Ex7

Yelyzaveta Klysa

2023-10-20

Contents

Task 1

1.1 Write your own function for the lasso using the shooting algorithm.

```r
set.seed(12208877)
rmse <- function(y, pred) {
  sqrt(mean((y - pred)^2))
}

soft <- function(a, d){
  sign(a) * max(abs(a) - d, 0)
}

my_lasso <- function(X, y, lam, tol=10^-7, max_iter= 10^5) {
  df <- data.frame(X, y)
  lm_help <- lm(y~.,df)
  coeff <- coef(lm_help)

  X <- data.frame(X)
  n <- nrow(X)
  ones_inter <- rep(1, n)
  X$intercept <- ones_inter
  X <- X %>% select(intercept, everything())
  p <- ncol(X)

  # update coefficients
  a <- 2 * colSums(X^2)
  for (i in 1:max_iter) {
    old_coef <- coeff

    for (j in 2:p){
      x_j <- X[, j]

      a_j <- a[j]
      c_j <- 2*sum(x_j * (y - as.matrix(X) %*% as.matrix(coeff) + coeff[j]*x_j))

      # soft
      coeff[j] <- soft(c_j/a_j, lam/a_j)
    }

    if (sum(abs(coeff - old_coef)) < tol){
      break
    }
  }

  return (coeff)
}
```

1.2 Write a function which computes the lasso using your algorithm for a vector of lam-s and which returns the matrix of coefficients and the corresponding lam values.

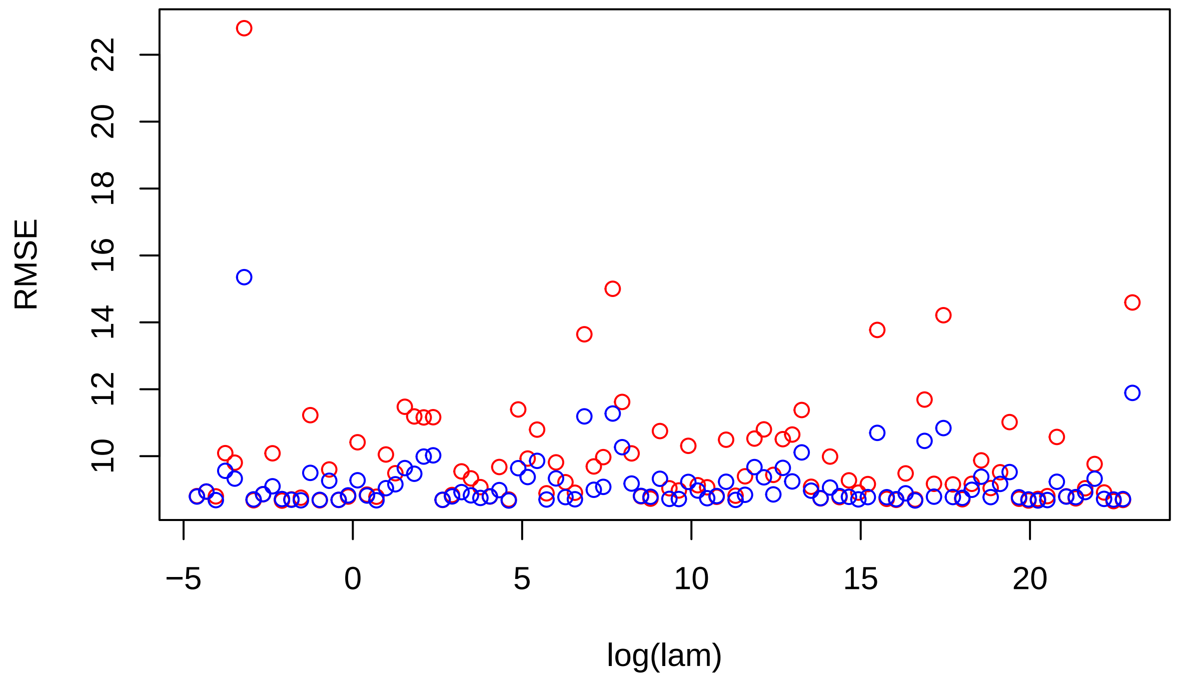
```r
lassoTuning <- function(X, y, lams){
  res <- c()
  for (lam in lams){
    coef <- my_lasso(X, y, lam)
    res <- c(res, lam, coef)
  }
  res <- matrix(res, nrow=length(lams), byrow=TRUE)
  return (res)
}
```

