Comparing penalized regression estimators

Teodor Chakarov

2023-11-29

Contents


```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.3
```

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

```
## Warning: package 'Matrix' was built under R version 4.2.3
```

```
## Loaded glmnet 4.1-8
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
library(ISLR)

set.seed(12141198)
```

Task 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.

```
my_lasso <- function(X, y, coeff, lam, eps = 1e-5, maxIter = 500){
  p <- ncol(X)

  X <- cbind(1, X)
```

```
  change <- 1
  numIters <- 1

  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 - X %*% coeff + coeff_j * x_ij))

      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)
}
```

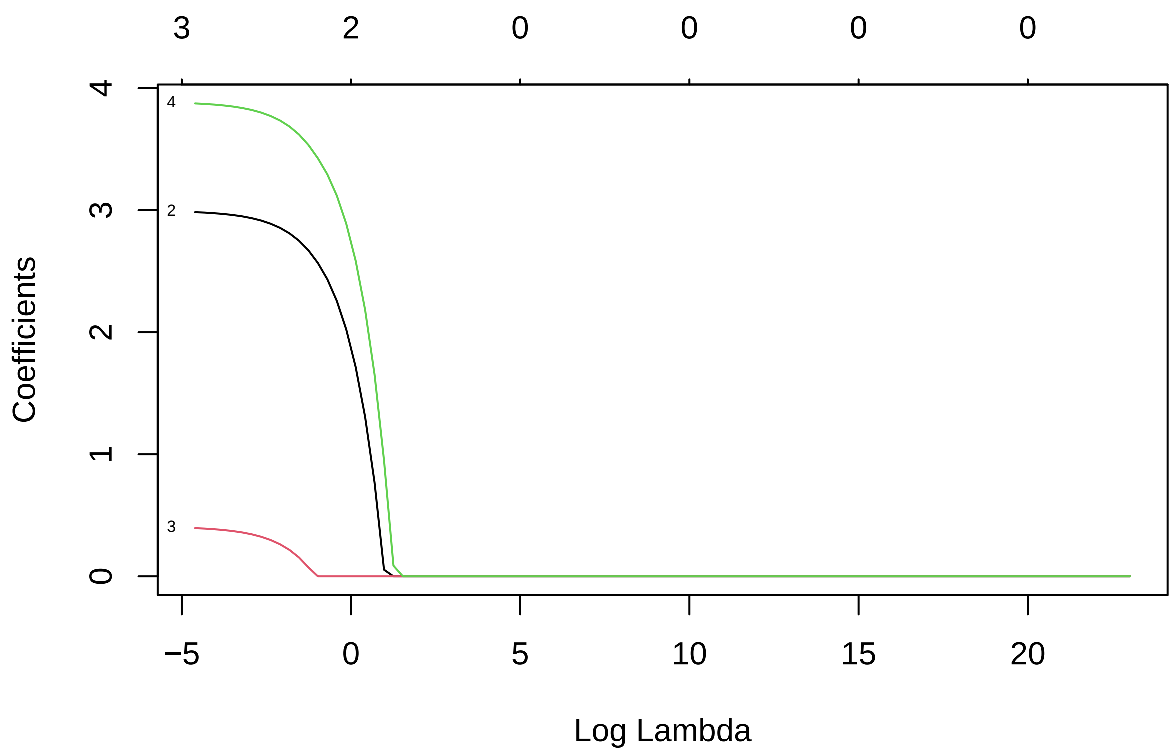
- Write a function which computes the lasso using your algorithm for a vector of lambdas and which returns the matrix of coefficients and the corresponding lambda values.

```
lasso_tuning <- function(X, y, coeff, lam_vals){
  result_vector <- c()
  for (l in lam_vals){
    coeffs <- my_lasso(X,y,coeff,l)
    result_vector <- c(result_vector,l,coeffs)
  }
  return (matrix(result_vector, nrow = length(lam_vals), ncol=dim(X)[2]+2, byrow=TRUE))
}
```

