Assignment 1: Ridge Regression

Victor Villegas, Roger Llorenç, Luis Sierra

2024-02-21

The datasets used in this assignment are the medical *prostate_data.txt* file, and the classic dataset on crime from Boston, Massachussets. https://lib.stat.cmu.edu/datasets/boston

First, we import the data and scale it appropriately, by centering and normalizing the variables.

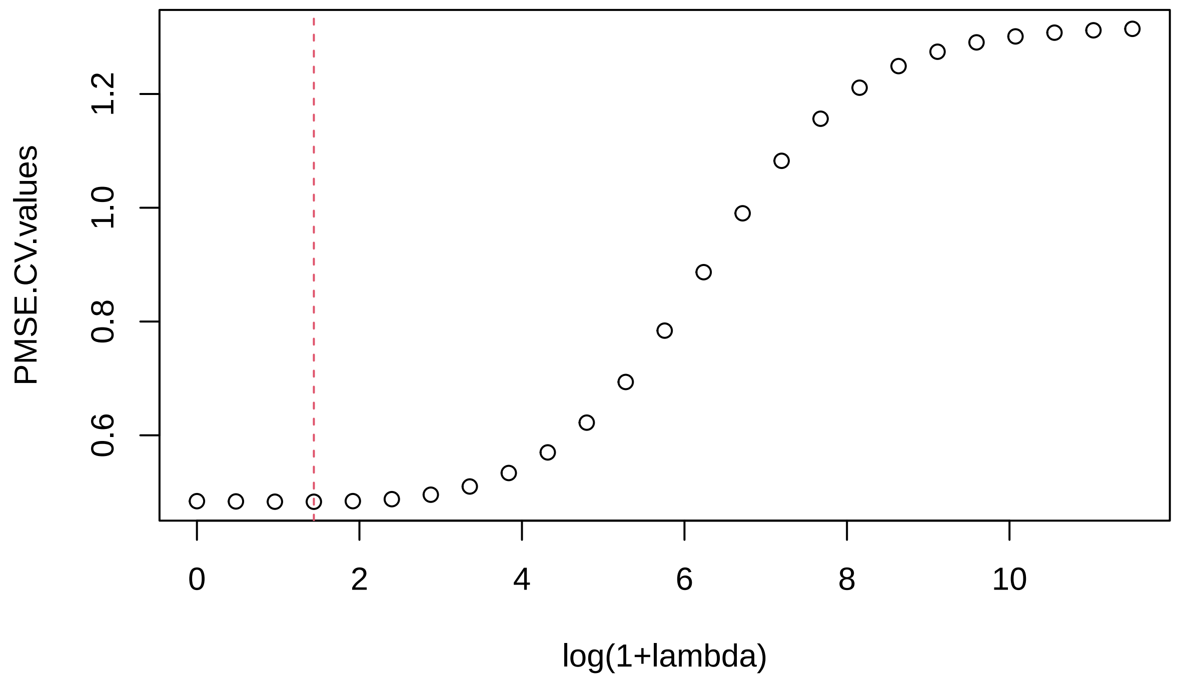
```r
prostate <- read.table("prostate_data.txt", header=TRUE, row.names = 1)
# plot(prostate)
train.sample <- which(prostate$train==TRUE)

library(MASS)
data(Boston)
```

```r
Y <- scale( prostate$lpsa, center=TRUE, scale=FALSE)
X <- scale( as.matrix(prostate[,1:8]), center=TRUE, scale=TRUE)
n <- dim(X)[1]
p <- dim(X)[2]
```