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

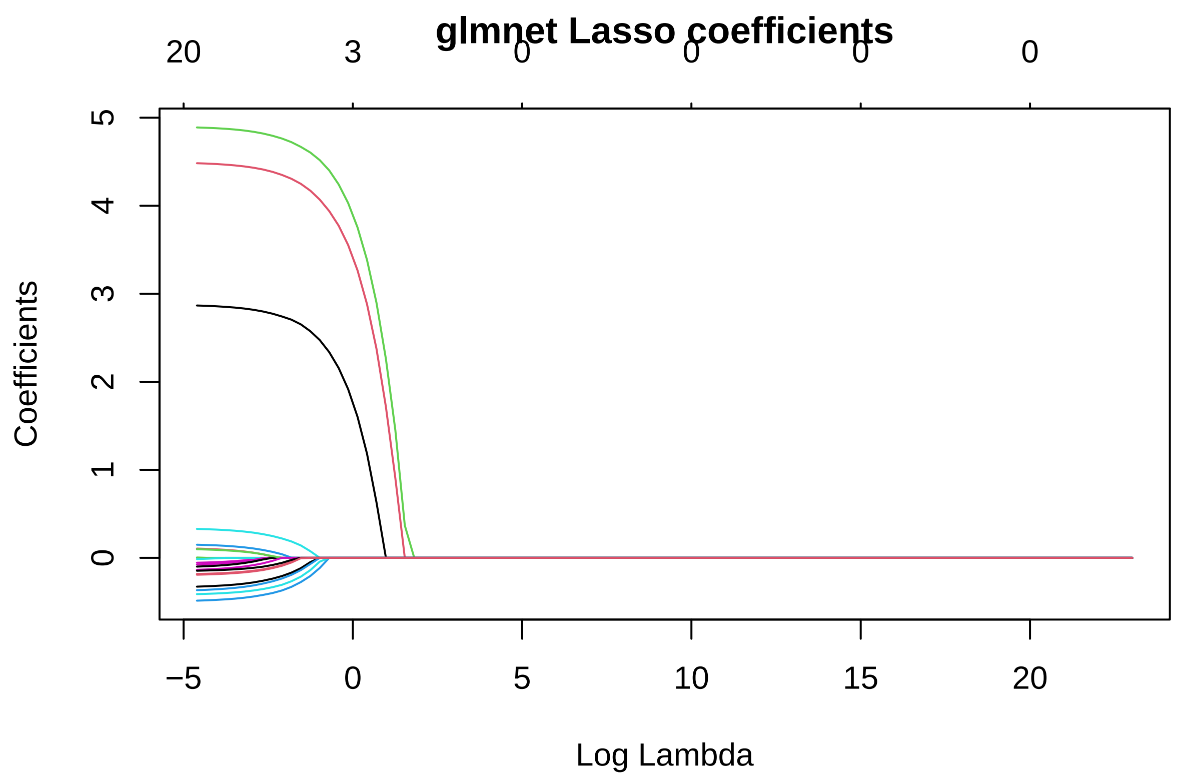
```r
n <- 500
p <- 20
X <- matrix(rnorm(n*p), ncol=p)
X <- scale(X)
eps <- rnorm(n, sd=5)
beta <- 3:5
```

```r
y <- 2 + X[,1:3] %*% beta + eps

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

shooting_coeffs <- lassoTuning(X, y, lambda.grid)

X_with_intercept <- cbind(rep(1,nrow(X)), X)
preds <- shooting_coeffs[,-1] %*% t(X_with_intercept)

res_lasso <- glmnet(X, y, lambda = lambda.grid)
lasso_pred <- predict(res_lasso, X)
```

