Ridge Regression

Gregoire Gasparini, Aurora Hofman, Sarah Musiol, Beatriu Tort

25 de febrero de 2020

For the Assignment about Ridge Regression, we compute a function to choose a penalization parameter. The theory about Ridge Regression is used to write basic functions. In order to test these functions the Boston Housing data is used with it.

Alternatively, we proposed to use a package that deals with Ridge Regression in the background.

Choosing the penalization parameter

Function for choosing the penalization parameter

```
prostate <- read.table("prostate_data.txt", header=TRUE, row.names = 1)

train.sample <- which(prostate$train==TRUE) ##separate trainingsdata from testdata

val.sample <- which(prostate$train==FALSE)


Y_t <- scale( prostate$lpsa[train.sample], center=TRUE, scale=FALSE) ## center but not scale for respon

X_t <- scale( as.matrix(prostate[train.sample,1:8]), center=TRUE, scale=TRUE) ##scale and center for re

Y_val <- scale( prostate$lpsa[val.sample], center=TRUE, scale=FALSE) ## center but not scale for respon

X_val <- scale( as.matrix(prostate[val.sample,1:8]), center=TRUE, scale=TRUE) ##scale and center for re


#predictors

p <- dim(X_t)[2]
n <- dim(X_t)[1]

XtX <- t(X_t)%*%X_t

d2 <- eigen(XtX,symmetric = TRUE, only.values = TRUE)$values #eigenvalues of xtx

(cond.number <- sqrt(max(d2)/min(d2)))
```

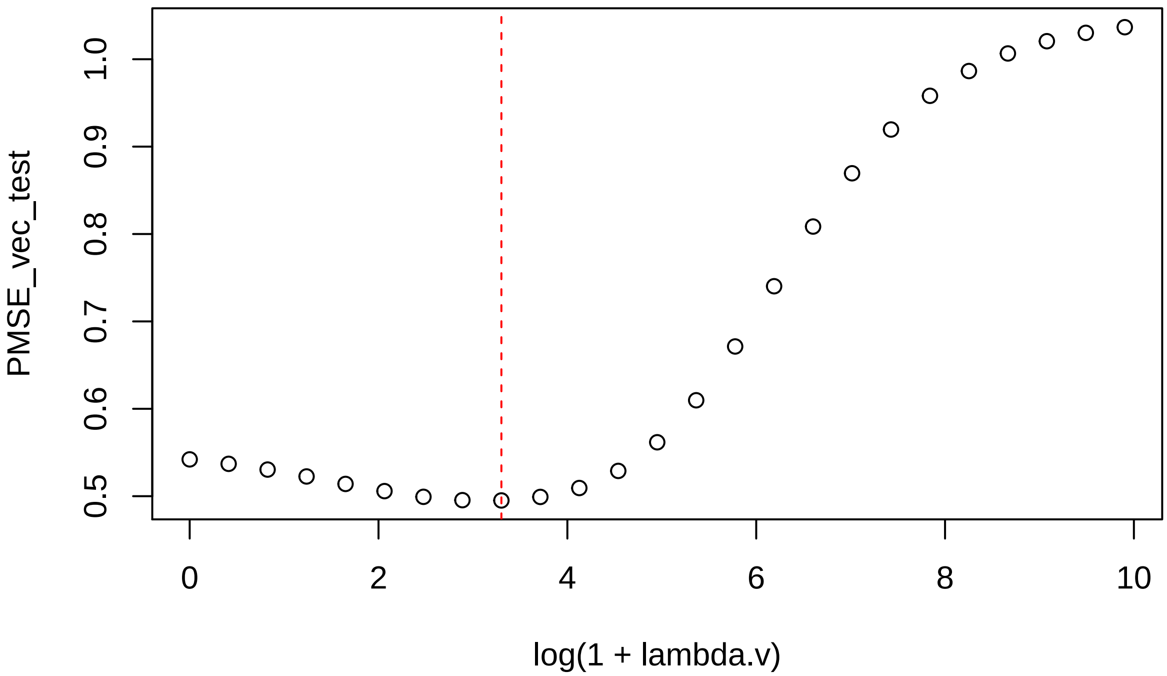
```
## [1] 4.435608
```

```
lambda.max = 2e4

n_lambdas <- 25 ## look at 25 different values
```

```r
lambda.v <- exp(seq(0,log(lambda.max+1),length=n_lambdas))-1 #lambda vector

n_val <- length(Y_val)
```

```r
PMSE_vs <- function(X_t, Y_t, X_val, Y_val, lambda){

  p <- dim(X_t)[2]
  n_lambdas <- length(lambda)
  XtX <- t(X_t)%*%X_t
  PMSE_vec <- vector("numeric", length = n_lambdas)

  for(l in 1:n_lambdas){
    lambda <- lambda.v[l]
    beta_hat <- solve(XtX + lambda*diag(1,p)) %*% t(X_t) %*% Y_t
    #y_hat = X %*% beta_hat
    m_hat_vec <- vector("numeric", length = n_val)

    for (n in 1:n_val){
      m_hat_vec[n] <- (Y_val[n]-(X_val[n,]%*%beta_hat))^2

    }

    PMSE_vec[l]<- sum(m_hat_vec)/n_val



  }

  return (PMSE_vec)

}
```