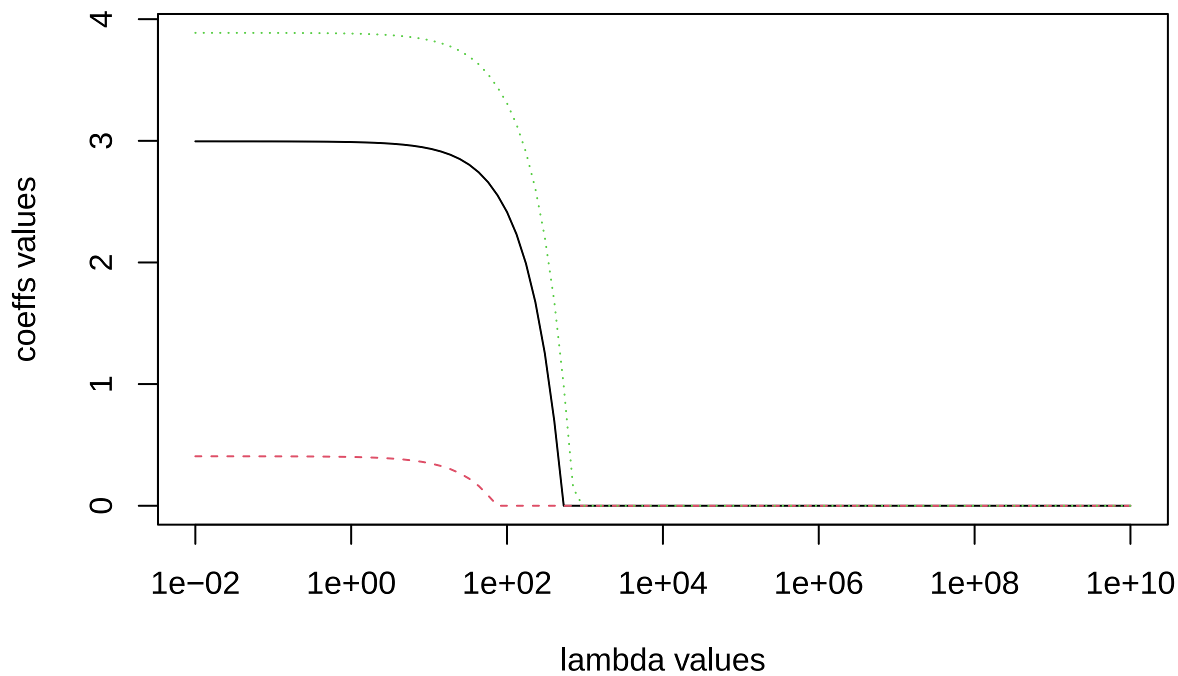
- Compare the performance and output of your functions against the lasso implementation from glmnet.

```
set.seed(12141198)

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

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

```r
laso_glm <- glmnet(X, y, alpha=1, lambda=lambda)
plot(laso_glm, xvar="lambda", label=TRUE)
```



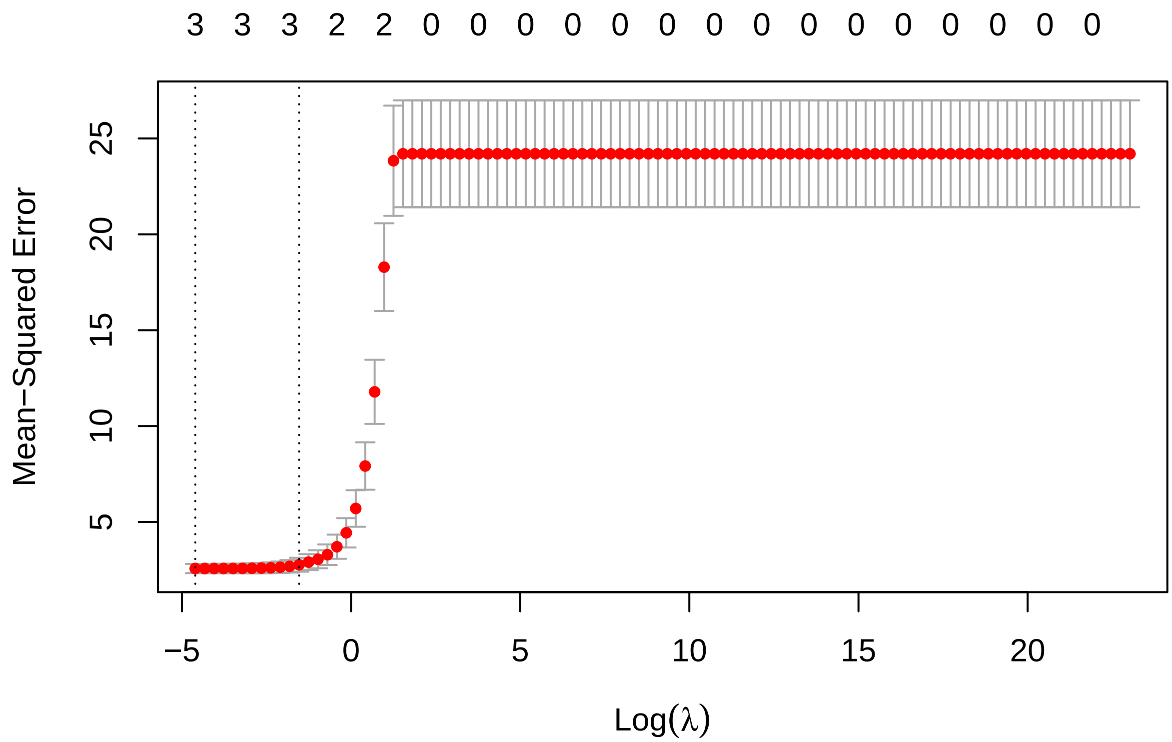

