Assignment 7: Comparing penalized regression estimators

Statistical Simulation and Computerintensive Methods

Joan Salvà

December 2022

Task 1

1. 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.

The intercept is zero for scaled data, so we can omit it for simplicity because we will scale the model data.

```
library(faraway)
```

```
## Warning: package 'faraway' was built under R version 4.2.2
```

```
data(swiss)
head(swiss)
dim(swiss)


swiss <- scale(swiss)

X <- as.matrix(swiss[, -4])
y <- swiss[,4]
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
## Moutier                  20.3
## Neuveville               20.6
## Porrentruy               26.6
## [1] 47  6
```

```r
softmax <- function(x, y){
  sign(x) * pmax(abs(x) - y, 0)
}


lasso_shoot <- function(X, y, lambda, tol = 1e-8, max_iter = 10000) {

  p <- ncol(X)
  XX <- crossprod(X, X)
  XX2 <- 2 * XX
  Xy <- crossprod(X, y)
  Xy2 <- 2 * Xy

  beta <- solve(XX + diag(lambda, p, p), Xy)
  # print(paste('First beta', beta))

  converged <- FALSE
  iteration <- 0

  while (!converged & (iteration < max_iter)){

    beta_prev <- beta

    for (j in 1:p){
      aj <- XX2[j,j]
      cj <- Xy2[j] - sum(XX2[j,] %*% beta) + beta[j] * XX2[j,j]
      beta[j] <- softmax(cj / aj, lambda / aj)
    }

    iteration <- iteration + 1
    # print(paste(beta, beta_prev))
    converged <- sum(abs(beta - beta_prev)) < tol

    # print(paste(converged, iteration))
  }
  out <- list(beta = beta, n_iter = iteration,
              converged = converged)
  return(out)
}
```

Let's test the function on the `Swiss` dataset. It converges to the following coefficients

```r
lasso_shoot(X, y, lambda = 0.001)
```

```
## $beta
##                          [,1]
## Fertility        -0.53068229
## Agriculture      -0.38361227
## Examination       0.34830015
## Catholic          0.43470398
## Infant.Mortality  0.06180505
##
## $n_iter
```

```
## [1] 14
##
## $converged
## [1] TRUE
```

2. Write a function which computes the Lasso using your algorithm for a vector of λs and which returns the matrix of coefficients and the corresponding λ values.

```
to <- 1
from <- 0.00001
n_lambdas <- 10

lambdas <- seq(from, to, by = (to - from) / (n_lambdas - 1))

lasso_coefficients <- function(X, y, lambdas, tol = 1e-8, max_iter = 10000) {
  df <- data.frame(matrix(ncol = 1 + ncol(X), nrow = 0))
  colnames(df) <- c('lambda', colnames(X))

  for (lambda in lambdas) {
    # print(paste('Getting Lasso coef for lambda:', lambda))
    res <- lasso_shoot(X, y, lambda = lambda, tol = tol, max_iter = max_iter)
    df[nrow(df) + 1,] <- c(lambda, t(res$beta))
  }
  return(df)
}
```

Let's test the function on the `Swiss` dataset. Each row contains the tested `lambda` and the computed coefficients.

```
res <- lasso_coefficients(X, y, lambdas)
print(res)
```

```
##     lambda  Fertility Agriculture Examination  Catholic Infant.Mortality
## 1  0.00001 -0.5307031  -0.3836185   0.3483097 0.4347296       0.06182070
## 2  0.11112 -0.5283660  -0.3829224   0.3472437 0.4318566       0.06006444
## 3  0.22223 -0.5260288  -0.3822262   0.3461777 0.4289837       0.05830817
## 4  0.33334 -0.5236916  -0.3815301   0.3451117 0.4261107       0.05655191
## 5  0.44445 -0.5213545  -0.3808340   0.3440457 0.4232378       0.05479564
## 6  0.55556 -0.5190173  -0.3801379   0.3429797 0.4203648       0.05303938
## 7  0.66667 -0.5166802  -0.3794418   0.3419137 0.4174919       0.05128311
## 8  0.77778 -0.5143430  -0.3787457   0.3408477 0.4146190       0.04952685
## 9  0.88889 -0.5120059  -0.3780496   0.3397818 0.4117460       0.04777058
## 10 1.00000 -0.5096687  -0.3773535   0.3387158 0.4088731       0.04601432
```