1. Chosing the penalization parameter λ:

The functions in the following section provide a way of finding the best value for the hyperparameter λ by estimating the PMSE in a validation set.

```r
# Calculates the effective degrees of freedom in a ridge regression model fit
effective_df <- function(lambda, X){
  d2 <- eigen(t(X)%*%X,symmetric = TRUE, only.values = TRUE)$values
  return(sum(d2/(d2+lambda)))
}


# Fits a ridge regression with inputs explanatory variables X, response y, penalty lambda
ridge_regression <- function(X, Y, lambda) {
  beta.par <- solve(t(X)%*%X + lambda*diag(1,dim(X)[2]))%*%t(X)%*%Y
  H.lambda <- X%*%t(solve(t(X)%*%X + lambda*diag(1,dim(X)[2]))) %*% t(X)
  df <- effective_df(lambda, X)
  return(list(beta.par = beta.par, H.lambda = H.lambda, df = df))
}


# Returns the vector of parameters minimising the PMSE(lambda) by LOOCV
PMSE_loocv <- function(lambda, X, Y) {
  output <- ridge_regression(X, Y, lambda)
  H.lambda <- output$H.lambda
  residuals <- (H.lambda%*%Y - Y)^2
  # Weights as influence of each data point on its own predicted value
  w <- diag(X%*%solve(t(X)%*%X + lambda*diag(1,dim(X)[2]))%*%t(X))
  squared_errors <- residuals/(1 - w)
  best_index <- which.min(squared_errors)
  PMSE.CV <- mean(squared_errors)
  return(PMSE.CV)
}


# Returns the vector of parameters minimising the PMSE(lambda) by Generalised CV
PMSE_gcv <- function(lambda, X, Y) {
  output <- ridge_regression(X, Y, lambda)
  H.lambda <- output$H.lambda
  residuals <- (H.lambda%*%Y - Y)^2
  # Weights as influence of each data point on its own predicted value
  nu <- sum(diag(X%*%solve(t(X)%*%X + lambda*diag(1,dim(X)[2]))%*%t(X)))
  squared_errors <- residuals/(1 - nu/dim(X)[1])
  best_index <- which.min(squared_errors)
  PMSE.CV <- mean(squared_errors)
  return(PMSE.CV)
}

# Shrinkage hyperparameter range
lambda.max <- 1e5
n.lambdas <- 25
lambda.v <- exp(seq(0,log(lambda.max+1),length=n.lambdas))-1

PMSE.CV.values <- lapply(lambda.v, PMSE_loocv, X, Y)
index <- which.min(PMSE.CV.values)
best_lambda <- lambda.v[index]
plot(log(1+lambda.v), PMSE.CV.values, xlab = "log(1+lambda)")
abline(v=log(1+best_lambda),col=2,lty=2)
```



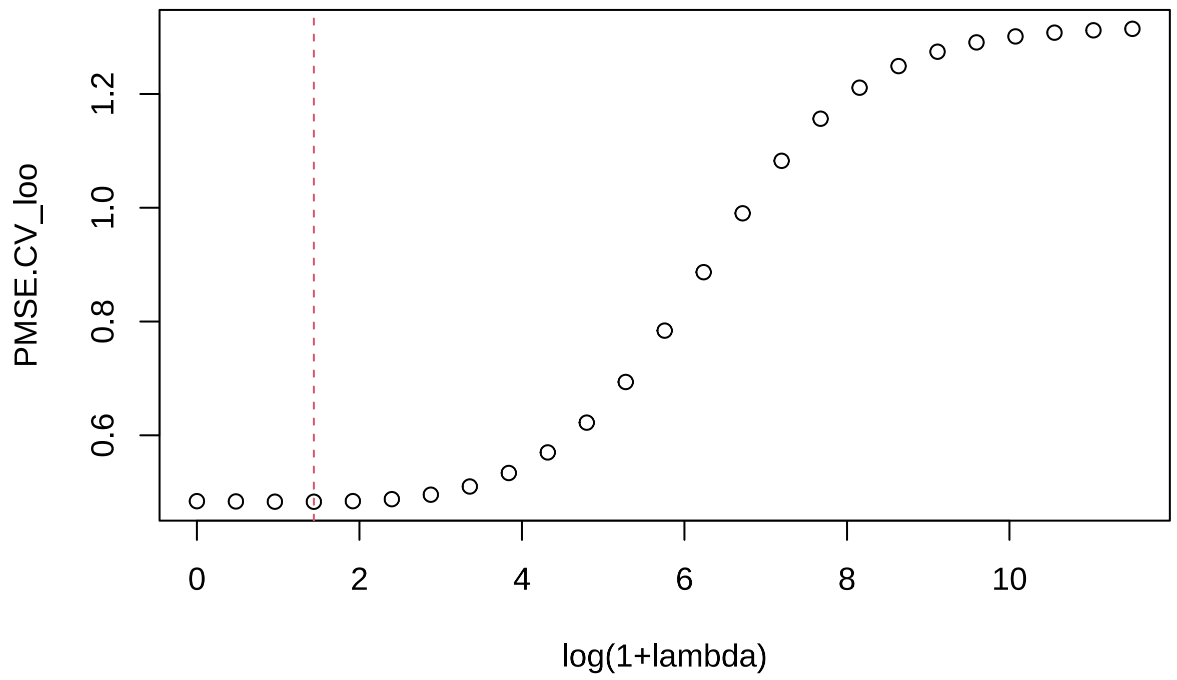
```
# plot(df.v, PMSE.CV.values, xlab="df(lambda)")
```

Instead of using *Leave-one-out* cross-validation, we can also use more elements in the calculation of our PMSE(λ) estimator. As such, the following chunk implements k-fold cross validation with different folds.