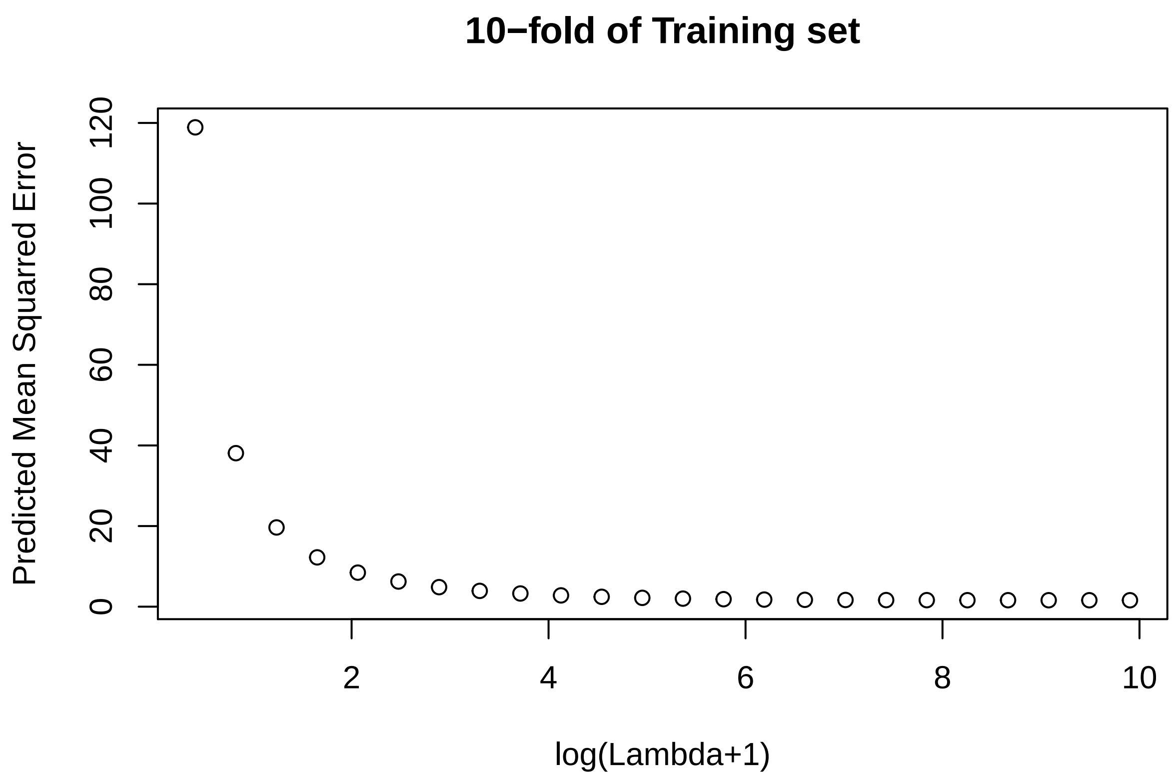
```r
PMSE_vec_test <- PMSE_vs(X_t, Y_t, X_val, Y_val, lambda.v)

lambda.CV <- lambda.v[which.min(PMSE_vec_test)]

plot(log(1+lambda.v), PMSE_vec_test)

abline(v=log(1+lambda.CV),col=2,lty=2)
```



Function for Implementing the Ridge Regression penalization parameter

```
PMSE_k_fold <- function(X_t, Y_t, lambda.v, k=10){

  n_X_t <- dim(X_t)[1]
  p_X_t <- dim(X_t)[2] #length of columns
  n_subset <- as.integer((n_X_t)/k)+ 1
  group <- rep(seq(1,k), times = n_subset)
  group <- group[1:n_X_t]
group_random <- sample(group)
X_t_group <- cbind(X_t, group_random)
Y_t_group <- cbind(Y_t, group_random)

#now we can start:

PMSE <- list()

for (la in 1:n_lambdas){

  group_random <- sample(group)

  X_t_group <- cbind(X_t, group_random)

  Y_t_group <- cbind(Y_t, group_random)

  #now we can start:

  PMSE <- list()

  for (la in 1:n_lambdas){
```