```r
X_mat <- matrix(c(x1,x2,x3),ncol=3)
lm_help <- lm(y~X_mat)
coeff_start <- as.vector(lm_help$coefficients)

lasso_res <- lasso_tuning(X_mat,y,coeff_start,lambda)

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

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



We can see that on the graphs it looks similar but the scaled lambda values are a bit different.

- 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.

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



Based on the output we can see that the smaller the lambda is the better MSE our model gives. Thus we can use hyperparamether optimization to get the least MSE.

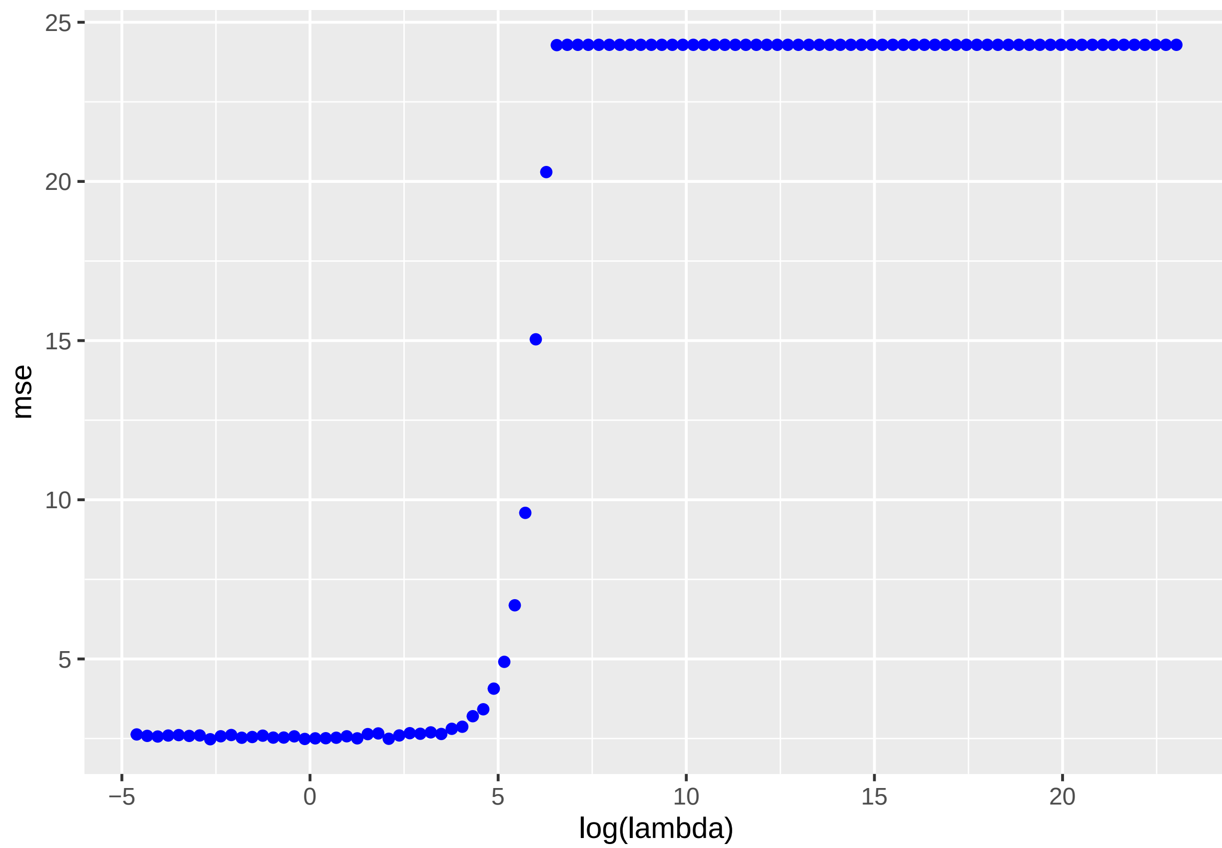
10-fold cross validation:

```
set.seed(12141198
         )
lasso_cross_validation <- function(X, y, initial_coeffs, lambda_values){

  num_samples <- dim(X)[1]
  num_features <- dim(X)[2]

  # Result vectors
  mse_mean <- c()
  mse_std <- c()

  for (lambda in lambda_values){
    mse <- c()
    folds <- split(1:num_samples, sample(rep(1:10, length.out = num_samples)))

    for (fold in folds){
      X_train <- X[-fold,]
      X_test <- X[fold,]
      y_train <- y[-fold]
      y_test <- y[fold]

      coeffs_cv <- as.matrix(my_lasso(X_train, y_train, initial_coeffs, lambda))
```

```r
      y_pred <- t(coeffs_cv[-1]) %*% t(X_test) + coeffs_cv[1]
      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" = lambda_values)


  plt <- ggplot(cv_lasso, aes(x = log(lambda), y = mse))
  plt <- plt + geom_point(col = 'blue')

  min_mse <- min(cv_lasso$mse)
  opt_lambda <- cv_lasso[cv_lasso$mse == min_mse, "lambda"]

