Exercise 7 - LASSO, Ridge Regression and elastic nets. Comparing penalized regression estimators

Yannik Gaebel

2022-11-27

Contents

Setup

Set random seed.

```
set.seed(12208157)
```

Import libraries

```
library(ISLR)
library(glmnet)
```

```
## Lade nötiges Paket: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
library(ggplot2)
```

Task 1.

a)

Write your own function for the lasso using the shooting algorithm. Give default values for the tolerance limit and the maximum number of iterations. Do not forget to compute the coefficient for the intercept.

```
myLasso <- function(X, y, coeff, lam, eps = 0.0000001, maxIter = 500){
  p <- dim(X)[2]

  # first row of ones for intercept
  X <- cbind(rep(1,nrow(X)),X)

  change <- 1
  numIters <- 1

  # update parameters as long as change term is bigger than eps
  while (change > eps & numIters < maxIter){
    coeffOld <- coeff

    for (j in 1:p){
      x.ij <- X[, j+1]
      coeff.j <- coeff[j+1]

      a.j <- 2*sum(x.ij^2)
      c.j <- 2*sum(x.ij * (y- t(t(coeff)%*%t(X)) + coeff.j*x.ij))

      # soft
      a <- c.j / a.j
      d <- lam / a.j
      coeff[j+1] <- sign(a) * max(0, abs(a)-d)
    }

    change <- sqrt(sum((coeff-coeffOld)^2))
```

```
    numIters <- numIters + 1
  }

  return (coeff)
}
```

b)

Write a function which computes the lasso using your algorithm for a vector of lambdas and which returns the matrix of coefficients and the corresponding lambda values.

```
lassoTuning <- function(X, y, coeff, lamValues){
  result_vector <- c()
  for (lam in lamValues){
    coeffs <- myLasso(X,y,coeff,lam)
    result_vector <- c(result_vector,lam,coeffs)
  }
  return (matrix(result_vector,nrow=length(lamValues),ncol=dim(X)[2]+2,byrow=TRUE))
}
```

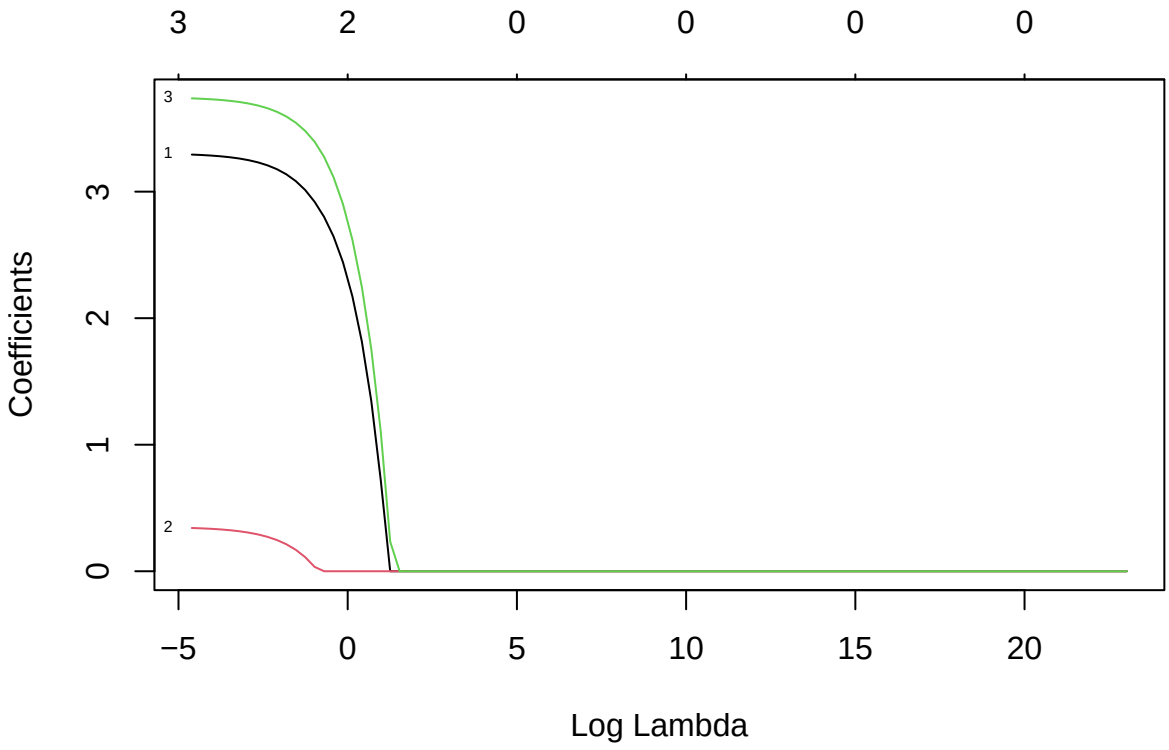
c)

