a + between them
  for (i in 2:3){
    for (j in 1:10){
      myformula[length(myformula)+1]<-(paste(myComb[[i]][,j],collapse = " + "))
    }
  }

  #Heres two rows that could replace the above for-loop
  #sapply(2:3, FUN = function(i) sapply(1:10, FUN = function(j)
```

```
 #paste(myComb[[i]][,j], collapse = " + " ) ) )


 #Extracting the combinations of 4 and 5 X-variables and adding a + between them
 #This is basicly a loop for the 4 combinations
 myformula <-c(myformula ,
               sapply(1:5, FUN =function(X)
                 paste(myComb[[4]][,X],collapse = " + " )
                 ),  paste(myComb[[5]],collapse = " + ")
               )

myformula<- paste("Y","~",myformula)

#### SPLITTING AND SUBSETING DATA IN TO K FOLDS

#calculatin no. rows
noobs<-dim(myData)[1]
K <- Nfolds

#Use sample to randomly draw the indexes of the dataset
#and reorder the data with them in a random manner
set.seed(12345)
myData<-myData[sample(noobs),]

#Create K equal indexes that are added to the data.
cut(1:noobs,breaks=K,labels = FALSE)

myData$index <- cut(1:noobs,breaks=K,labels = FALSE)


#init a counting vector "o" useed to loop in to the data.frame "linearModels"
#and a combination index "dataKombs" used for subseting the different datasets
#used for fitting models
o <- 1
linearModels<-data.frame(CV=1,model="text",nofeats=1,stringsAsFactors = FALSE)
dataKombs <- combn(1:K,K-1)
for (m in 1:length(myformula)){
for (l in (1:K)){

  #the data of the K-folds used for the model estimation
  data<-subset(myData, myData$index %in% dataKombs[,l] )


  #the fold that was left out in the model estimation
  predmatrix<-model.matrix(formula(myformula[m]),
            subset(myData, !(myData$index %in% dataKombs[,l] )))



  #Calculating the CV score for each model. sum((Y - Y(hat))^2)
  CV<-sum(
    (
      #this is the observed Y for the left out fold.
```

```
       subset(myData, !(myData$index %in% dataKombs[,1] ))[,1] -
        #predmatrix description above.
        predmatrix %*%
        #the estimated beta-hats.
        linreg(formula = myformula[m], data = data)$beta.hat
       )^2
     )

   #inserting the results in to the linearModels data.frame
   linearModels[o,] <- c(CV,myformula[m],ncol(predmatrix) - 1)
   o <- o + 1
    }
 }
 #reforming data to numeric again.
 linearModels[,1] <- as.numeric( linearModels[,1])
 linearModels[,3] <- as.numeric( linearModels[,3])

 #The mean for the different models, each model is estimadet K times
 plotdata<-suppressWarnings(aggregate(linearModels,by = list(linearModels$model),FUN= mean)[,-3])

 #renaming a column to ease plotting
 colnames(plotdata)[1] <- "Model"

 #plotting
 engr<-ggplot() +
   geom_line(data = aggregate(plotdata,list(plotdata$nofeats),FUN = min)[,c(3,4)],aes(x=nofeats,y=CV)) +
   geom_point(data = plotdata,aes(x=nofeats,y=CV,col = factor(nofeats)) ) +
       labs(title = "CV scores for different no. feats",color = "No. feat")


#displays the plot
plot(engr)

 #Returns the models with the lowest average CV-score
 return( plotdata[min(plotdata$CV) == plotdata$CV,c(1,3,2)])

}

##1.2

myLM(Y = swiss[,1],
     X = swiss[ , 2:ncol(swiss) ],
     Nfolds = 5 )
## 2.1

setwd("/Users/EmilsHem/Documents/732A95/lab2")
TC<- read.csv2("tecator.csv")

ggplot(data = TC, aes(x=Protein,Moisture)) + geom_point()

## 2.3

set.seed(12345)
```

```
obs   <- dim(TC)[1]
id    <- sample(1:obs, floor(obs*0.5))
train <- TC[id,]
valid <- TC[-id,]


  dataMSE <- data.frame(trainMSE = 1, testMSE = 1, noPoly = 1)
  i <-1

for (modell in 1:6) {

  linj<-lm(formula(paste("Moisture ~",paste("I(Protein^",1:modell,")", collapse = " + "))),data = train
  trainMSE<-mean(linj$residuals^2)
  testMSE<-mean( (predict(linj,newdata = valid) - valid$Moisture)^2)
  dataMSE[i,] <- c(trainMSE,testMSE,i)
  i <- i + 1

}

 aplot<- ggplot(data = dataMSE) +
    geom_line(aes(x = noPoly ,y = trainMSE, col = "Train")) +
    geom_line(aes(x = noPoly ,y = testMSE, col = "Test")) +
    labs(title="MSE for different polynomial functions",
         color ="MSE", x="Flexibility", y = "MSE")
  plot(aplot)


##2.4

library(MASS)
varSelect<-lm(Fat ~ . , data = TC[,2:102])
stepRes<-stepAIC(varSelect,trace = FALSE)
cat(length(stepRes$coefficients) - 1)


## 2.5

library(glmnet)
library(reshape2)
wideWeg <- glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 0)
mycoeffs <- coefficients(wideWeg)[-1,]
colnames(mycoeffs) <- wideWeg$lambda

myPlot<-melt(as.matrix(mycoeffs),id = rownames,c("lambda","coef"))

colnames(myPlot) <- c("features","lambda","coef")


ggplot(myPlot,aes(x=log(lambda),y = coef,color = features)) + geom_line(show.legend = FALSE) + labs(x =

## 2.6

lassoReg <- glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 1)
```

```r
mycoeffs <- coefficients(lassoReg)[-1,]
colnames(mycoeffs) <- lassoReg$lambda

myLasso<-melt(as.matrix(mycoeffs),id = rownames,c("lambda","coef"))

colnames(myLasso) <- c("features","lambda","coef")


ggplot(myLasso,aes(x=log(lambda),y = coef,color = features)) +
  geom_line(show.legend = FALSE) + labs(x = expression(log(lambda)) )

## 2.7
set.seed(12345)
cvLASSO<-cv.glmnet(x = as.matrix(TC[,2:101]), y = TC[,102], alpha = 1)

nocoeff<-matrix(coef(cvLASSO,s ="lambda.min"))
nocoeff[nocoeff != 0]
cat(paste("The no. of variables included in the optimal model was: ",length(nocoeff[nocoeff != 0]) - 1))
cat(paste("The lambda of the optimal model was: ",cvLASSO$lambda.min))#antalet coeff exklusive intercep
plotLasso<- data.frame(CV=cvLASSO$cvm,lambda= cvLASSO$lambda)

ggplot(data=plotLasso,aes(x = log(lambda), y = CV)) + geom_point() +
  labs(x = expression(log(lambda)) )
```