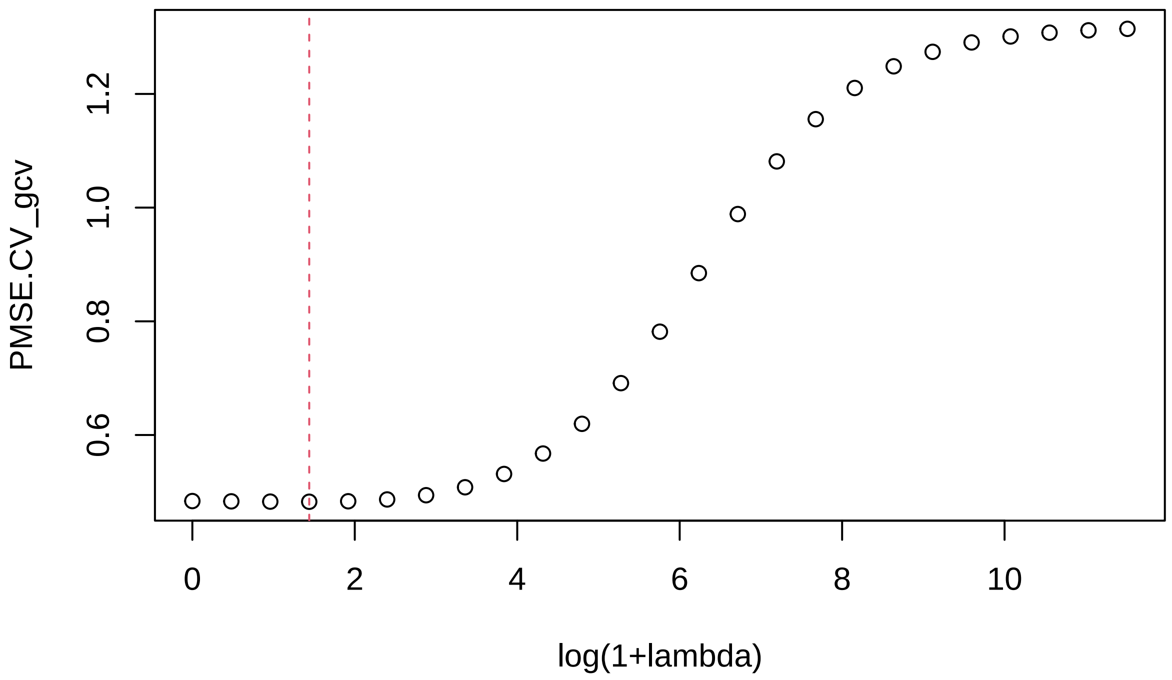
```
# Function to randomly split n datapoints into k folds, returning the index ordering in a list
split_indices <- function(n, k) {
  set.seed(1234) # To ensure replicability
  indices <- sample(n) # Generate random sequence from 1 to n
  fold_size <- ceiling(n / k)
  fold_indices <- split(indices, (seq_along(indices) - 1) %/% fold_size + 1)
  return(fold_indices)
}

#
PMSE_kfold <- function(X, Y, lambda.v, fold_indices) {
  lambda.max <- 1e5
  n <- dim(X)[1]
  p <- dim(X)[2]

  PMSE.CV_kfold <- numeric(length(lambda.v))

  for (l in 1:n.lambdas) {
    lambda <- lambda.v[l]
    PMSE.CV_kfold[l] <- 0

    for (fold in 1:length(fold_indices)) {
      train_indices <- unlist(fold_indices[-fold])
      val_indices <- fold_indices[[fold]]

      X_train <- X[train_indices, ]
      Y_train <- Y[train_indices]
      X_val <- X[val_indices, ]
      Y_val <- Y[val_indices]

      beta <- solve(t(X_train) %*% X_train + lambda * diag(1, p)) %*% t(X_train) %*% Y_train
      Y_pred <- X_val %*% beta

      PMSE.CV_kfold[l] <- PMSE.CV_kfold[l] + sum((Y_pred - Y_val)^2)
    }
    PMSE.CV_kfold[l] <- PMSE.CV_kfold[l]/n
  }
  return(PMSE.CV_kfold)
}
```