  return(list('plot' = plt, 'lambda_minMSE' = opt_lambda))
}

cv_lasso_result <- lasso_cross_validation(X_mat, y, coeff_start, lambda)
print('Output:')
```

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

```r
print(cv_lasso_result$plot)
```




```
print(paste0('Lambda which minimizes the Mean Squared Error: ', cv_lasso_result$lambda_minMSE))
```

```
## [1] "Lambda which minimizes the Mean Squared Error: 0.0705480231071865"
```

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)
head(data)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
## -Al Newman          185   37     1   23   8    21     2    214    42      1
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Alan Ashby         321  414    375      N        W     632      43     10
```

```
## -Alvin Davis         224  266    263      A        W     880      82      14
## -Andre Dawson        828  838    354      N        E     200      11       3
## -Andres Galarraga     48   46     33      N        E     805      40       4
## -Alfredo Griffin     501  336    194      A        W     282     421      25
## -Al Newman            30    9     24      N        E      76     127       7
##                    Salary NewLeague
## -Alan Ashby         475.0         N
## -Alvin Davis        480.0         A
## -Andre Dawson       500.0         N
## -Andres Galarraga    91.5         N
## -Alfredo Griffin    750.0         A
## -Al Newman           70.0         A
```

Data preparation:

```
data$League <- as.numeric(factor(data$League))
data$Division <- as.numeric(factor(data$Division))
data$NewLeague <- as.numeric(factor(data$NewLeague))
y <- data$Salary
data$Salary <- NULL