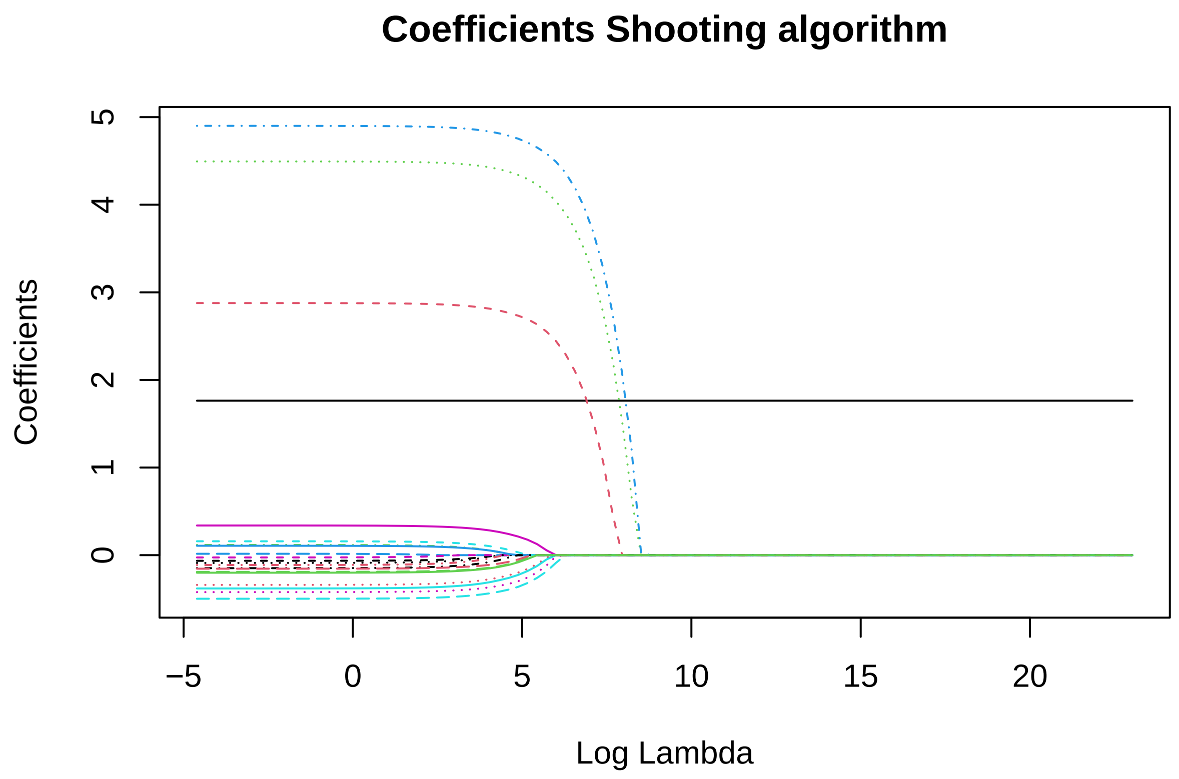
```r
rmse_shooting <- numeric(length(lambda.grid))
rmse_lasso <- numeric(length(lambda.grid))
for (i in 1:length(lambda.grid)){
  rmse_shooting[i] <- rmse(y, t(preds)[i,])
  rmse_lasso[i] <- rmse(y, lasso_pred[i,])
}
res <- data.frame(lambda=log(lambda.grid),
                  shooting=rmse_shooting,
                  glmnet=rmse_lasso)
plot(x=res$lambda, y=res$shooting, ylab='RMSE', xlab='log(lam)', col="red")
points(x=res$lambda, y=res$glmnet, col="blue")
```



```r
plot(res_lasso, xvar="lambda", main="glmnet Lasso coefficients")
```



```r
matplot(log(shooting_coeffs[,1]), shooting_coeffs[,-1], main="Coefficients Shooting algorithm",
        type="l", xlab = "Log Lambda", ylab = "Coefficients")
```