3. Compare the performance and output of your functions against the Lasso implementation from `glmnet`.

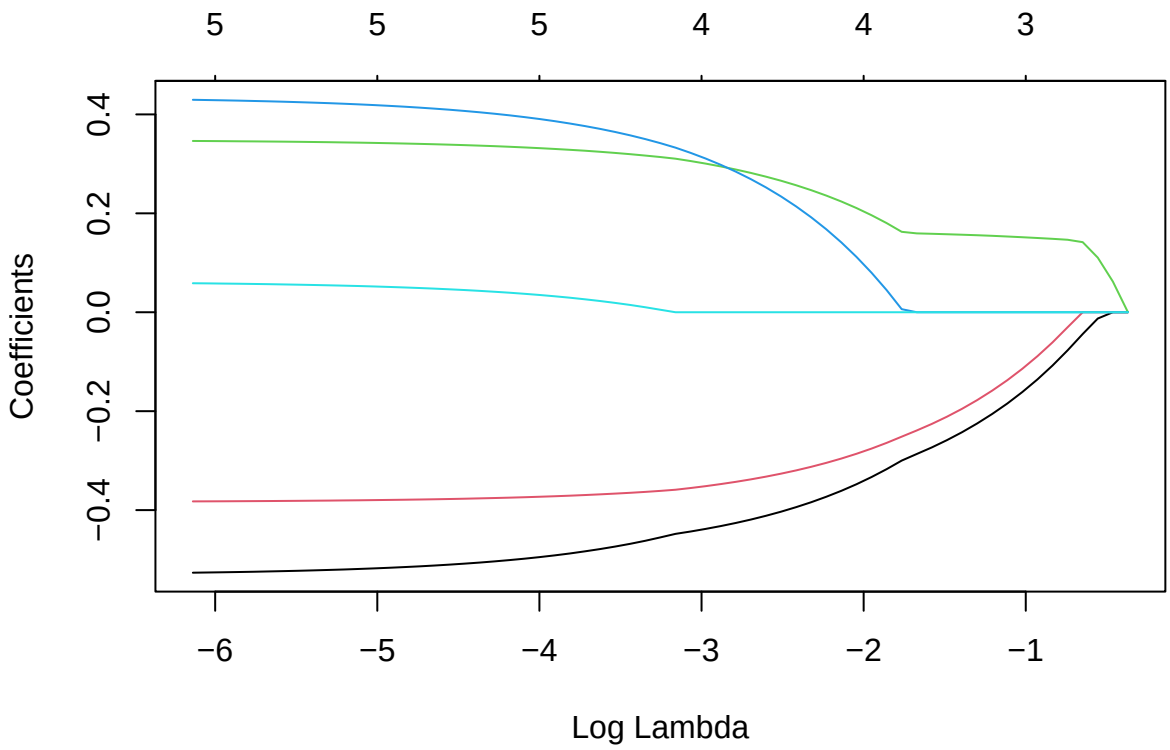
```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.2
```

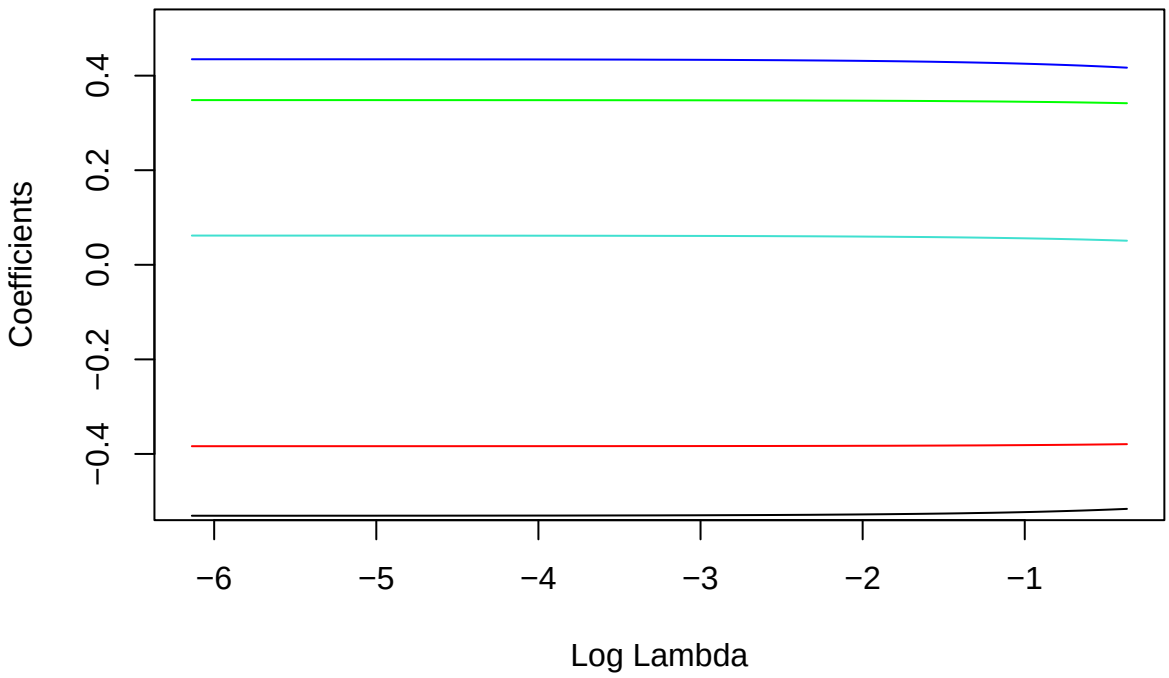
```
## Loading required package: Matrix
```