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

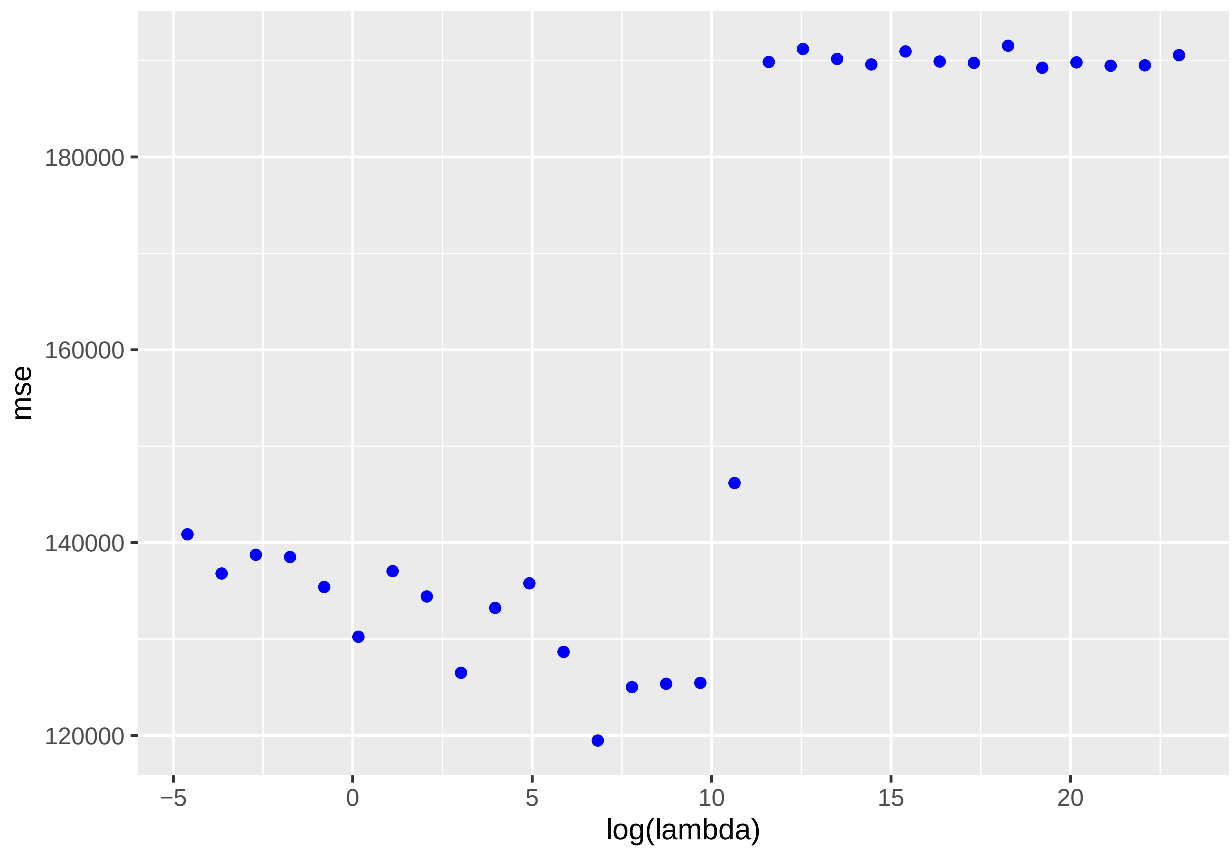
Train test split:

```
train_test_split <- function(data,y, size = 0.7){
  n <- nrow(data)
  train_sample <- sample(1:n,n*size)

  x_train <- data[train_sample,]
  x_test <- data[-train_sample,]
  y_train <- y[train_sample]
  y_test <- y[-train_sample]

  return(list(x_train = x_train, x_test = x_test, y_train = y_train, y_test = y_test))

}

split_data <- train_test_split(data,y, size = 0.7)

x_train <- split_data$x_train
x_test <- split_data$x_test
y_train <- split_data$y_train
y_test <- split_data$y_test
```

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

```
lm_help <- lm(y_train~x_train)
coeff_start <- as.vector(lm_help$coefficients)

lambda_1<- 10^seq(-2,10, length=30)
lasso_cross_validation(x_train,y_train,coeff_start,lambda_1)
```