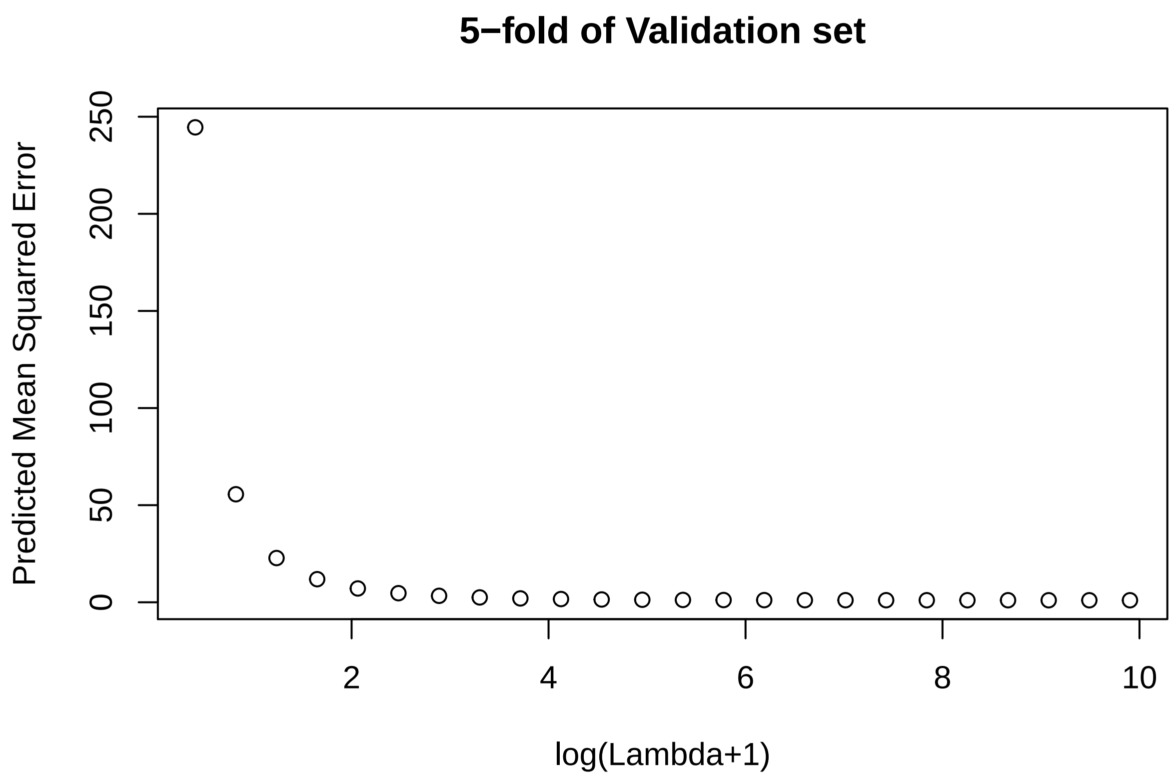
```
    lambda <- lambda.v[la]

    y_hat <- list()
    beta <- list()
    h <- list()
    y <- list()

    for (l in 1:k){

      new_X_t_val <- subset(X_t_group, group_random==l)[ ,1:p_X_t]
      new_X_t_test <- subset(X_t_group, group_random!=l)[ ,1:p_X_t]

      new_Y_t_val <- subset.matrix(Y_t_group, group_random==l)[,1]
      new_Y_t_test <- subset.matrix(Y_t_group, group_random!=l)[,1]

      p_new <- dim(new_X_t_test)[2]
      p_new_v <- dim(new_X_t_val)[2]

      beta[[l]] <- solve(t(new_X_t_test)%*%new_X_t_test + lambda*diag(1,p_new))%*% t(new_X_t_test)%*%(ne

      H_val <- new_X_t_val%*%solve(t(new_X_t_val)%*%new_X_t_val + (lambda+1e-13)*diag(1,p_new_v))%*% t(n

      # singular matrix for lambda = 0 -> trick: add a very small number

      y_hat[[l]] <-  (new_X_t_val)%*%beta[[l]]
      h[[l]] <- diag(H_val)
      y[[l]] <- new_Y_t_val


  }

  y_hat <- c(do.call(rbind, y_hat))
  beta <- c(do.call(cbind, beta))
  h <- c(do.call(rbind, h))
  y <- c(do.call(rbind, y))

  PMSE[[la]] <- 1/n_X_t * sum(((y-y_hat)/(1-h))^2)

  }

}

return(PMSE)

}



PMSE <- PMSE_k_fold(X_t = X_t, Y_t = Y_t, lambda.v = lambda.v, k = 10)

plot(log(lambda.v[-1]+1), PMSE[-1], ylab = "Predicted Mean Squarred Error", xlab = "log(Lambda+1)", mai
```



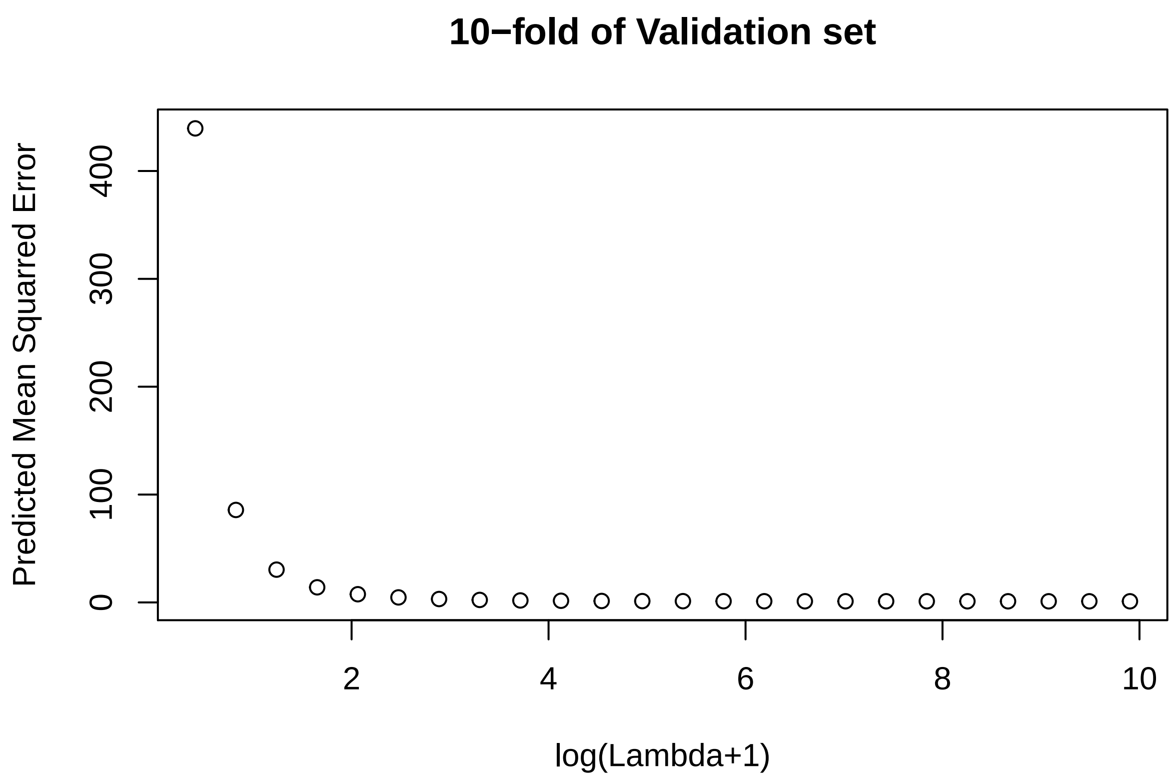
In the graphic of the PMSE of 10-fold of Training set we have a negative exponential distribution. We suspect that there is some problem in the code and this is not working normally but we weren't able to find it. Theorically we should have a graphic with a lower scale and that goes down and then up again so we have a minimum lambda that would be the optimum one.