Compare the performance and output of your functions against the lasso implementation from glmnet.

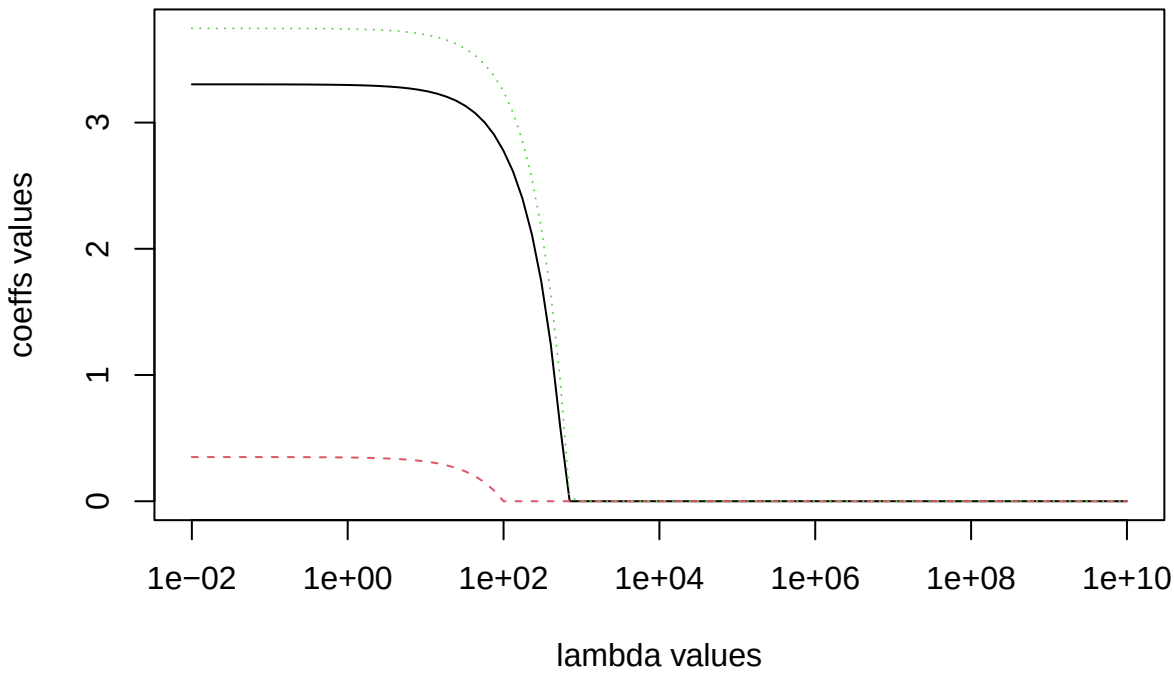
I compare the resulting coefficients.

```
# generate random test data
set.seed(12208157)

n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
eps <- rnorm(n, sd=1.5)
Beta <- c(2,3,0.5,4)
X <- model.matrix(~x1+x2+x3)
y <- X %*% Beta + eps
X <- matrix(c(x1,x2,x3),ncol=3)

lambda.grid <- 10^seq(-2,10, length=100)
```

```
# fit glmnet implementation for lasso
lasso.glm <- glmnet(X, y, alpha=1, lambda=lambda.grid)
plot(lasso.glm, xvar="lambda", label=TRUE)
```




```
# fit my own implementation for lasso
lm_help <- lm(y~X)
coeff_start <- as.vector(lm_help$coefficients)

lassoResults <- lassoTuning(X,y,coeff_start,lambda.grid)

lambdas <- lassoResults[1:100,1:1]
coeffs <- lassoResults[1:100,3:5]

matplot(lambdas, coeffs, type="l", log="x", xlab = "lambda values", ylab = "coeffs values")
```