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

```
# glmnet
mod.glmnet <- glmnet(X, y)
plot(mod.glmnet, xvar='lambda')
```



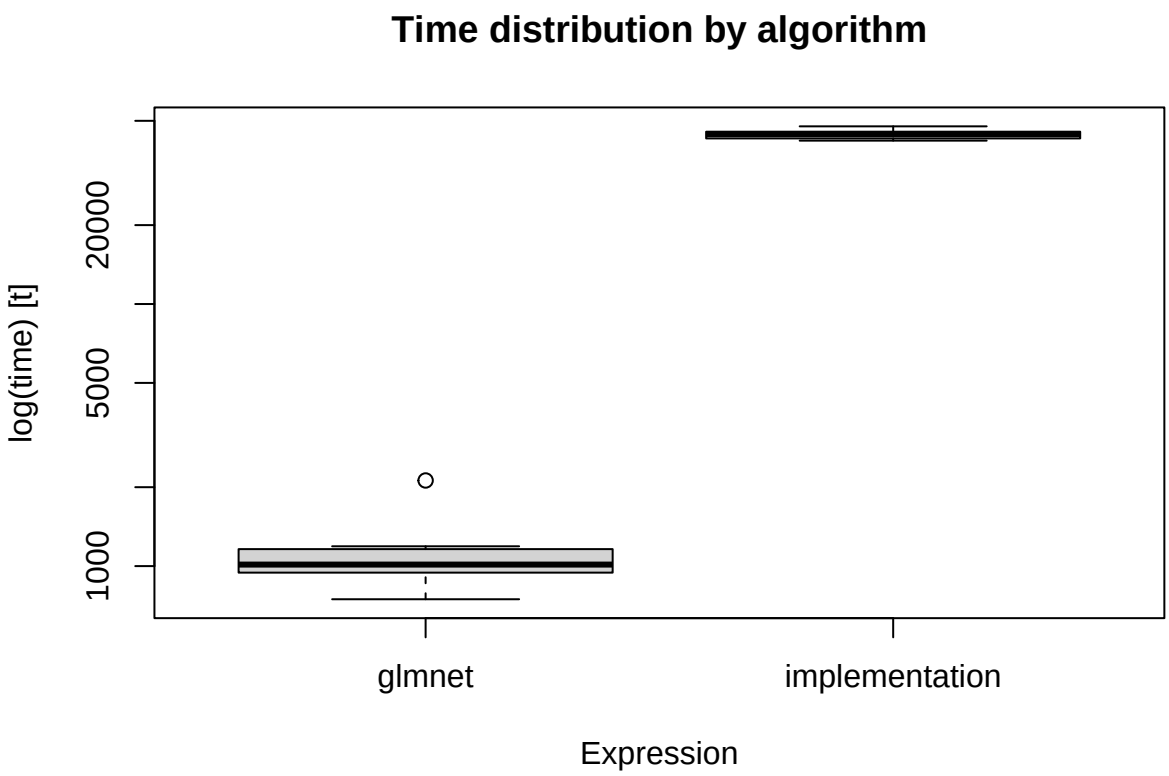

```
# Our implementation. We use the same lambdas explored in glmnet
mod.coef <- lasso_coefficients(X, y, mod.glmnet$lambda)

plot(Fertility ~ log(lambda), data = mod.coef, type='l', col = 'black',
     ylim=c(-.5,.5), ylab = 'Coefficients', xlab = 'Log Lambda')
lines(x = log(mod.coef$lambda), mod.coef$Agriculture, col = 'red')
lines(x = log(mod.coef$lambda), y = mod.coef$Examination, col = 'green')
lines(x = log(mod.coef$lambda), y = mod.coef$Catholic, col = 'blue')
lines(x = log(mod.coef$lambda), y = mod.coef$Infant.Mortality, col = 'turquoise')
```