Comparison between 5-fold and 10-fold

```
PMSE_val_5 <- PMSE_k_fold(X_t = X_val, Y_t = Y_val, lambda.v = lambda.v, k = 5)

plot(log(lambda.v[-1]+1), PMSE_val_5[-1], ylab = "Predicted Mean Squarred Error", xlab = "log(Lambda+1)
```

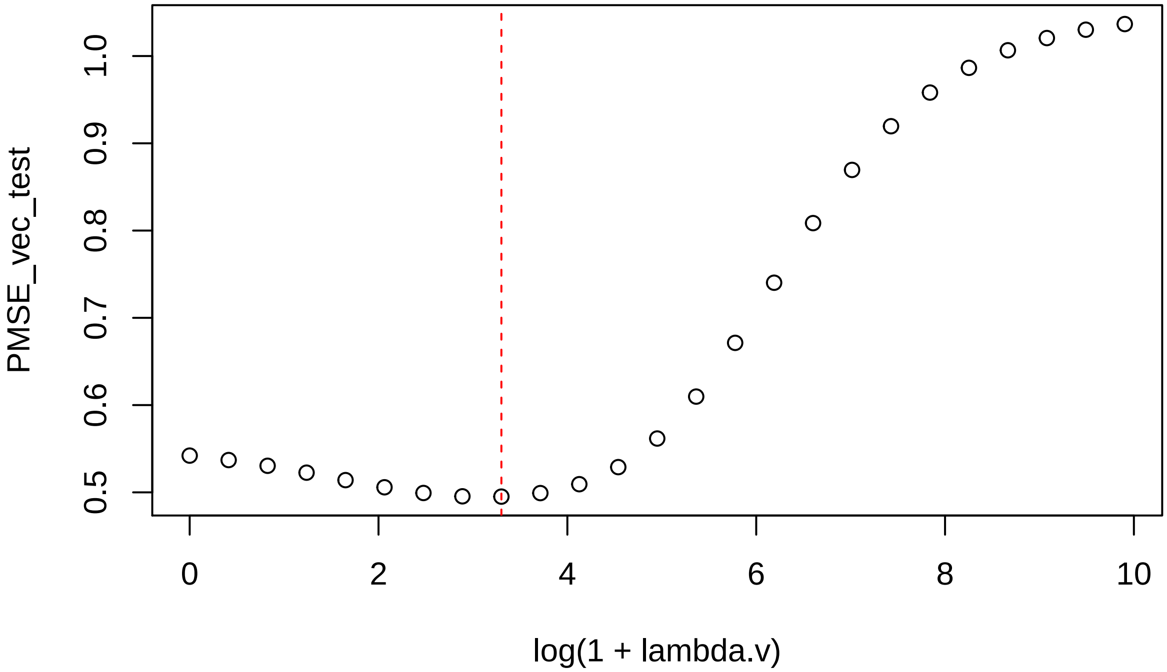



```
PMSE_val_10 <- PMSE_k_fold(X_t = X_val, Y_t = Y_val, lambda.v = lambda.v, k = 10)
plot(log(lambda.v[-1]+1), PMSE_val_10[-1], ylab = "Predicted Mean Squarred Error", xlab = "log(Lambda+1)
```

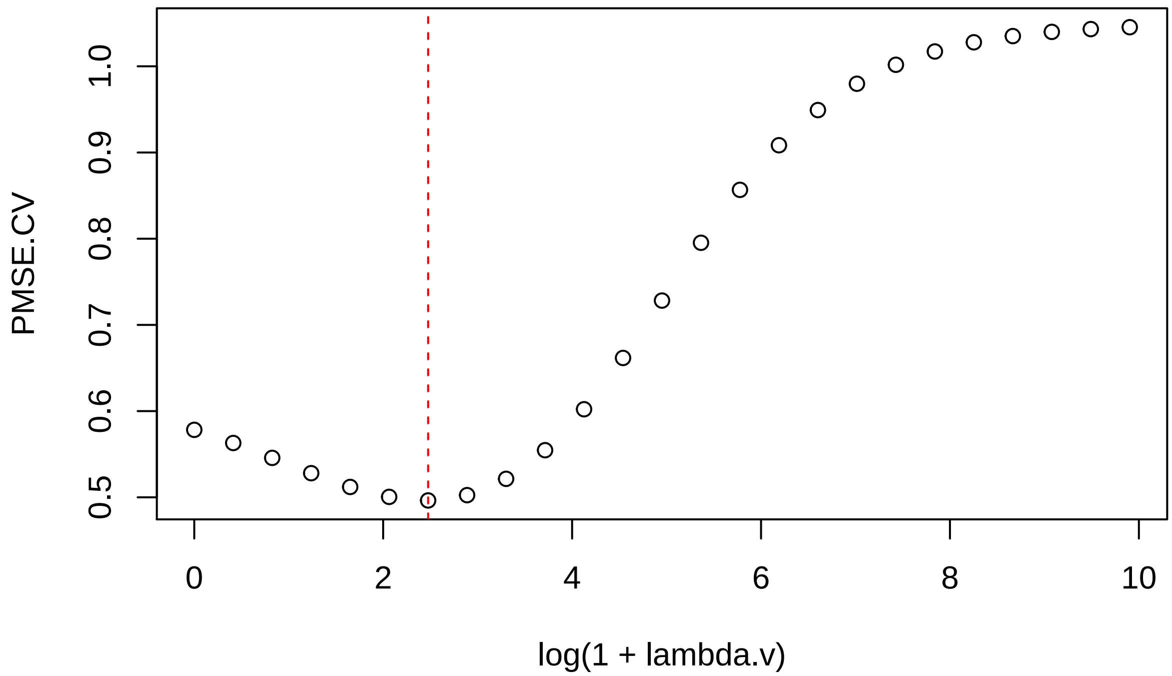



```
#I am not sure about if the one below is the leave-one-out cross validation or not.
```