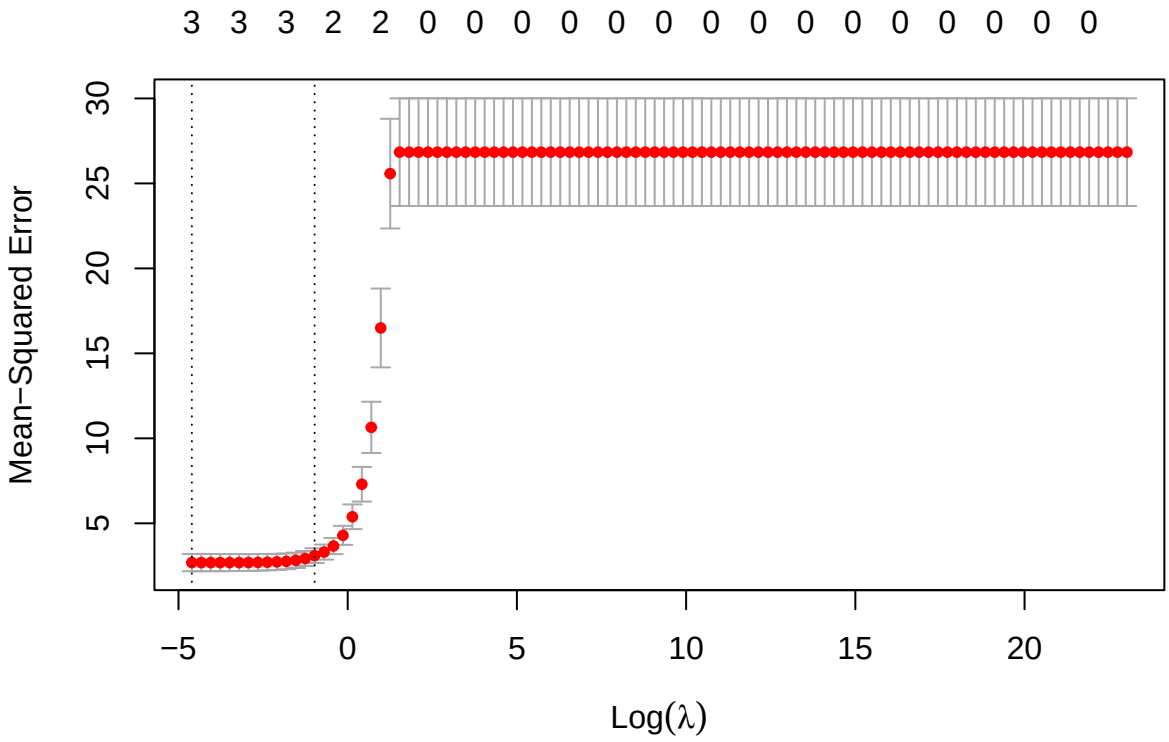
The change in coefficients for different lambda values looks very similar for for glmnet and the shooting algorithm implementation.

d)

Write a function to perform 10-fold cross-validation for the lasso using MSE as the performance measure. The object should be similarly to the cv.glmnet give the same plot and return the lambda which minimizes the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation, respectively.