The variation of the coefficients have the same direction, even though in our implenetation, they do not converge to zero as lambda increases.

Let's test the runtime of our functions using the package `microbenchmark`:

```
library(microbenchmark)
micro <- microbenchmark(glmnet(X, y),
                        lasso_coefficients(X, y, mod.glmnet$lambda), times=20)
micro$expr <- ifelse(micro$expr != 'glmnet(X, y)', 'implementation', 'glmnet')
boxplot(micro, main="Time distribution by algorithm", )
```



As expected, implementation is way slower than `glmnet`.

4. 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 λ which minimizes the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation, respectively.

We first code the functions to compute the error measures on each fold.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.2
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:faraway':
##
##     melanoma
```

```
mse <- function(pred, y) {
  return(mean((pred - y)^2))
}

rmse <- function(pred, y) {
  return(sqrt(mse(pred, y)))
}

mad <- function(pred, y) {
  return(median(abs(pred - y)))
}

predict.test.lambdas <- function(X_test, y_test, lasso_coef) {

  lasso_coef$mse <- NA
  lasso_coef$rmse <- NA
  lasso_coef$mad <- NA

  for (i in 1:nrow(lasso_coef)) {
    lambda <- lasso_coef[i, 1]
    # print(paste('Lambda', lambda))
    coefs <- as.vector(unlist(lasso_coef[i, 2:(ncol(lasso_coef) - 3)]))
    pred <- X_test %*% coefs

    lasso_coef[i,]$mse <- mse(pred, y_test)
    lasso_coef[i,]$rmse <- rmse(pred, y_test)
    lasso_coef[i,]$mad <- mad(pred, y_test)

  }
  return(lasso_coef)
}
```

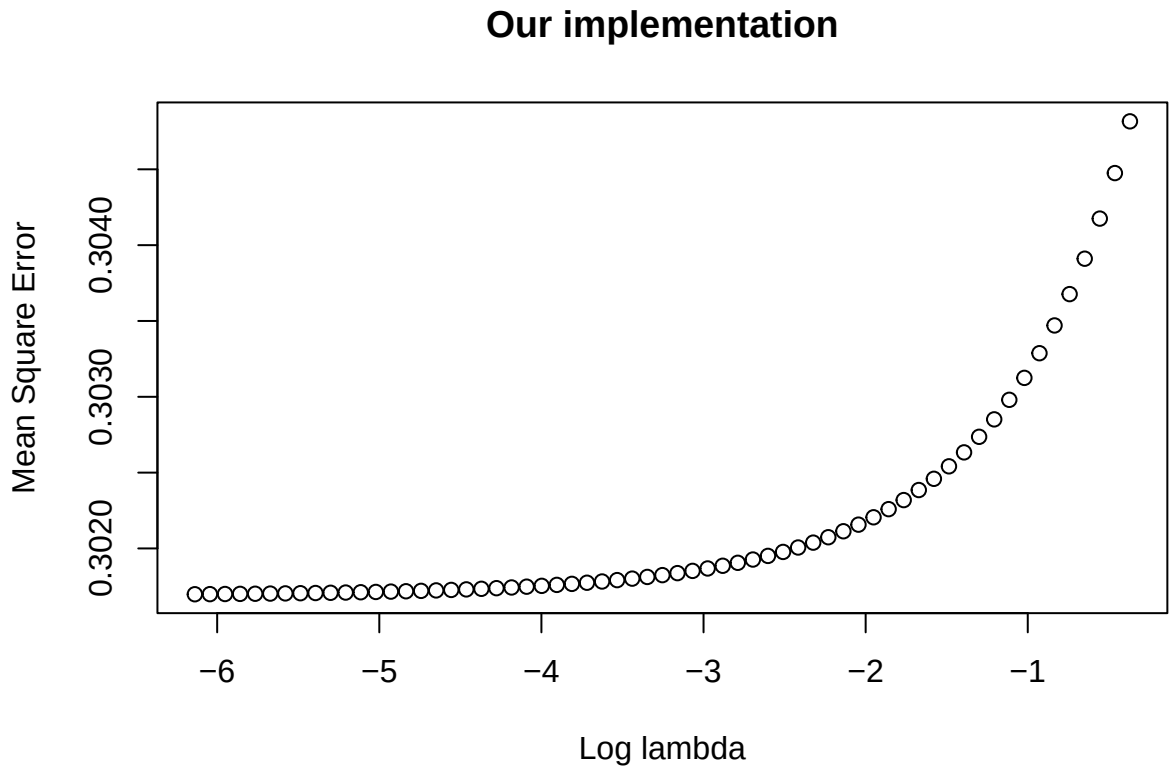
Then we use the method `createFolds` from `caret` to compute the k-fold indices.