Now we will apply use the previously defined functions to perform a ridge regression on our datasets, starting on the prostate cancer data.

```
Y <- scale(prostate$lpsa, center=TRUE, scale=FALSE) # Center but no rescaling on the response
X <- scale(as.matrix(prostate[,1:8]), center = TRUE, scale = TRUE) # Center and rescaling, predictor va

n <- nrow(X)
lambda.v <- exp(seq(0,log(lambda.max+1),length=n.lambdas))-1

PMSE.CV_loocv <- lapply(lambda.v, PMSE_loocv, X, Y)
lambda.CV_loocv <- lambda.v[which.min(PMSE.CV_loocv)] # Returns the index which we select
plot(log(1+lambda.v), PMSE.CV_loocv, xlab = "log(1+lambda)", ylab="PMSE.CV_loo")
abline(v=log(1+lambda.CV_loocv),col=2,lty=2)
```



```
# Folds of size 5
fold_indices <- split_indices(n, 5)
PMSE.CV_kfold <- PMSE_kfold(X, Y, lambda.v, fold_indices)

# Folds of size 10
fold_indices <- split_indices(n, 10)
PMSE.CV_kfold <- PMSE_kfold(X, Y, lambda.v, fold_indices)

# Generalised Cross Validation
PMSE.CV_gcv <- lapply(lambda.v, PMSE_gcv, X, Y)
lambda.CV_gcv <- lambda.v[which.min(PMSE.CV_gcv)]

plot(log(1+lambda.v), PMSE.CV_gcv, xlab = "log(1+lambda)", ylab="PMSE.CV_gcv")
abline(v=log(1+lambda.CV_gcv),col=2,lty=2)
```



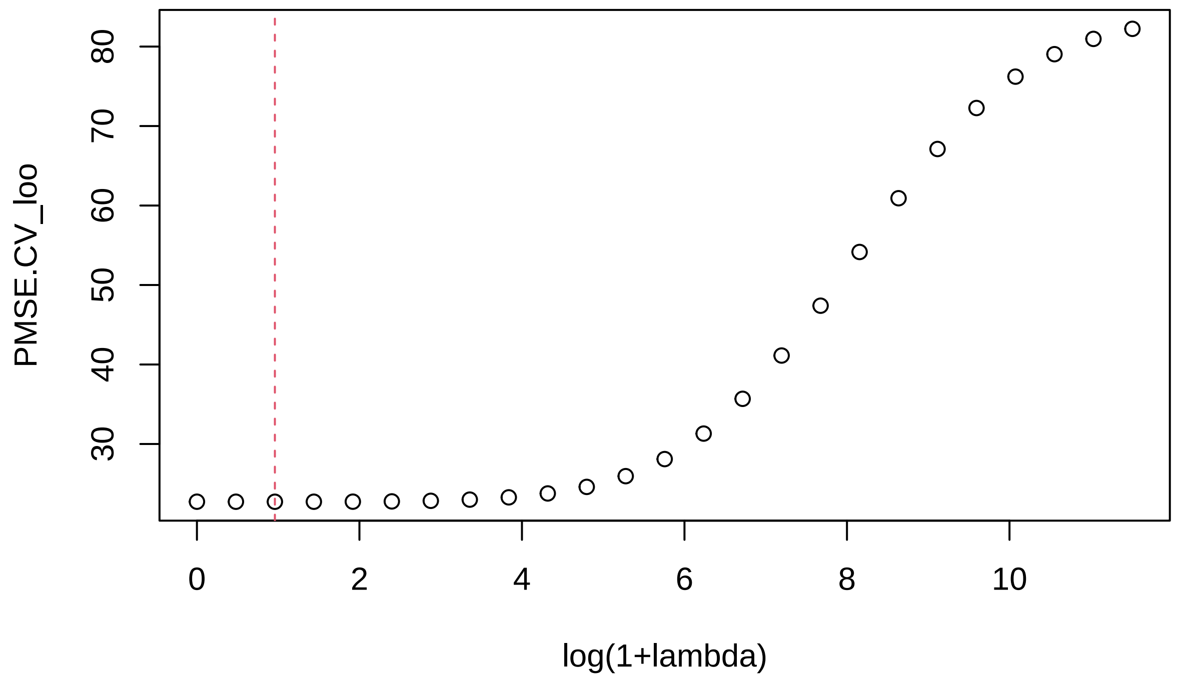
2. Ridge regression for the Boston Housing data

Now moving on to the Boston dataset, we shall apply the same transformations to then find the best possible λ by cross validation.