glmnet implementation:

```
cv.glm <- cv.glmnet(X, y, alpha = 1, lambda = lambda.grid)
plot(cv.glm)
```



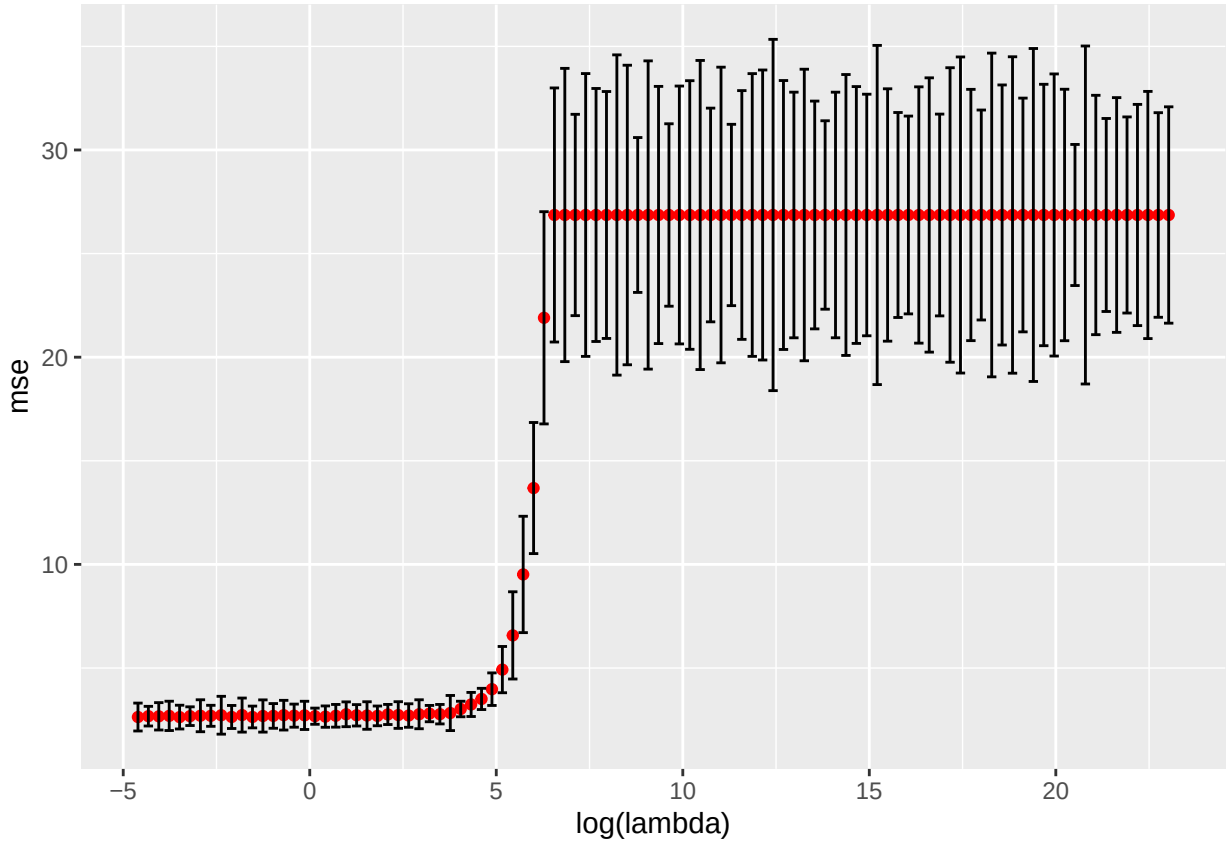
10-fold cross validation:

```
lassoCV <- function(X, y, coeffs, lamValues){

  nrows <- dim(X)[1]
  ncols <- dim(X)[2]

  # result vectors
  mse.mean <- c()
  mse.std <- c()

  for (lam in lamValues){
    mse <- c()

    # split data
    # create list of folds
    folds <- split(1:nrow(X), sample(rep(1:10, length.out=nrow(X))))

    for (fold in folds){
      X.train <- X[-fold,]
      X.test <- X[fold,]
      y.train <- y[-fold]
      y.test <- y[fold]

      # get ceofficients with lasso function
      coeffs.cv <- myLasso(X.train,y.train,coeffs,lam)
```

```
      #predict
      coeffs.cv <- as.matrix(coeffs.cv)
      y.pred <- t(coeffs.cv[-1]) %*% t(X.test) + coeffs.cv[1]
      #print(y.pred)

      mse.fold <- mean((y.test - y.pred)^2)
      mse <- c(mse,mse.fold)

    }

    mse.mean <- c(mse.mean, mean(mse))
    mse.std <- c(mse.std, sd(mse))
  }

  cv.lasso <- data.frame(mse = mse.mean,
                         std = mse.std,
                         lambda = lamValues)

  # plot results
  plt <- ggplot(cv.lasso, aes(x=log(lambda), y=mse))
  plt <- plt + geom_point(col='red')
  plt <- plt + geom_errorbar(aes(ymin=mse-(std/2), ymax=mse+(std/2)))

  # identify optimal lambda value
  min.mse <- min(cv.lasso$mse)
  opt.lam <- cv.lasso[cv.lasso$mse == min.mse, "lambda"]

  return(list('plot' = plt, 'lambda.minMSE' = opt.lam))

}

cv.lasso <- lassoCV(X, y, coeff_start,lambda.grid)
print('Output:')
```

```
## [1] "Output:"
```

```
cv.lasso$plot
```




```
print(paste0('Lambda which minimizes the Root Mean Squared Error: ', cv.lasso$lambda.minMSE))
```

```
## [1] "Lambda which minimizes the Root Mean Squared Error: 0.0305385550883342"
```