```
cv.lasso <- function(X, y, lambdas, k = 10) {
  colnames <- c('lambda', 'RMSE', 'MSE', 'MAD', 'RMSE_sd', 'MSE_sd', 'MAD_sd')
  lambda_cv_errors <- data.frame(matrix(nrow=0, ncol=length(colnames)))
  names(lambda_cv_errors) <- colnames

  fold_errors <- list()

  set.seed(0)

  folds  <- createFolds(1:nrow(X), k)
  ifold <- 1
  for (fold in folds) {
    print(paste('Validating on fold', ifold))
    X_train <- as.matrix(X[-fold,])
    y_train <- y[-fold]

    X_test <- as.matrix(X[fold,])
    y_test <- y[fold]
```

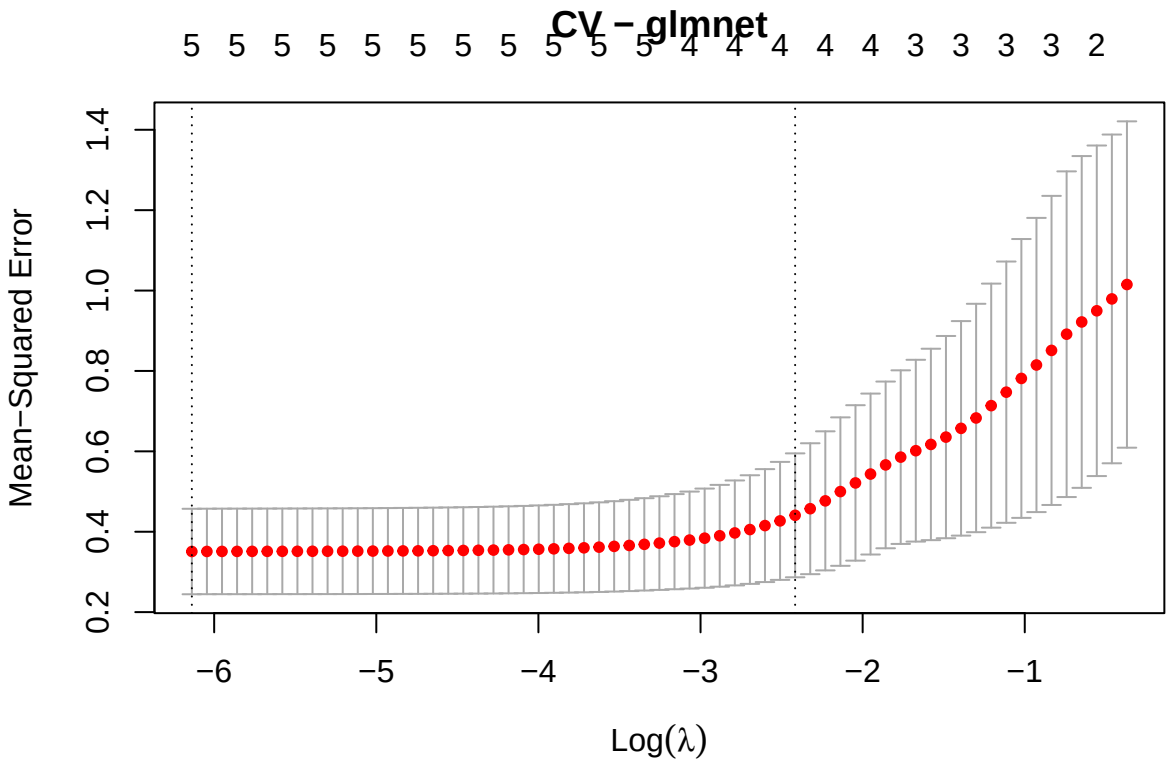
```
    res <- lasso_coefficients(X_train, y_train, lambdas)
    res <- predict.test.lambdas(X_test, y_test, res)