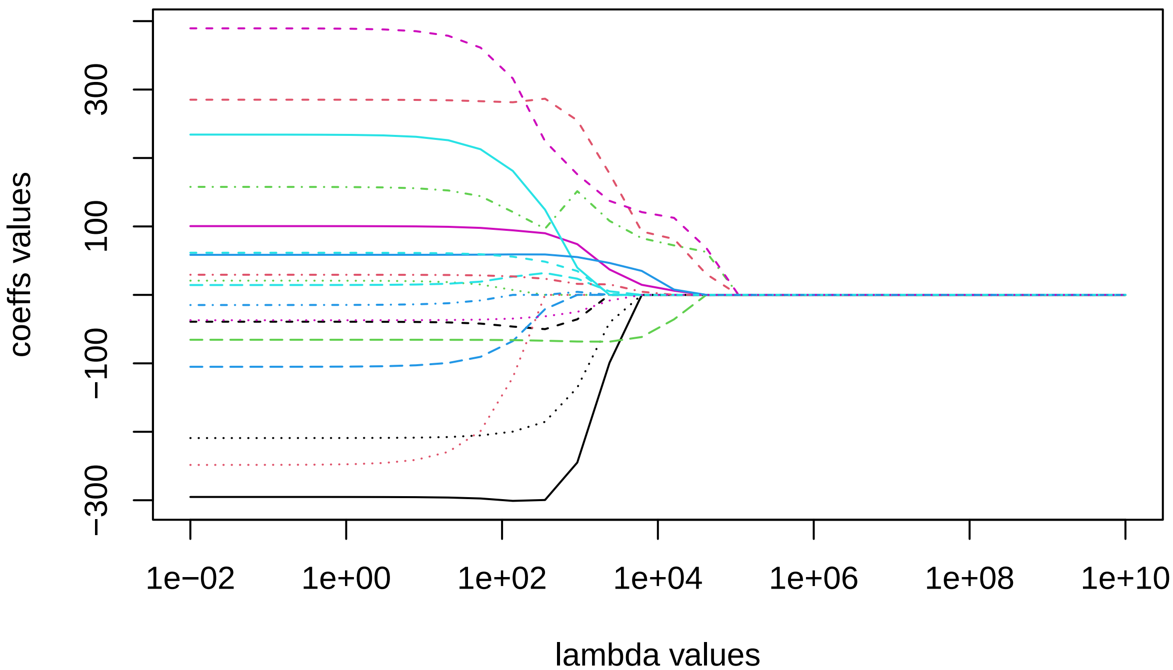
```
## $plot
```




```
##
## $lambda_minMSE
## [1] 923.6709
```

Best lambda identified by cross validation function is: 5,25

```r
lambda_1 <- 10^seq(-2,10, length=30)

model_res <- lasso_tuning(x_train,y_train,coeff_start,lambda_1)

lambdas <- model_res[1:30,1:1]
coeffs <- model_res[1:30,3:20]

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

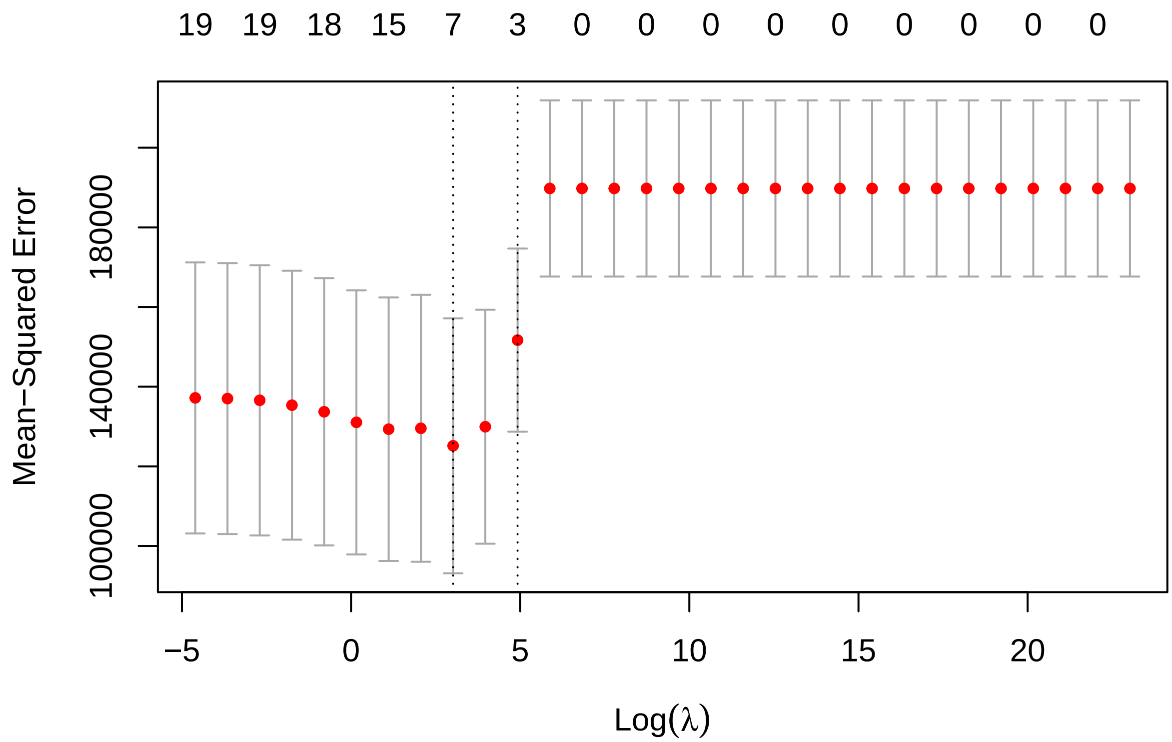


As lambda increases, the magnitude of the coefficients tends to shrink towards zero. This is indicative of the regularization effect where increasing the penalty term in the cost function forces the coefficients to become smaller to avoid overfitting.

At higher lambda values, many of the coefficients are exactly zero, which implies that the model is performing feature selection. Only the most important features are retained with non-zero coefficients.

- Compare your fit against the lasso implementation from glmnet.