Task 2.

We will work with the Hitters data in the ISLR package. Take the salary variable as the response variable and create the model matrix x based on all other variables in the data set. Then divide the data into training and testing data with a ratio of 70:30.

```
# remove missing values
data <- na.omit(Hitters)

# convert factors to numeric
data$League <- as.numeric(factor(data$League))
data$Division <- as.numeric(factor(data$Division))
data$NewLeague <- as.numeric(factor(data$NewLeague))

response <- data$Salary
data$Salary <- NULL

# apply scaling
data <- as.matrix(scale(data,center=TRUE,scale=TRUE))
```

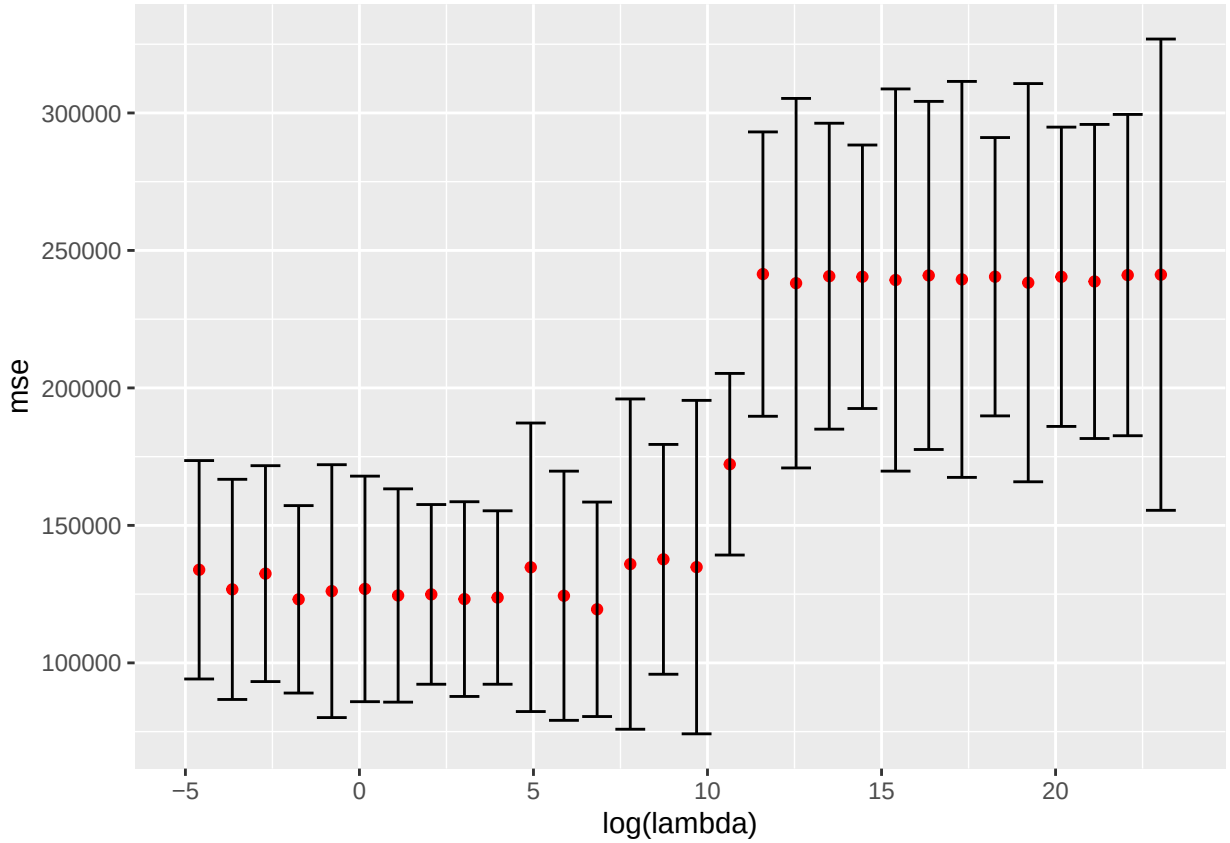
```r
# train test split
n <- nrow(data)