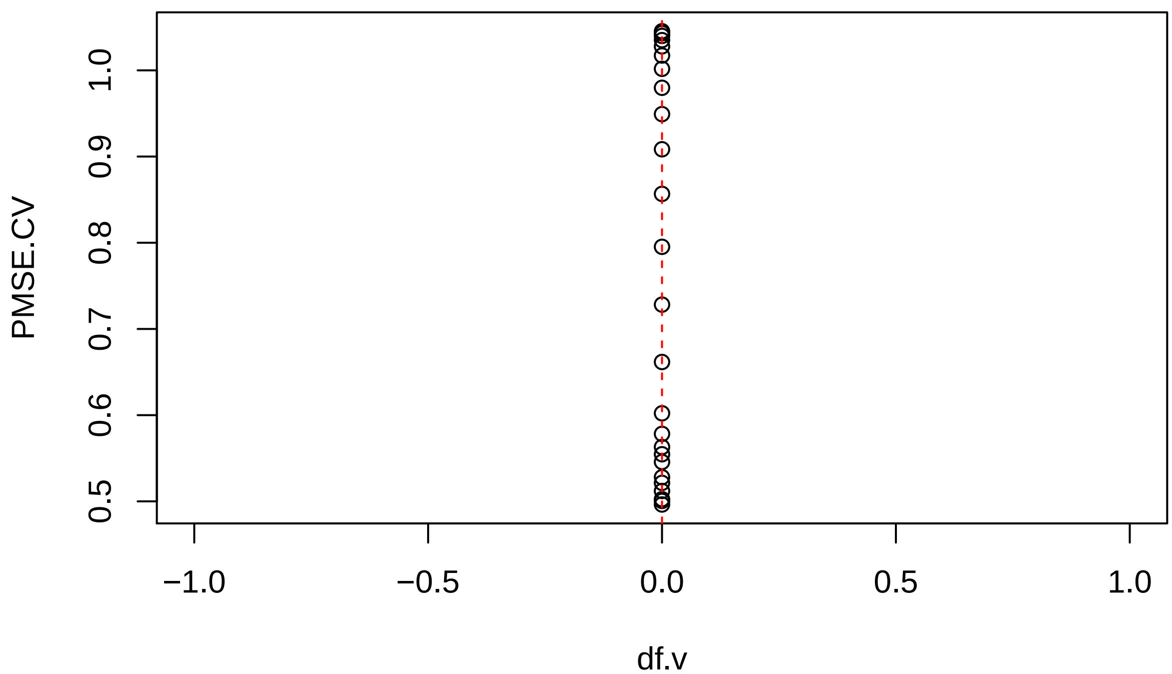
```r
plot(log(1+lambda.v), PMSE_vec_test)
abline(v=log(1+lambda.CV),col=2,lty=2)
```




```r
## leave-one-out cross validation

df.v <- numeric(n_lambdas)

PMSE.CV <- numeric(n_lambdas)
for (l in 1:n_lambdas){
  lambda <- lambda.v[l]
  PMSE.CV[l] <- 0
  for (i in 1:n_val){
#    m.Y.i <- mean(Y[-i])
    m.Y.i <- 0
    X.i <- X_val[-i,]
    Y.i <- Y_val[-i]-m.Y.i
    Xi <- X_val[i,]
    Yi <- Y_val[i]
    beta.i <- solve(t(X.i)%*%X.i + lambda*diag(1,p)) %*% t(X.i) %*% Y.i
    hat.Yi <- Xi %*% beta.i + m.Y.i
    PMSE.CV[l] <- PMSE.CV[l] + (hat.Yi-Yi)^2
  }
  PMSE.CV[l] <- PMSE.CV[l]/n_val
}
lambda.CV <- lambda.v[which.min(PMSE.CV)]
df.CV <- df.v[which.min(PMSE.CV)]

plot(log(1+lambda.v), PMSE.CV)
abline(v=log(1+lambda.CV),col=2,lty=2)
```