```
cv_data <- cv.glmnet(x_train, y_train, alpha = 1, lambda = lambda_1)
plot(cv_data)
```



The coefficients are shrunk earlier. For smaller lambda values.

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

```
glm_lasso <- cv.glmnet(x_train, y_train, alpha = 1)

glm_ridge <- cv.glmnet(x_train, y_train, alpha = 0)

least_squares <- lm(y_train ~ x_train)
```

- 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.

```
# prediction of lasso
models <- list(
  list(model = glm_lasso, name = "Lasso Regression"),
  list(model = glm_ridge, name = "Ridge Regression"),
  list(model = least_squares, name = "Least Squares Regression")
)

# Create a loop to iterate through the models
for (regression in models) {
  model <- regression$model
  model_name <- regression$name

  # Predict using the model
```

```r
  pred <- predict(model, newx = x_test)

  # Calculate MSE
  mse <- mean((y_test - pred)^2)

  # Print the MSE along with the model name
  cat(paste("MSE of", model_name, ":", mse, "\n"))
}
```

```
## MSE of Lasso Regression : 185095.688997585
## MSE of Ridge Regression : 176519.61222883

## Warning in y_test - pred: longer object length is not a multiple of shorter
## object length

## MSE of Least Squares Regression : 373265.101403051
```

We can see that Ridge regression has the lowest MSE and the LSR has the worst of them. Lasso uses much less variables than ridge so it might lead to a better explainable model.

Task 3

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

Regularized regression is a technique used in machine learning to find the best-fitting line or model for a dataset while penalizing large coefficients. The goal is to produce a simpler model that generalizes better to new data. This is done by adding a penalty term to the traditional least squares loss function, which encourages the model's parameters to have smaller absolute values.

Ridge regression: In ridge regression, the shrinkage factor is added to each parameter multiplicatively. Specifically, the updated value of the i-th parameter, where λ is the shrinkage parameter, X_i is the i-th feature matrix, β_0 is the intercept, β_1 is the slope, and σ_i is the standard deviation of the error terms. *Lasso regression*: In Lasso regression, the shrinkage factor is added to each parameter exponentially. Specifically, the updated value of the i-th parameter, where λ is the shrinkage parameter, X_i is the i-th feature matrix, β_0 is the intercept, β_1 is the slope, and β is the vector of all betas. Here, the sign function takes the value of 1 if $X_i^T \beta > 0$, and -1 otherwise.

For ridge regression, those coefficients will be shrunk towards zero regardless of their actual importance, whereas for Lasso regression, only the coefficients with non-zero variance will be shrunk. As a result, Lasso regression tends to produce more sparse solutions than ridge regression.