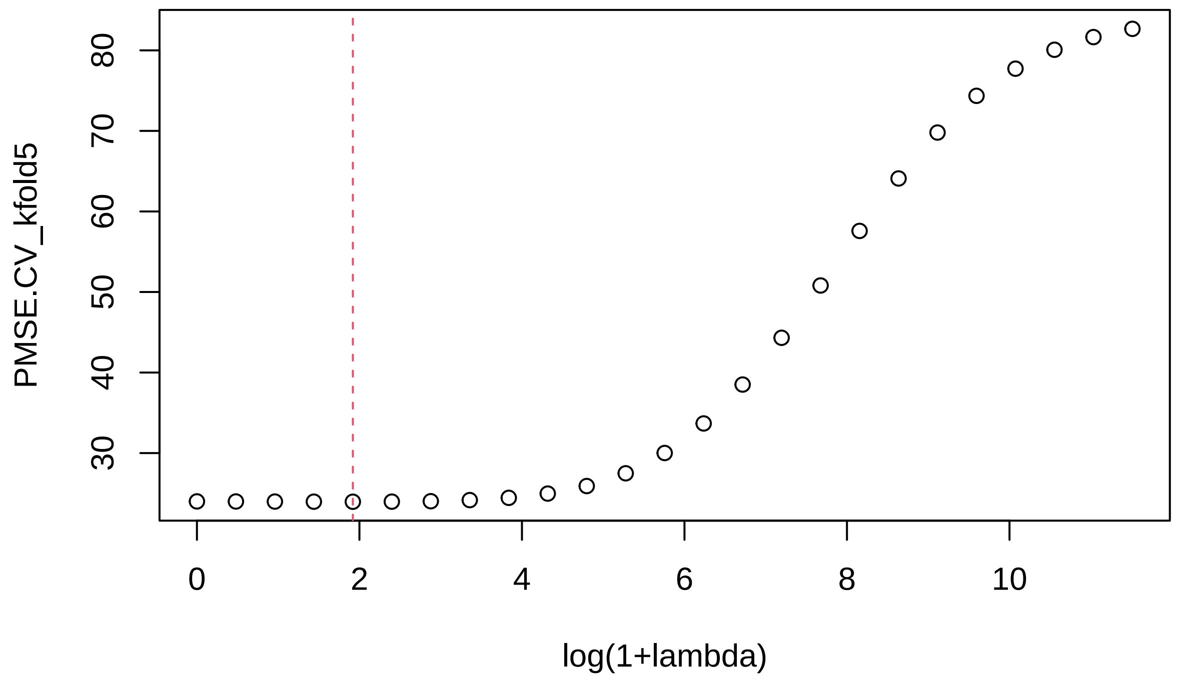
```
Y <- scale(Boston$medv, center=TRUE, scale=FALSE) # Center but no rescaling on the response
X <- scale(as.matrix(Boston[,1:13]), center = TRUE, scale = TRUE) # Center and rescaling,

n <- dim(X)[1]
p <- dim(X)[2]
lambda.max <- 1e5
n.lambdas <- 25
lambda.v <- exp(seq(0,log(lambda.max+1),length=n.lambdas))-1
```

We use the same range of values for λ.