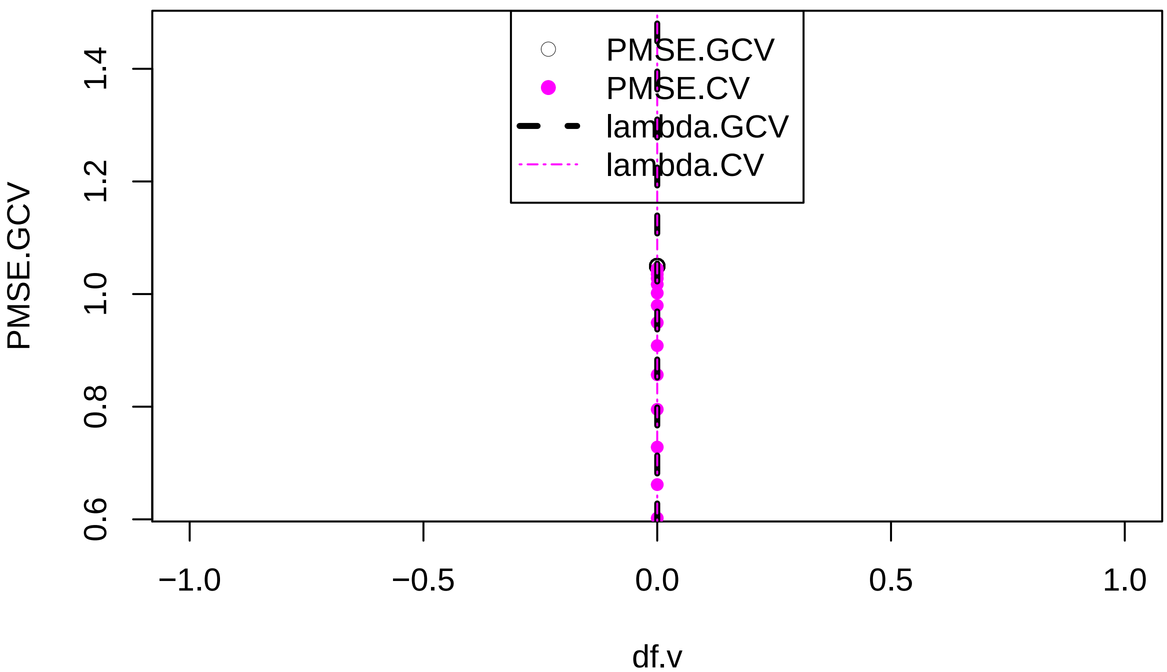



```
plot(df.v, PMSE.CV)
abline(v=df.CV,col=2,lty=2)
```




```
## Generalized Cross Validation (GCV)

beta.path <- matrix(0,nrow=n_lambdas, ncol=p)
diag.H.lambda <- matrix(0,nrow=n_lambdas, ncol=n_val)


PMSE.GCV <- numeric(n_lambdas)
for (l in 1:n_lambdas){
```

```
  lambda <- lambda.v[l]
  hat.Y <- X_val %*% beta.path[l,]
  nu <- sum(diag.H.lambda[l,])
  PMSE.GCV[l] <- sum( ((Y_val-hat.Y)/(1-nu/n_val))^2 )/n_val
}
lambda.GCV <- lambda.v[which.min(PMSE.GCV)]
df.GCV <- df.v[which.min(PMSE.GCV)]

PMSE.CV.H.lambda <- numeric(n_lambdas)
lambda.CV.H.lambda <- lambda.v[which.min(PMSE.CV.H.lambda)]
df.CV.H.lambda <- df.v[which.min(PMSE.CV.H.lambda)]

plot(df.v, PMSE.GCV)
points(df.v, PMSE.CV,col=6,pch=19,cex=.75)
abline(v=df.GCV,col=1,lty=2,lwd=3)
abline(v=df.CV.H.lambda,col=6,lty=6)
legend("top",c("PMSE.GCV","PMSE.CV","lambda.GCV","lambda.CV"),
       pch=c(1,19,NA,NA),lty=c(0,0,2,6),lwd=c(0,0,3,1),col=c(1,6,1,6))
```



Ridge Regression for the Boston Housing data

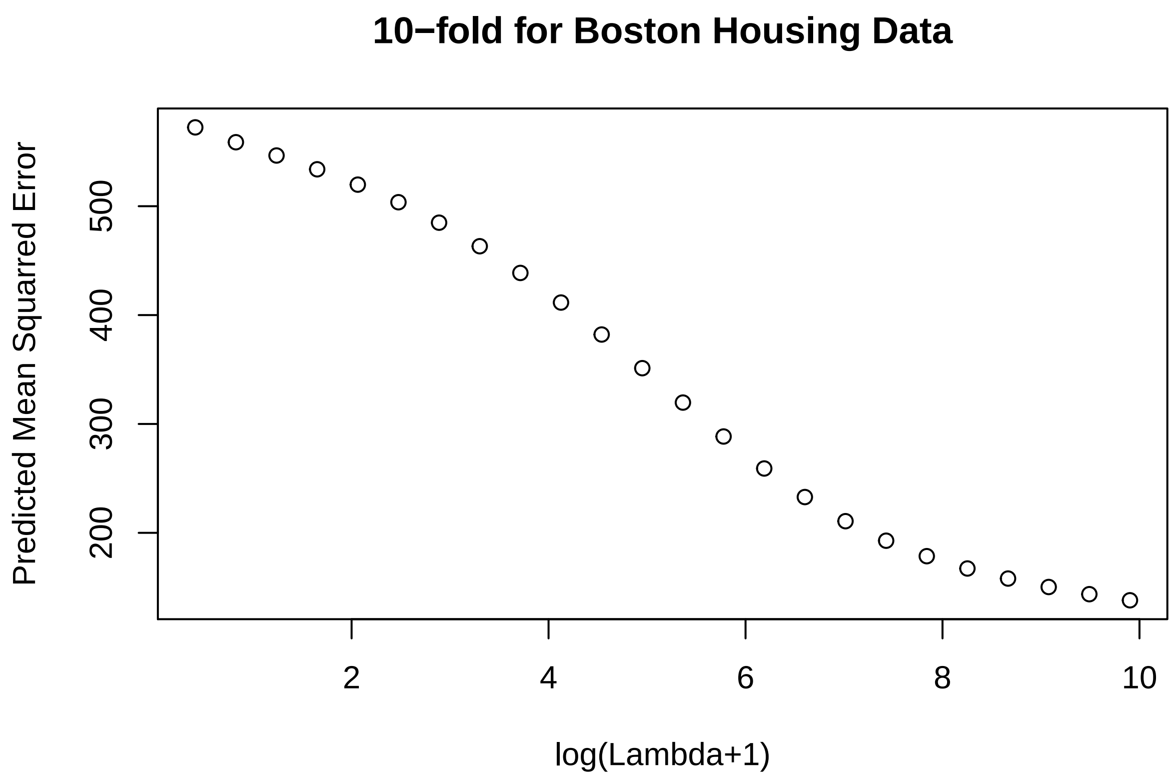
Loading the (corrected) Boston Housing data

```
library(MASS)

data(Boston)

help(Boston)
```

```r
boston <- load("boston.Rdata")

boston <- boston.c

# cv.glmnet cannot handel factors -> "CHAS" is a factor
boston.c$CHAS <- as.numeric(boston.c$CHAS)

response <- "MEDV"

explanatory <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B"
```