train_sample <- sample(1:n,n*0.7)

data.train <- data[train_sample,]
data.test <- data[-train_sample,]
response.train <- response[train_sample]
response.test <- response[-train_sample]
```

a)

Use your lasso function to decide which lambda is best here. Plot also the whole path for the coefficients. I had to experiment with the lambda grid a little bit because the runtime was much larger than before.

```r
lm_help <- lm(response.train~data.train)
coeff_start <- as.vector(lm_help$coefficients)

lambda.grid <- 10^seq(-2,10, length=30)

lassoCV(data.train,response.train,coeff_start,lambda.grid)
```

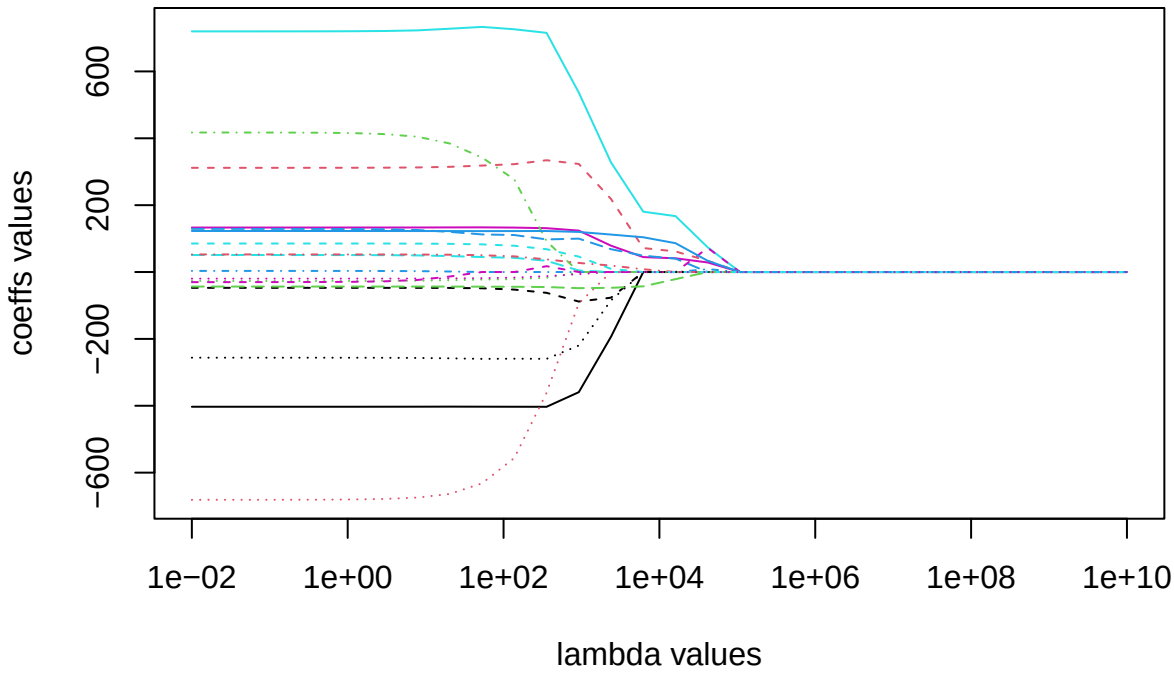
```
## $plot
```



```
##
## $lambda.minMSE
## [1] 923.6709
```