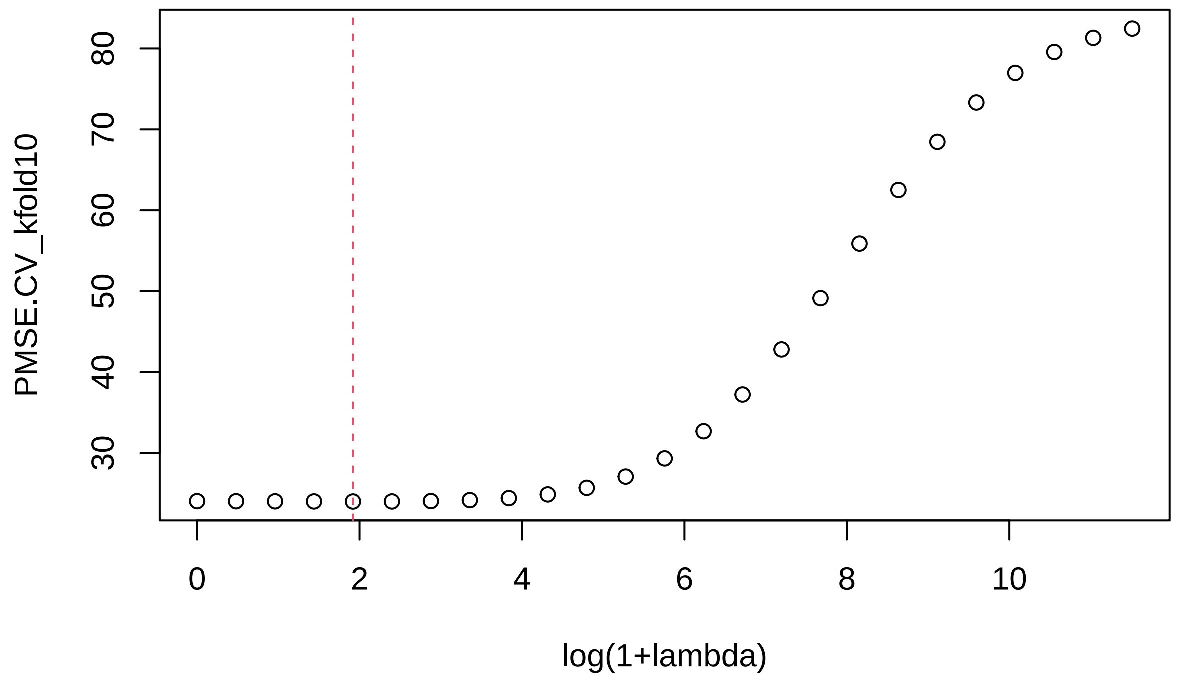
```
PMSE.CV_loocv <- lapply(lambda.v, PMSE_loocv, X, Y)
lambda.CV_loocv <- lambda.v[which.min(PMSE.CV_loocv)] # Returns the index which we select
plot(log(1+lambda.v), PMSE.CV_loocv, xlab = "log(1+lambda)", ylab="PMSE.CV_loo")
abline(v=log(1+lambda.CV_loocv),col=2,lty=2)
```



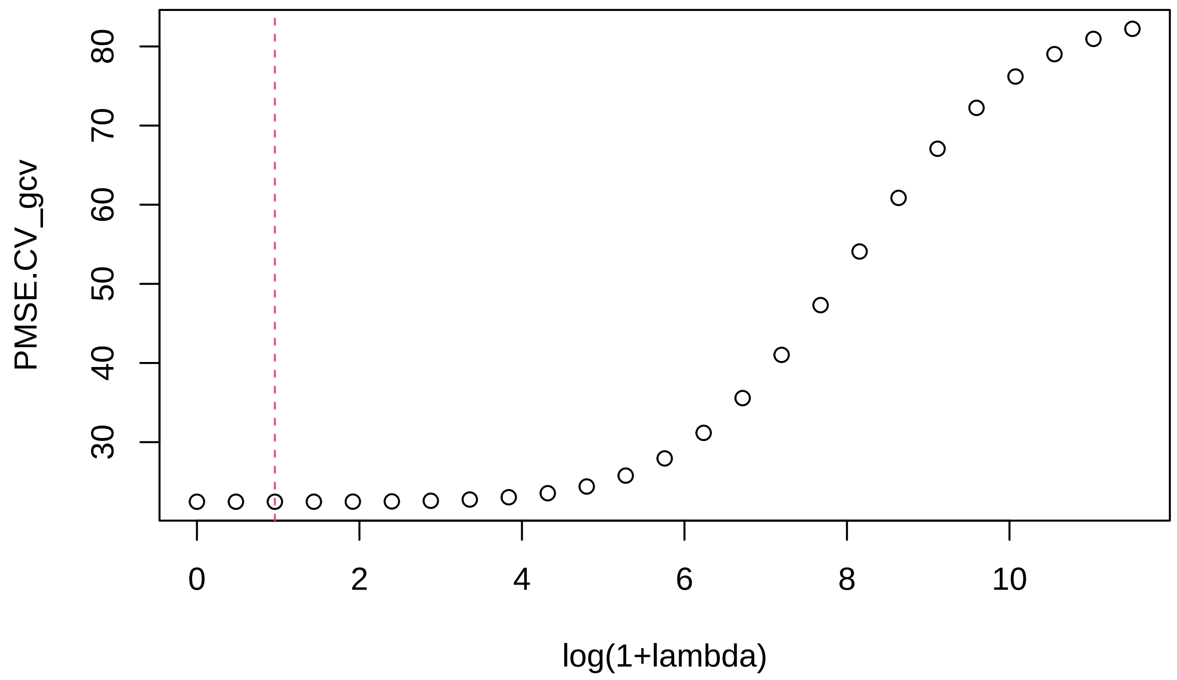
```r
# Folds of size 5
fold_indices5 <- split_indices(n, 5)
PMSE.CV_kfold5 <- PMSE_kfold(X, Y, lambda.v, fold_indices5)
lambda.CV_kfold5 <- lambda.v[which.min(PMSE.CV_kfold5)]

plot(log(1+lambda.v), PMSE.CV_kfold5, xlab = "log(1+lambda)", ylab="PMSE.CV_kfold5")
abline(v=log(1+lambda.CV_kfold5),col=2,lty=2)
```



```r
# Folds of size 10
fold_indices10 <- split_indices(n, 10)
PMSE.CV_kfold10 <- PMSE_kfold(X, Y, lambda.v, fold_indices10)
lambda.CV_kfold10 <- lambda.v[which.min(PMSE.CV_kfold10)]

plot(log(1+lambda.v), PMSE.CV_kfold10, xlab = "log(1+lambda)", ylab="PMSE.CV_kfold10")
abline(v=log(1+lambda.CV_kfold10),col=2,lty=2)
```