The results of prediction for different lambdas are quite similar for our algorithm and glmnet, though glmnet, in general, has smaller rmse value. In both cases coefficients converge to zero. For our algorithm, it with bigger lambda comparing to glmnet.

```
microbenchmark("shooting" = {lassoTuning(X, y, lambda.grid)},
               "glmnet" = {glmnet(X, y, lambda = lambda.grid)}, times=10)
```

```
## Unit: milliseconds
##      expr         min          lq        mean      median          uq
##  shooting 2438.776501 2455.410601 2565.315051 2550.560750 2630.654402
##    glmnet    1.655701    1.706501    2.038091    1.975502    2.013201
##          max neval
##  2756.852101    10
##     3.361601    10
```

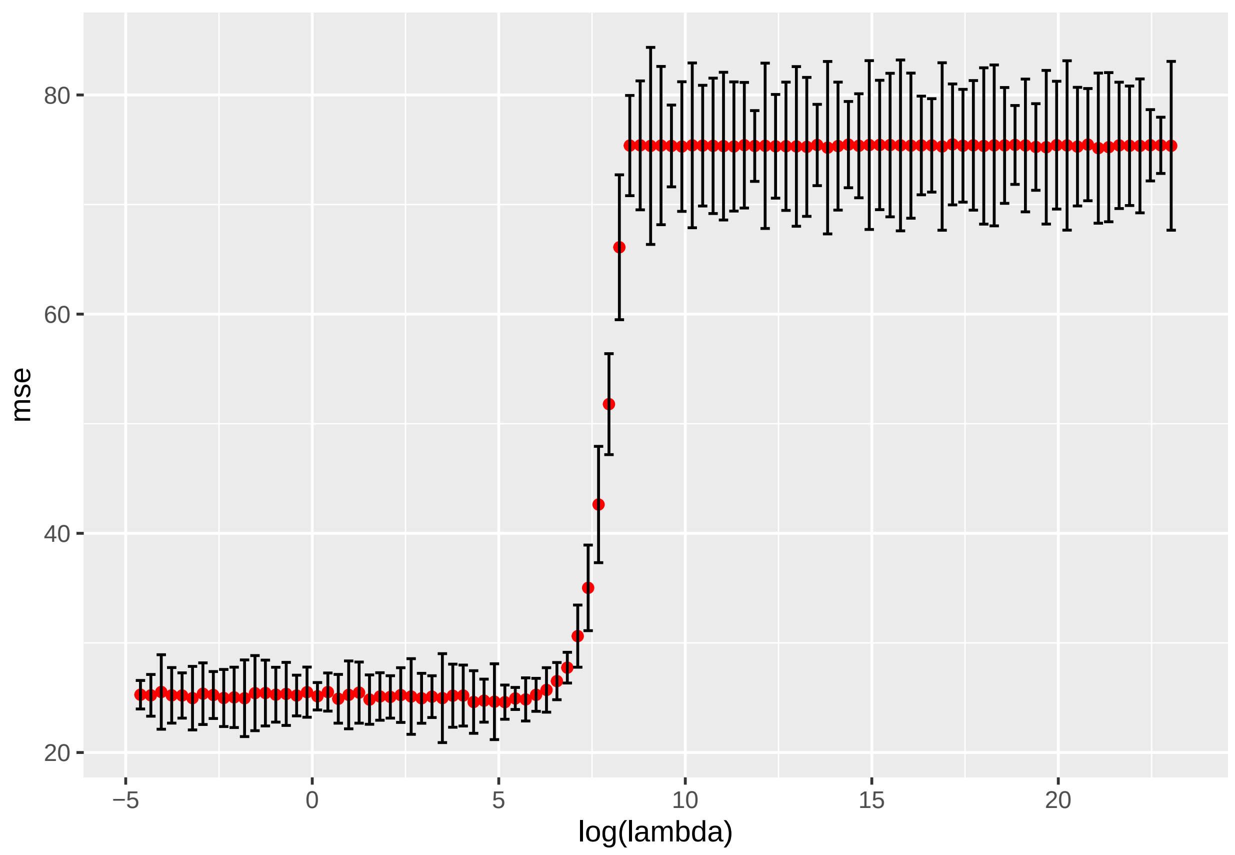
The execution time of our algorithm is much longer than original one.