Best lambda identified by cross validation function is: 3.03

```
numLambdas <- 30
lambda.grid <- 10^seq(-2,10, length=numLambdas)

lassoResults <- lassoTuning(data.train,response.train,coeff_start,lambda.grid)

lambdas <- lassoResults[1:numLambdas,1:1]
coeffs <- lassoResults[1:numLambdas,3:20]

matplot(lambdas, coeffs, type="l", log="x", xlab = "lambda values", ylab = "coeffs values")
```

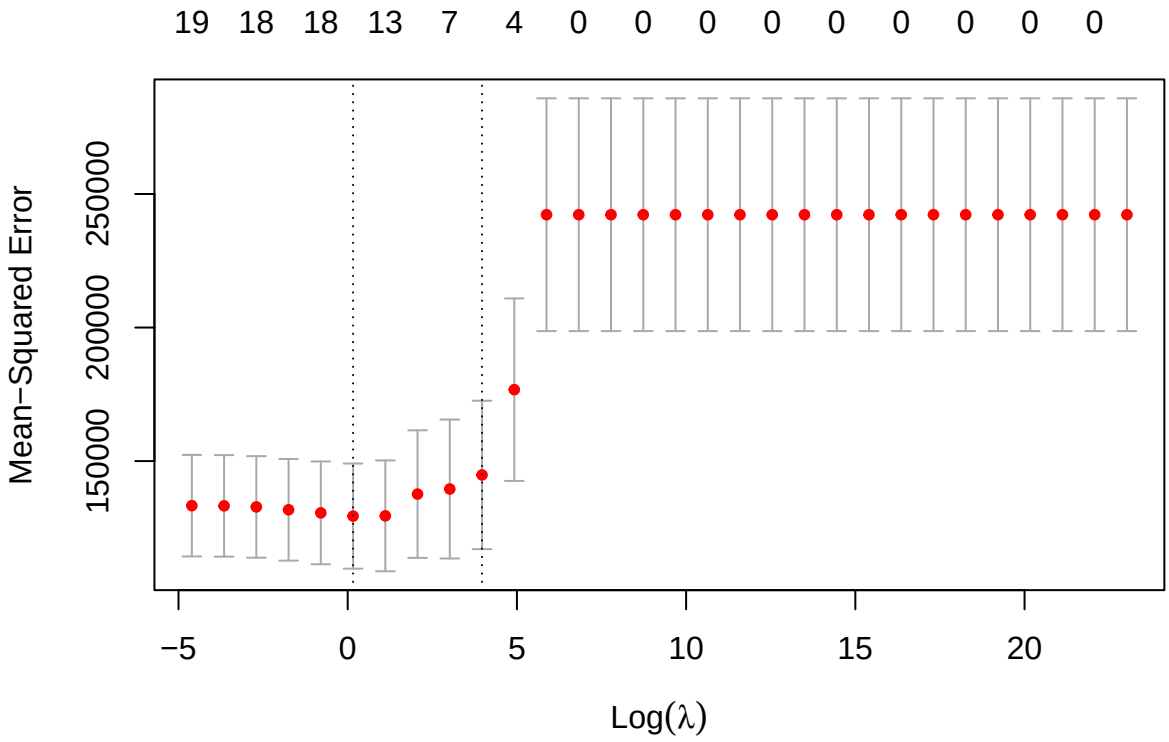


b)

Compare your fit against the lasso implementation from glmnet.

Compare the output of glmnet cross validation with the output of my function:

```
cv.glm.hitters <- cv.glmnet(data.train, response.train, alpha = 1, lambda = lambda.grid)
plot(cv.glm.hitters)
```



The output looks similar, however there are some differences for the implementation of glmnet:

- The coefficients are shrunk earlier. For smaller lambda values.
- The performance is much better.
- The error bars fluctuate much less.

c)

Fit also a ridge regression and a least squares regression for the data (you can use here glmnet).

```
# fit lasso regression
glm.lasso <- cv.glmnet(data.train, response.train, alpha = 1)

# fit ridge regression
glm.ridge <- cv.glmnet(data.train, response.train, alpha = 0)

# fit least squares regression
least_squares <- lm(response.train ~ data.train)
```

d)

Compute the lasso, ridge regression and ls regression predictions for the testing data. Which method gives the better predictions? Interpret all three models and argue about their performances.

```r
# prediction of lasso
lasso_pred <- predict(glm.lasso,newx=data.test,s="lambda.1se")
print('MSE of lasso regression:')
```

```
## [1] "MSE of lasso regression:"
```

```r
mean((response.test - lasso_pred)^2)
```

```
## [1] 74663.83
```

```r
# prediction of ridge
ridge_pred <- predict(glm.ridge,newx=data.test,s="lambda.1se")
print('MSE of ridge regression:')
```

```
## [1] "MSE of ridge regression:"
```

```r
mean((response.test - ridge_pred)^2)
```

```
## [1] 70651.82
```