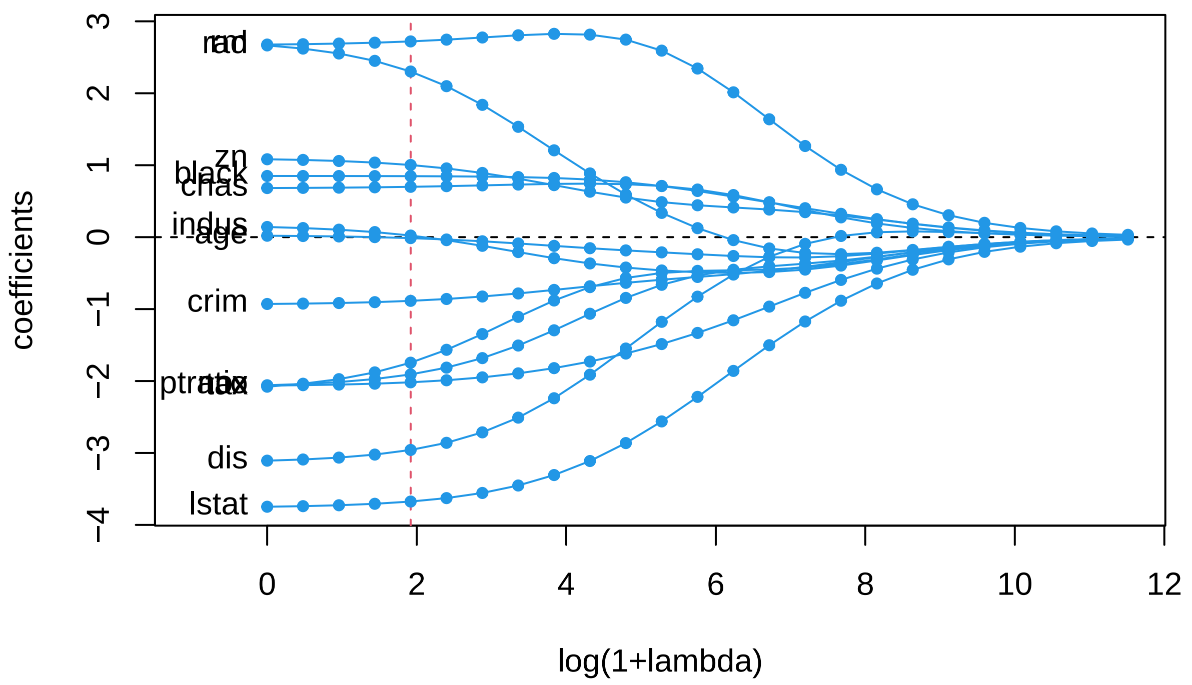


```r
# Generalised Cross Validation
PMSE.CV_gcv <- lapply(lambda.v, PMSE_gcv, X, Y)
lambda.CV_gcv <- lambda.v[which.min(PMSE.CV_gcv)]

plot(log(1+lambda.v), PMSE.CV_gcv, xlab = "log(1+lambda)", ylab="PMSE.CV_gcv")
abline(v=log(1+lambda.CV_gcv),col=2,lty=2)
```