    fold_errors[[ifold]] <- res
    ifold <- ifold+1

  }

  for (i in 1:length(lambdas)) {
    mse <- c()
    rmse <- c()
    mad <- c()
    for (ifold in 1:k) {
      mse <- c(mse, fold_errors[[ifold]][i, ]$mse)
      rmse <- c(rmse, fold_errors[[ifold]][i, ]$rmse)
      mad <- c(mad, fold_errors[[ifold]][i, ]$mad)
    }
    lambda_cv_errors[i,] <- c(
      lambdas[i], mean(rmse), mean(mse), mean(mad), sd(rmse), sd(mse), sd(mad))

  }

  return(lambda_cv_errors)

}
```

Let's test the function on the `Swiss` dataset.

```
res <- cv.lasso(X, y, lambdas = mod.glmnet$lambda)

plot(log(res$lambda), res$MSE, main = 'Our implementation',
     xlab = 'Log lambda', ylab = 'Mean Square Error')
```




```
mod.cv.glmnet <- cv.glmnet(X, y)
plot(mod.cv.glmnet, main = 'CV - glmnet')
```




```
print('Optimal lambdas: ')
print(paste('According to RMSE:', res[which.min(res$RMSE),]$lambda))
print(paste('According to MSE:', res[which.min(res$MSE),]$lambda))
print(paste('According to MAD:', res[which.min(res$MAD),]$lambda))
```

```
## [1] "Validating on fold 1"
## [1] "Validating on fold 2"
## [1] "Validating on fold 3"
## [1] "Validating on fold 4"
## [1] "Validating on fold 5"
## [1] "Validating on fold 6"
## [1] "Validating on fold 7"
## [1] "Validating on fold 8"
## [1] "Validating on fold 9"
## [1] "Validating on fold 10"
## [1] "Optimal lambdas: "
## [1] "According to RMSE: 0.69094539920573"
## [1] "According to MSE: 0.00215969898556652"
## [1] "According to MAD: 0.69094539920573"
```

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.

We will omit the missing values, one-hot encode the categorical variables, and scale the every column.

```
library(ISLR)
data('Hitters')

X <- na.omit(Hitters)
y <- X$Salary

X$Salary <- NULL

library(mltools)
```

```
## Warning: package 'mltools' was built under R version 4.2.2
```

```
##
## Attaching package: 'mltools'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     mse, rmse
```

```
library(data.table)
X <- scale(one_hot(as.data.table(X)))

train_indices <- sample(1:nrow(X), size = round(0.7 * nrow(X)), replace = FALSE)

xtrain <- as.matrix(X[train_indices, ])
ytrain <- y[train_indices]
xtest <- as.matrix(X[-train_indices, ])
ytest <- y[-train_indices]
```

1. Use your Lasso function to decide which lambda is best here. Plot also the whole path for the coefficients.

```
mod.lasso <- cv.glmnet(as.matrix(xtrain), ytrain)
n_lambdas <- 5
from <-  min(mod.lasso$lambda)
to <- max(mod.lasso$lambda)
lambdas <- seq(from, to, by = (to - from) / (n_lambdas - 1))

res <- cv.lasso(xtrain, ytrain, lambdas = lambdas, k = 10)

print('Optimal lambdas: ')
print(paste('According to RMSE:', res[which.min(res$RMSE),]$lambda))
print(paste('According to MSE:', res[which.min(res$MSE),]$lambda))
print(paste('According to MAD:', res[which.min(res$MAD),]$lambda))
```

```
## [1] "Validating on fold 1"
## [1] "Validating on fold 2"
## [1] "Validating on fold 3"
## [1] "Validating on fold 4"
```

```
## [1] "Validating on fold 5"
## [1] "Validating on fold 6"
## [1] "Validating on fold 7"
## [1] "Validating on fold 8"
## [1] "Validating on fold 9"
## [1] "Validating on fold 10"
## [1] "Optimal lambdas: "
## [1] "According to RMSE: 275.074496064849"
## [1] "According to MSE: 275.074496064849"
## [1] "According to MAD: 137.57211873795"
```

2. Compare your fit against the Lasso implementation from `glmnet`.

We will take the best lambda according to the MSE.