```r
# prediction of least squares
ls_pred <- predict(least_squares,newx=data.test)
print('MSE of least squares regression:')
```

```
## [1] "MSE of least squares regression:"
```

```r
mean((response.test - ls_pred)^2)
```

```
## Warning in response.test - ls_pred: Länge des längeren Objektes
##       ist kein Vielfaches der Länge des kürzeren Objektes
```

```
## [1] 248547.4
```

The ridge regression has a lower mse than lasso regression. The worst performing model by far is the least squares regression. The regularization seams to make the model fit better for this dataset. The ridge regression performs a little better than lasso regression. However, lasso generally uses much less variables than ridge so it might lead to a better explainable model with just a small sacrifice of performance.

Task 3.

Explain the notion of regularized regression, shrinkage and how Ridge regression and LASSO regression differ.

Answer:

The idea of regularized regression is to simplify the model as far as possible without sacrificing performance. There are multiple advantages to this. First having a smaller model with shrunk parameters can make the model easier to interpret and shows which variables are actually important to predict the response variable. Second by simplifying the model it can prevent overfitting. Shrinking the coefficients can prevent the model

from fitting to close to the training data and rather learn important trends in the data. It introduces a bias into the model which can result in reduced variance which can lead to improved performance.

Ridge and LASSO regression differ in the penalty term they impose. Ridge regression adds the sum of squared coefficients to the function to be minimized. LASSO however adds the sum of absolute coefficient values to the function to be minimized. Practically ridge regression shrinks insignificant coefficients close to zero and LASSO regression shrinks them to zero. For this reason LASSO is sometimes preferred.