Using obtained functions for Boston Housing data

```r
PMSE_10 <- PMSE_k_fold(X_t = as.matrix(boston.c[, explanatory]), Y_t = boston.c$MEDV, lambda.v = lambda

plot(log(lambda.v[-1]+1), PMSE_10[-1], ylab = "Predicted Mean Squarred Error", xlab = "log(Lambda+1)", m
```



Alternative solution for Ridge Regression for the Boston Housing data

There exists a package called 'glmnet' that deals with elastic nets. Specifying $alpha = 0$ Ridge Regression is applied on the data.

```r
#install.packages("glmnet")

library(glmnet)
```

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

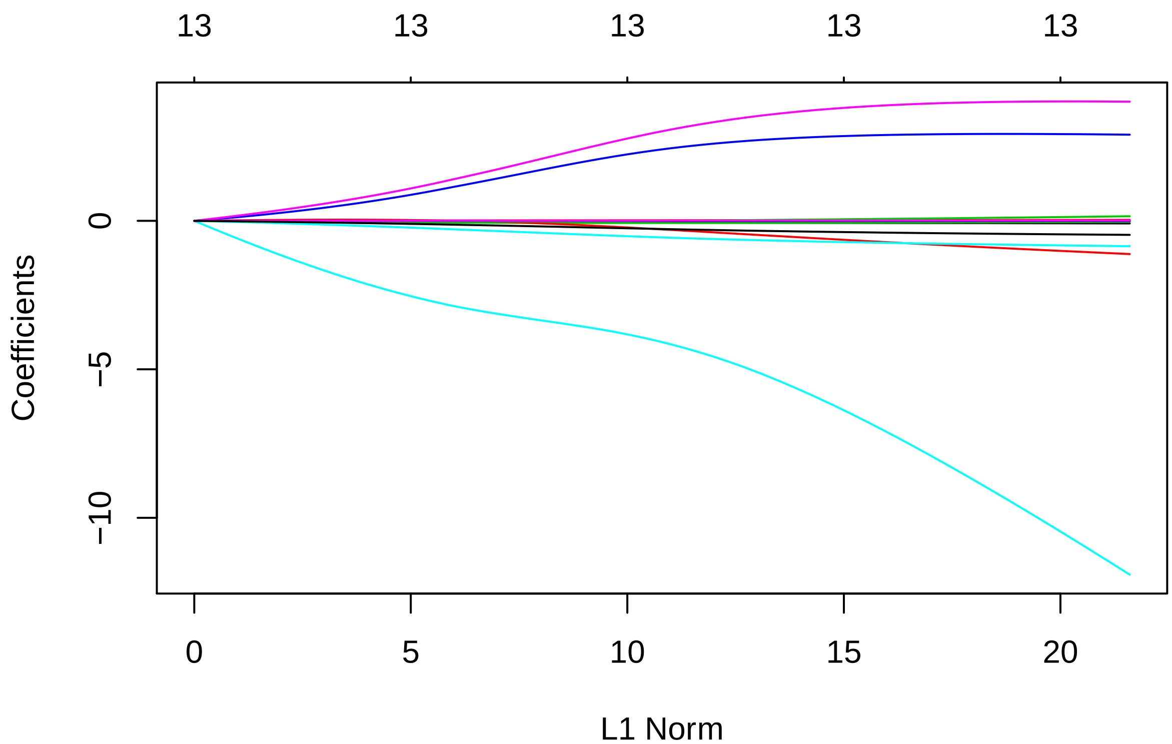
```
## Loaded glmnet 3.0-2
```

```r
(ridge <- glmnet(y = boston.c$MEDV, x = as.matrix(boston.c[, explanatory]), alpha = 0))
```

```
##
## Call:  glmnet(x = as.matrix(boston.c[, explanatory]), y = boston.c$MEDV,      alpha = 0)
##
##     Df    %Dev Lambda
## 1   13 0.00000 6778.0
## 2   13 0.00792 6176.0
## 3   13 0.00868 5627.0
## 4   13 0.00952 5127.0
## 5   13 0.01043 4672.0
## 6   13 0.01143 4257.0
## 7   13 0.01253 3878.0
## 8   13 0.01372 3534.0
## 9   13 0.01503 3220.0
## 10  13 0.01646 2934.0
## 11  13 0.01802 2673.0
## 12  13 0.01972 2436.0
## 13  13 0.02159 2219.0
## 14  13 0.02362 2022.0
## 15  13 0.02583 1843.0
## 16  13 0.02824 1679.0
## 17  13 0.03087 1530.0
## 18  13 0.03372 1394.0
## 19  13 0.03683 1270.0
## 20  13 0.04020 1157.0
## 21  13 0.04386 1054.0
## 22  13 0.04783  960.7
## 23  13 0.05213  875.4
## 24  13 0.05678  797.6
## 25  13 0.06180  726.7
## 26  13 0.06721  662.2
## 27  13 0.07304  603.4
## 28  13 0.07932  549.8
## 29  13 0.08605  500.9
## 30  13 0.09326  456.4
## 31  13 0.10100  415.9
## 32  13 0.10920  378.9
## 33  13 0.11790  345.3
## 34  13 0.12720  314.6
## 35  13 0.13710  286.6
## 36  13 0.14750  261.2
## 37  13 0.15840  238.0
## 38  13 0.16990  216.8
## 39  13 0.18200  197.6
## 40  13 0.19450  180.0
## 41  13 0.20760  164.0
## 42  13 0.22110  149.5
## 43  13 0.23510  136.2
## 44  13 0.24940  124.1
## 45  13 0.26420  113.1
## 46  13 0.27920  103.0
## 47  13 0.29450   93.9
```

```
## 48   13 0.31000   85.5
## 49   13 0.32570   77.9
## 50   13 0.34150   71.0
## 51   13 0.35730   64.7
## 52   13 0.37320   59.0
## 53   13 0.38900   53.7
## 54   13 0.40470   48.9
## 55   13 0.42030   44.6
## 56   13 0.43570   40.6
## 57   13 0.45090   37.0
## 58   13 0.46590   33.7
## 59   13 0.48060   30.7
## 60   13 0.49490   28.0
## 61   13 0.50900   25.5
## 62   13 0.52270   23.2
## 63   13 0.53600   21.2
## 64   13 0.54890   19.3
## 65   13 0.56140   17.6
## 66   13 0.57340   16.0
## 67   13 0.58500   14.6
## 68   13 0.59610   13.3
## 69   13 0.60660   12.1
## 70   13 0.61670   11.1
## 71   13 0.62620   10.1
## 72   13 0.63530    9.2
## 73   13 0.64380    8.4
## 74   13 0.65170    7.6
## 75   13 0.65920    6.9
## 76   13 0.66610    6.3
## 77   13 0.67260    5.8
## 78   13 0.67860    5.2
## 79   13 0.68410    4.8
## 80   13 0.68920    4.4
## 81   13 0.69390    4.0
## 82   13 0.69820    3.6
## 83   13 0.70210    3.3
## 84   13 0.70570    3.0
## 85   13 0.70900    2.7
## 86   13 0.71200    2.5
## 87   13 0.71480    2.3
## 88   13 0.71730    2.1
## 89   13 0.71960    1.9
## 90   13 0.72170    1.7
## 91   13 0.72360    1.6
## 92   13 0.72530    1.4
## 93   13 0.72690    1.3
## 94   13 0.72830    1.2
## 95   13 0.72960    1.1
## 96   13 0.73080    1.0
## 97   13 0.73190    0.9
## 98   13 0.73280    0.8
## 99   13 0.73370    0.7
## 100  13 0.73450    0.7
```