1.4 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 lambda which minimizes the Root Mean Squared Error and Mean Squared Error, respectively.

```
set.seed(12208877)
my_lasso_cv <- function(X, y, lambdas, k=10){
  mse_mean <- numeric(length(lambdas))
  mse_std <- numeric(length(lambdas))
```

```r
  for (lidx in 1:length(lambdas)){
    lam <- lambdas[lidx]
    folds <- createFolds(y, k = k, list = TRUE, returnTrain = FALSE)
    mse_folds <- numeric(length(folds))

    for (i in 1:length(folds)){
      fold <- folds[[i]]

      X_train <- X[-fold,]
      X_test <- X[fold,]
      y_train <- y[-fold]
      y_test <- y[fold]

      coeffs_fold <- my_lasso(X_train, y_train, lam)
      coeffs_fold <- as.matrix(coeffs_fold)
      X_test_wintercept <- cbind(rep(1, nrow(X_test)), X_test)
      y_pred <- t(coeffs_fold) %*% t(X_test_wintercept)

      mse_folds[i] <- mean((y_test - y_pred)^2)
    }
    mse_mean[lidx] <- mean(mse_folds)
    mse_std[lidx] <- sd(mse_folds)
  }
  res_cv <- data.frame(mse = mse_mean,
                       std = mse_std,
                       lambda = lambdas)
  # plot results
  plt <- ggplot(res_cv, aes(x=log(lambda), y=mse)) +
    geom_point(col='red') +
    geom_errorbar(aes(ymin=mse-(std/2), ymax=mse+(std/2)))

  opt_l <- res_cv[res_cv$mse == min(res_cv$mse), "lambda"]
  return(list('plot' = plt, 'lambda.min' = opt_l))
}
```

```r
cv_shooting <- my_lasso_cv(X, y, lambda.grid)
cv_shooting$plot
```



```
cv_shooting$lambda.min
```

```
## [1] 174.7528
```

We can see that the algorithm works quite nicely. When lambda increases, the mse gets bigger as well. The optimal lambda for our implementation is 174. Its log is around 4.8.