```
lambda <- res[which.min(res$MSE),]$lambda
lasso_coef <- lasso_shoot(xtrain, ytrain, lambda)$beta

pred <- xtest %*% lasso_coef
print(paste('Implemented Lasso with lambda:', lambda))
print(paste('   RMSE:', rmse(pred, ytest), 'MSE:',
            mse(pred, ytest), 'MAD:', mad(pred, ytest)))

glmnet.pred <- predict(mod.lasso, xtest)
print(paste('Glmnet Lasso with lambda:', mod.lasso$lambda.1se))
print(paste('   RMSE:', rmse(glmnet.pred, ytest), 'MSE:',
            mse(glmnet.pred, ytest), 'MAD:', mad(glmnet.pred, ytest)))
```

```
## [1] "Implemented Lasso with lambda: 275.074496064849"
## [1] "   RMSE: 747.563537279714 MSE: 558851.242270159 MAD: 579.272111926874"
## [1] "Glmnet Lasso with lambda: 68.1380530338573"
## [1] "   RMSE: 404.699085290286 MSE: 163781.349634794 MAD: 168.738454769116"
```

Our model predicts fairly worse than `glmnet`. As we saw in the plots, the coefficients are not the same, and therefore we can't expect the predictions to be similar.

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

```
# Ridge regression
mod.ridge <- cv.glmnet(as.matrix(xtrain), ytrain, alpha = 0)
mod.ls <- lm(ytrain ~ ., data = as.data.frame(xtrain))
```

4. 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.

```
# Ridge regression
pred.ridge <- predict(mod.ridge, as.matrix(xtest))
pred.ls <- predict(mod.ls, as.data.frame(xtest))
```

```
## Warning in predict.lm(mod.ls, as.data.frame(xtest)): prediction from a rank-
## deficient fit may be misleading
```

```
pred.lasso <- predict(mod.lasso, as.matrix(xtest))

tab <- data.frame(
  model = c('Ridge', 'Lasso', 'LS'),
  RMSE = c(rmse(pred.ridge, ytest), rmse(pred.lasso, ytest), rmse(pred.ls, ytest)),
  MSE = c(mse(pred.ridge, ytest), mse(pred.lasso, ytest), mse(pred.ls, ytest)),
  MAD = c(mad(pred.ridge, ytest), mad(pred.lasso, ytest), mad(pred.ls, ytest))
)
library(knitr)
knitr::kable(tab, caption = 'Test error by model')
```

Table 1: Test error by model

model	RMSE	MSE	MAD
Ridge	403.3461	162688.1	191.9959
Lasso	404.6991	163781.3	168.7385
LS	427.2545	182546.4	234.4869

Lasso and Ridge clearly overperform the standard Linear Squares regression for the Hitters dataset. They provide better RMSE, MSE and MAD. Comparing Ridge and Lasso, the first one has less RMSE and MSE, while the second one has a significantly better MAD. We would choose Ridge, but knowing that this choice is very dependent on the dataset, and that the Lasso regression would provide a similar accuracy.

##Task 3 Explain the notion of regularized regression, shrinkage and how Ridge regression and Lasso regression differ.

Small models are usually desirable, so one would like to select the best subset of features to fit a model. One can use different feature selection methods to find a good subset of explanatory variables: PCA, step-wise regression, best subset regression... An alternative is to fit a model but 'regularizing' the coefficients. That is including a penalty on their size, 'shrinking' them towards zero and thus effectively removing parameters with small coefficients from the model.

Ridge and Lasso regression are two regularized approaches that differ in the way they penalize the size of the coefficients. They both determine the coefficients that minimize the Least Squares term of the standard Linear Regression, but with the following added components: Lasso: L_1 norm $\sum_{i=1}^{p} |\beta_i|$ Ridge: L_2 norm $\frac{1}{2}\sqrt{\sum_{i=1}^{p} \beta_i^2}$

Regularization of the coefficients also provides models that are less likely to overfit because they are simpler than the baseline model.