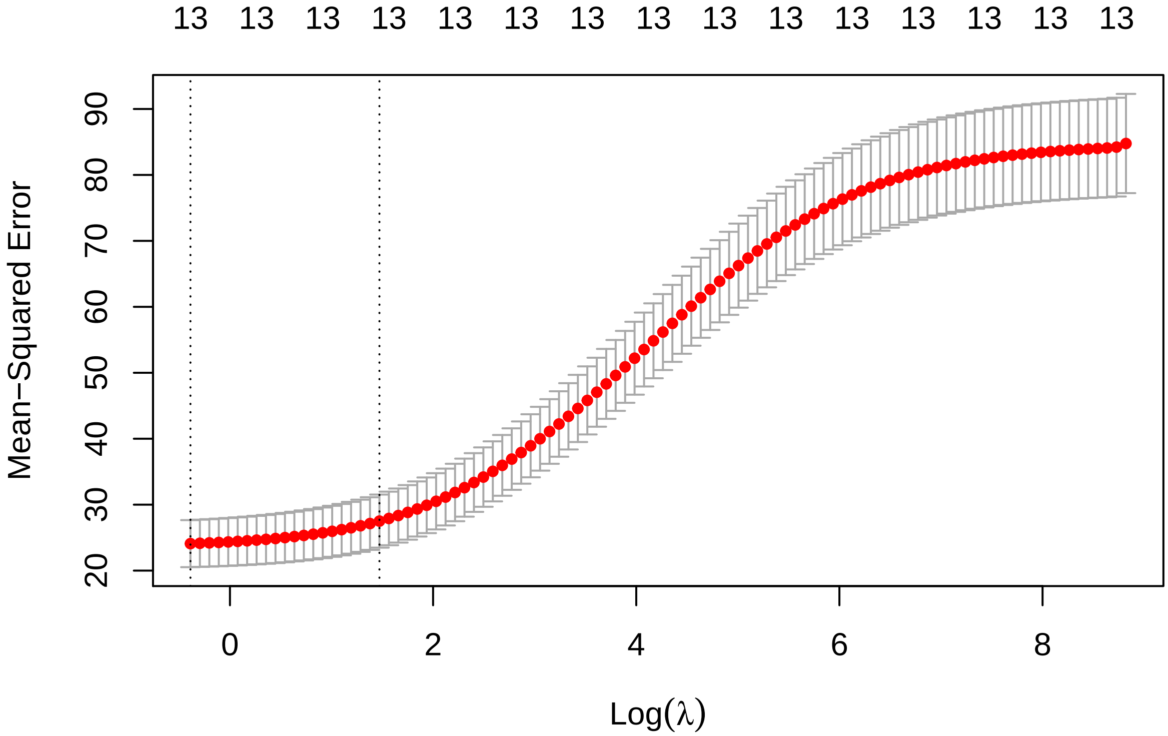
```r
# alpha = 0: Ridge Regression

# alpha = 1: Lasso Regression

plot(ridge)
```




```r
(cv.ridge <- cv.glmnet(y = boston.c$MEDV, x = as.matrix(boston.c[, explanatory]), alpha = 0))
```

```
##
## Call:  cv.glmnet(x = as.matrix(boston.c[, explanatory]), y = boston.c$MEDV,      alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min  0.678   24.09 3.567      13
## 1se  4.357   27.50 4.025      13
```

```r
plot(cv.ridge)
```


13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
Mean−Squared Error
20
30
40
50
60
70
80
90
0
2
4
6
8
Log(λ)