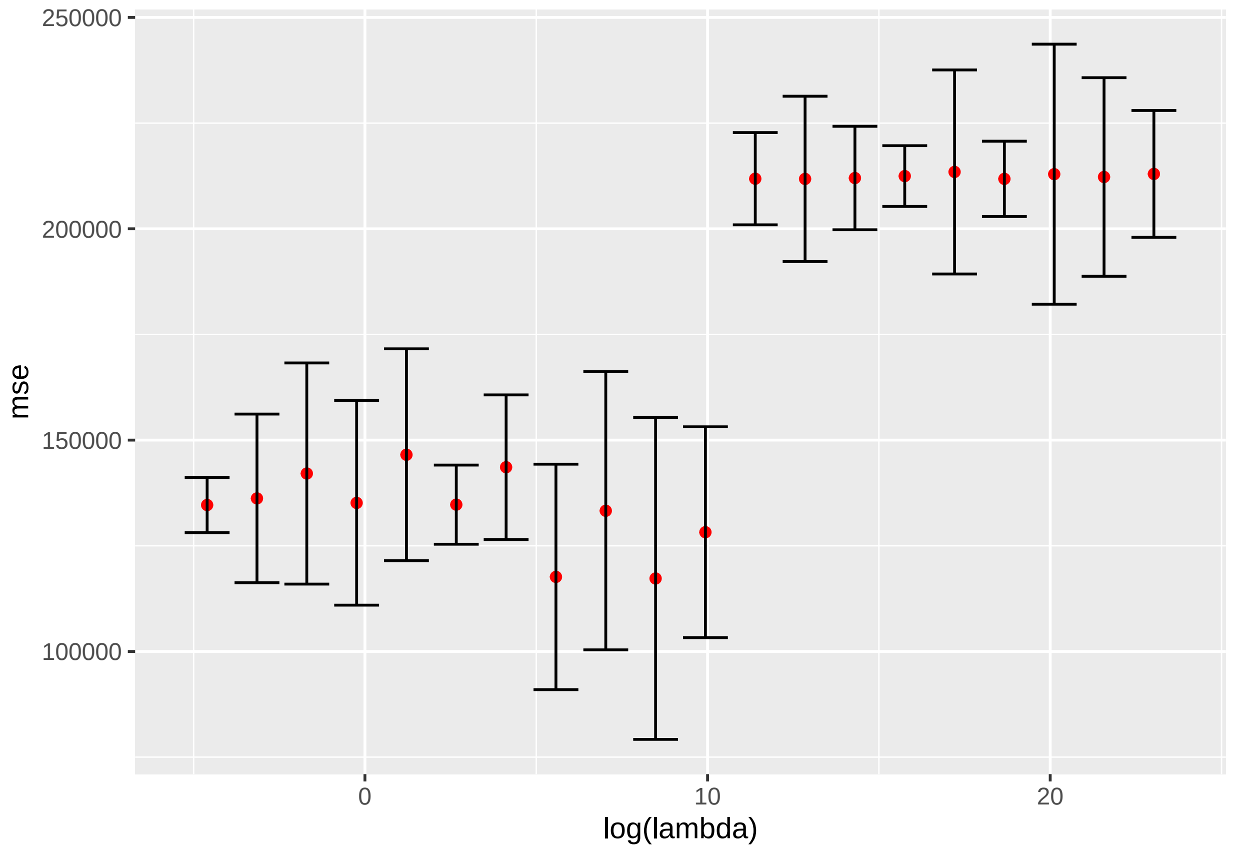
Task 2

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

```
set.seed(12208877)
data(Hitters)
data <- na.omit(Hitters)
data$League <- as.numeric(data$League) - 1
data$Division <- as.numeric(data$Division) - 1
data$NewLeague <- as.numeric(data$NewLeague) - 1
n <- nrow(data)
train_idxs <- sample(1:n, n*0.7)

X <- as.matrix(scale(data[,-19], center=TRUE, scale=TRUE))

HX_train <- as.matrix(X[train_idxs,])
HX_test <- as.matrix(X[-train_idxs,])
Hy_train <- data[train_idxs, 19]
```

```r
Hy_test <- data[-train_idxs, 19]

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

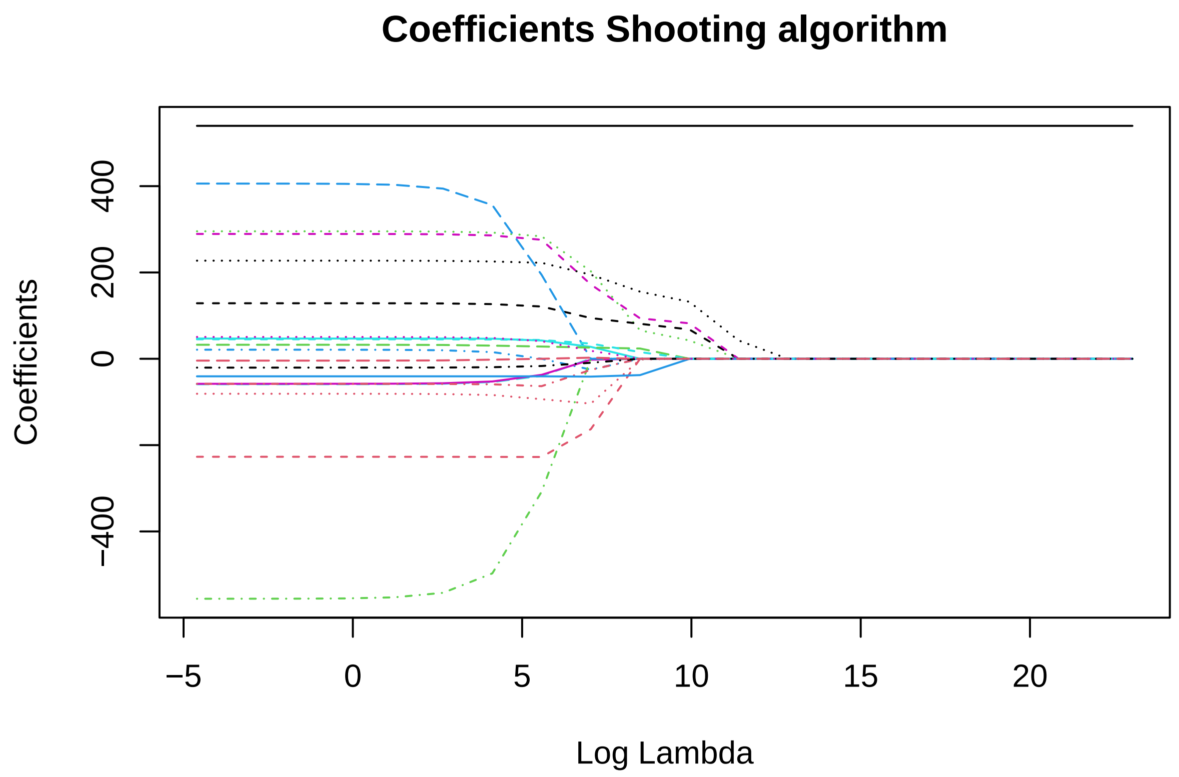
```r
lambda.grid <- 10^seq(-2,10, length=20)
cv_res <- my_lasso_cv(HX_train, Hy_train, lambda.grid, k=3)
cv_res$plot
```