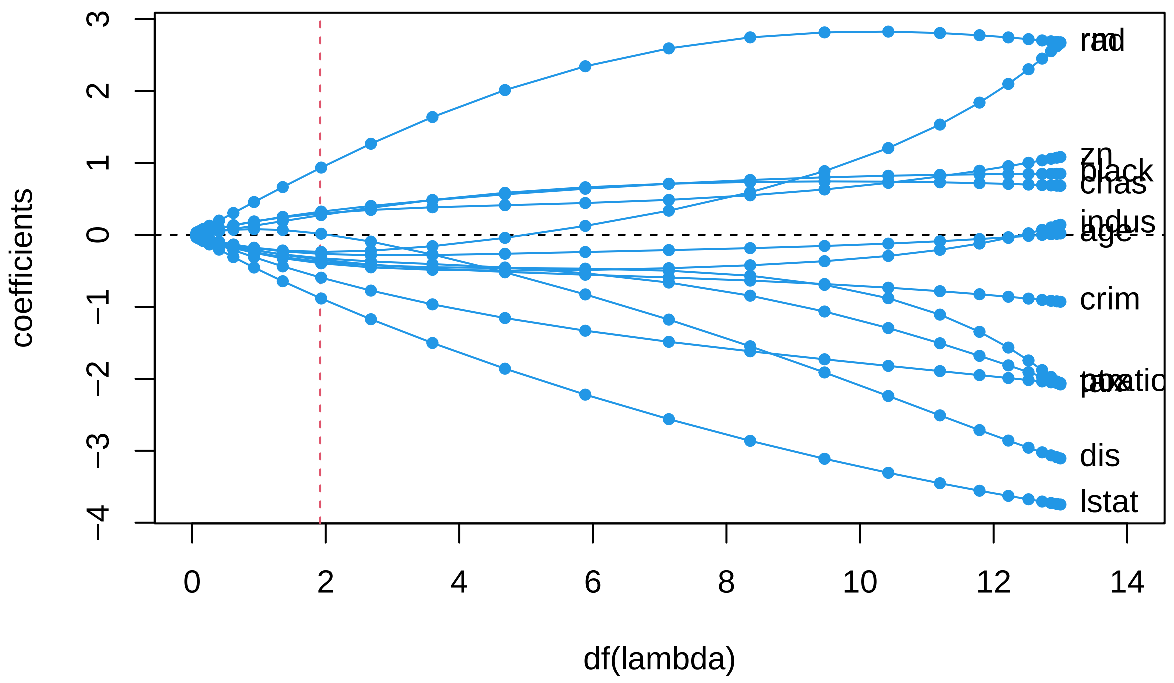


Finally, we shall take a look at the path the coefficients take to analyse the behaviour of the model as the value of λ evolves.

```r
# Coefficients path
beta.path <- matrix(0,nrow=n.lambdas, ncol=p)
diag.H.lambda <- matrix(0,nrow=n.lambdas, ncol=n)
for (l in 1:n.lambdas){
  lambda <- lambda.v[l]
  H.lambda.aux <- t(solve(t(X)%*%X + lambda*diag(1,p))) %*% t(X)
  beta.path[l,] <-  H.lambda.aux %*% Y
  H.lambda <- X %*% H.lambda.aux
  diag.H.lambda[l,] <- diag(H.lambda)
}
plot(c(-1,log(1+lambda.v[n.lambdas])), range(beta.path),type="n",
     xlab="log(1+lambda)",ylab="coefficients")
abline(h=0,lty=2)
abline(v=log(1+lambda.CV_kfold10),col=2,lty=2)
for(j in 1:p){
  lines(log(1+lambda.v),beta.path[,j],col=4)
  points(log(1+lambda.v),beta.path[,j],pch=19,cex=.7,col=4)
}
text(0*(1:p), beta.path[1,],names(Boston)[1:p],pos=2)
```



```
df.v <- lapply(lambda.v, effective_df, X)

# estimated coefficients path against effective degrees of freedom
plot(c(0,p+1), range(beta.path),type="n",xlab="df(lambda)",ylab="coefficients")
abline(h=0,lty=2)
abline(v=log(1+lambda.CV_kfold10),col=2,lty=2)
PMSE.CV_kfold <- PMSE_kfold(X, Y, lambda.v, fold_indices)
for(j in 1:p){
  lines(df.v,beta.path[,j],col=4)
  points(df.v,beta.path[,j],pch=19,cex=.7,col=4)
}
text(p+0*(1:p), beta.path[1,],names(Boston)[1:p],pos=4)
```



We have added the 10-fold CV λ to the previous coefficient path evolution to see where the supposedly best λ leaves the values of the coefficients, and indeed it shrinks them all by a similar amount, and the *indus* and *age* variables, which were small for the unrestricted model, are left close to zero when the corss validation attains the minimum PMSE.