```r
cv_res$lambda.min
```

```
## [1] 4832.93
```

```r
shooting_coeffs <- lassoTuning(HX_train, Hy_train, lambda.grid)
matplot(log(shooting_coeffs[,1]), shooting_coeffs[,-1], main="Coefficients Shooting algorithm",
        type="l", xlab = "Log Lambda", ylab = "Coefficients")
```



```r
coff_shooting <- my_lasso(HX_train, Hy_train, lam=cv_res$lambda.min)
test_winter <- cbind(rep(1,nrow(HX_test)), HX_test)
pred_shooting <- coff_shooting %*% t(test_winter)
rmse(Hy_test, pred_shooting)
```

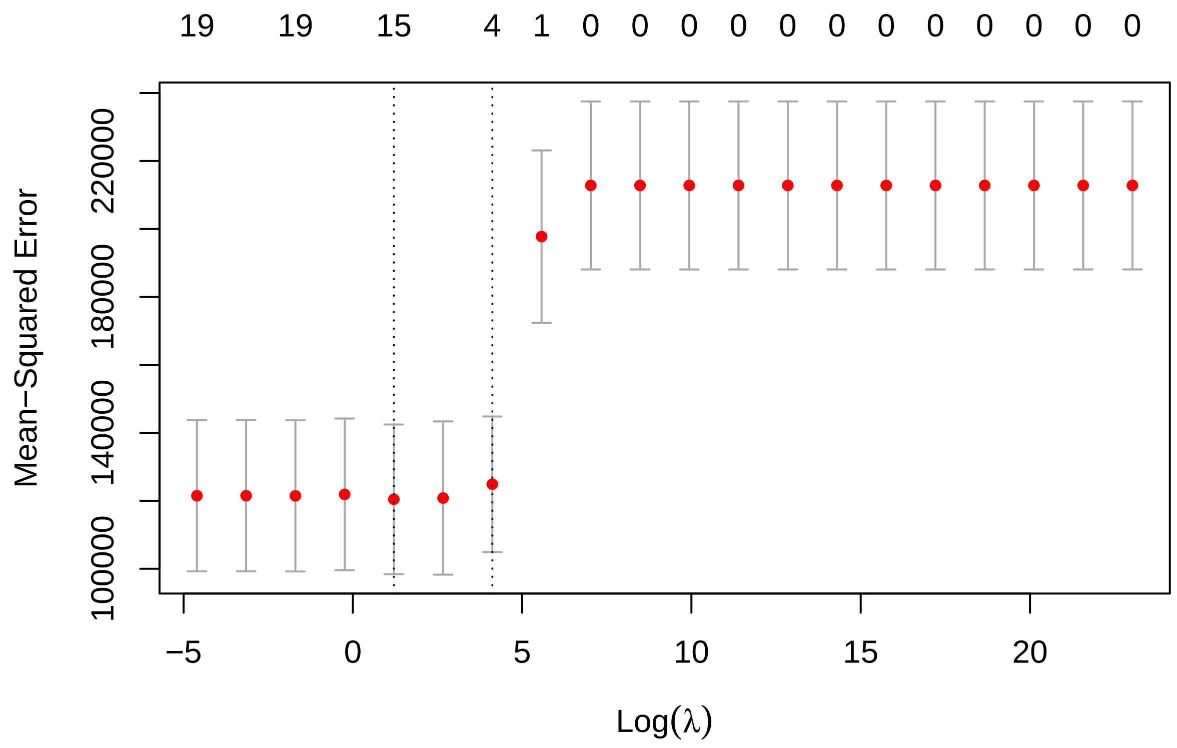
```
## [1] 357.0203
```

```r
sum(coff_shooting != 0)
```

```
## [1] 9
```

2.2 Compare your fit against the lasso implementation from glmnet.

```r
cv_glm <- cv.glmnet(HX_train, Hy_train, alpha=1, lambda = lambda.grid)
plot(cv_glm)
```



```
cv_glm$lambda.min
```

```
## [1] 3.359818
```

```
set.seed(12208877)
glm_lasso <- glmnet(HX_train, Hy_train, alpha = 1, lambda=cv_glm$lambda.min)
pred_lasso <- predict(glm_lasso, HX_test)
rmse(Hy_test, pred_lasso)
```

```
## [1] 355.2122
```

```
sum(coef(glm_lasso)!=0)
```

```
## [1] 16
```

Our model has slightly worse RMSE value, though it uses 7 less features for prediction, which in many cases would be more optimal.

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

```
set.seed(12208877)
ridge_cv <- cv.glmnet(HX_train, Hy_train, alpha=0, lambda=lambda.grid)
glm_ridge <- glmnet(HX_train, Hy_train, alpha = 0, lambda=ridge_cv$lambda.min)
df <- data.frame(HX_train, Hy_train)
ls_reg <- lm(Hy_train ~.,df)
```

2.4 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.

```r
pred_ridge <- predict(glm_ridge, HX_test)
rmse(Hy_test, pred_ridge)
```

```
## [1] 355.4794
```

```r
sum(coef(glm_ridge) != 0)
```

```
## [1] 20
```

```r
pred_ls <- predict(ls_reg, data.frame(HX_test, y=Hy_test))
rmse(Hy_test, pred_ls)
```

```
## [1] 350.8185
```

```r
sum(coef(ls_reg) != 0)
```

```
## [1] 20
```

The least squares gives the smallest rmse out of the 4 algorithms. Lasso regression from glmnet is slightly better than ridge. Even though all 4 rmse values are similar, lasso uses way less features: glmnet - 16, ours - 9. Depending on the problem, less features might be more beneficial for a huge amount of data, so the choice what model to use in the end is quite circumstantial. However, to avoid overfitting, lasso can be preferable.

Task 3

Regularized regression is a method to penalize least squares algorithm, when coefficients are too big. It is used to avoid overfitting and for feature selection, since we tend to choose the model with the least possible features without losing the correctness of performance. There are 2 additional penalty terms expressed in L1 norm for lasso regression and L2 norm for ridge. It means that they add extra values to the minimization function: ridge - sum of squared coefficients, lasso - sum of absolute coefficients. Ridge regression shrinks non-significant coefficients close to zero, while Lasso reduces them to 0. Hence, Lasso is often chosen to simplify the model with feature selection.