Lasso estimation in LM and GLM

Joel Cantero Priego and Ricard Meyerhofer Parra

19/11/2019

Introduction

In this assignment, we are going to use the Boston House-price dataset. It concerns housing values in 506 suburbs of Boston corresponding to year 1978. The following is a list where we can see all the variables of the dataset:

Variable name	Description	Values
CRIM	per capita crime rate by town	Integer
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.	Integer
INDUS	proportion of non-retail business acres per town	Integer
CHAS	Charles River dummy variable	Factor with 2 levels
NOX	nitric oxides concentration	Integer
RM	average number of rooms per dwelling	Integer
AGE	proportion of owner-occupied units built prior to 1940	Integer
DIS	weighted distances to five Boston employment centres	Integer
RAD	index of accessibility to radial highways	Integer
TAX	full-value property-tax rate per $10,000	Integer
PTRADIO	pupil-teacher ratio by town	Integer
B	1000(Bk - 0.63)ˆ2 where Bk is the proportion of blacks by town	Integer
LSTAT	% lower status of the population	Integer
MEDV	Median value of owner-occupied homes in $1000's	Integer

The Boston House-price corrected dataset (available in boston.Rdata) contains the same data (with some corrections) and it also includes the UTM coordinates of the geographical centers of each neighborhood.

Our first step is to load this dataset and check for missing values thanks to mice package.

As we can see, there is not any missing value for any observation so we can continue and it is not necessary to deal with imputation.

Lasso estimation

As mentioned in the statement, we are going to model this dataset by using the glmnet Lasso estimation. This regression model, will have $CMEDV$ as response variable and the remaining 13 aforementioned variables as explanatory variables.

In order to do so, first we select these 13 variables as explanatory variables and we convert CHAS factor variable to numeric one, because we have to scale it before applying Lasso Estimation. Then, we assign CMEDV variable (corrected version of MEDV) as our response variable Y.

```
# Selecting 13 explanatory variables from Boston dataset.
X <- dataset[,8:20]
```

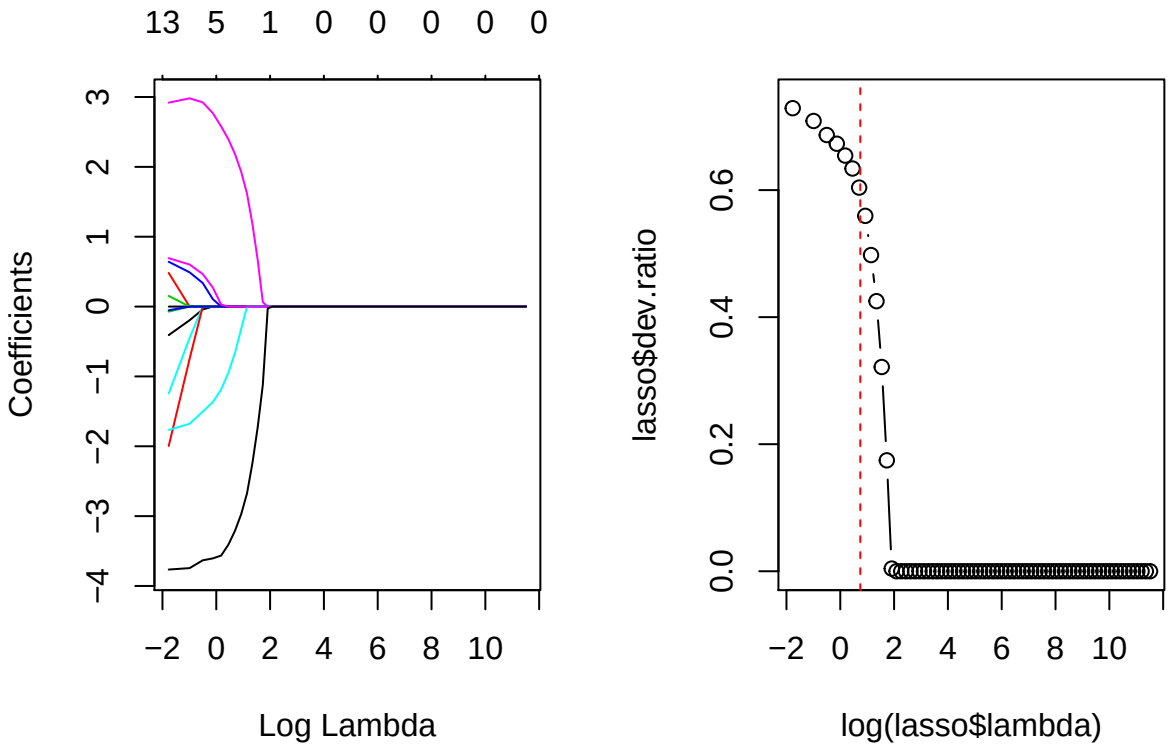
```r
# Converting CHAS factor variable as numeric.
X$CHAS <- as.numeric(X$CHAS)
# Scaling X and centering Y
X <- scale( as.matrix(X), center=T, scale=T)
Y <- scale( dataset$CMEDV, center=T, scale=F)
```

```r
#Lambdas
lambda.max <- 1e5
n.lambdas <- 74
lambda.values <- exp(seq(0,log(lambda.max+1),length=n.lambdas))-1
d2 <- eigen(t(X)%*%X,symmetric = TRUE, only.values = TRUE)$values
df.v <- numeric(n.lambdas)

lambda.v <- seq(lambda.max,0,length=n.lambdas)
for (l in 1:n.lambdas){
  lambda <- lambda.v[l]
  df.v[l] <- sum(d2/(d2+lambda))
}

# Lasso estimation in glmnet
lasso <- glmnet(X, Y, lambda=lambda.values)

# Plotting
minLambda = min(lasso$lambda + lasso$dev)
par(mfrow=c(1,2))
plot(lasso, xvar="lambda")
plot(log(lasso$lambda), lasso$dev.ratio, type='b')
abline(v=minLambda,col=2,lty=2)
```



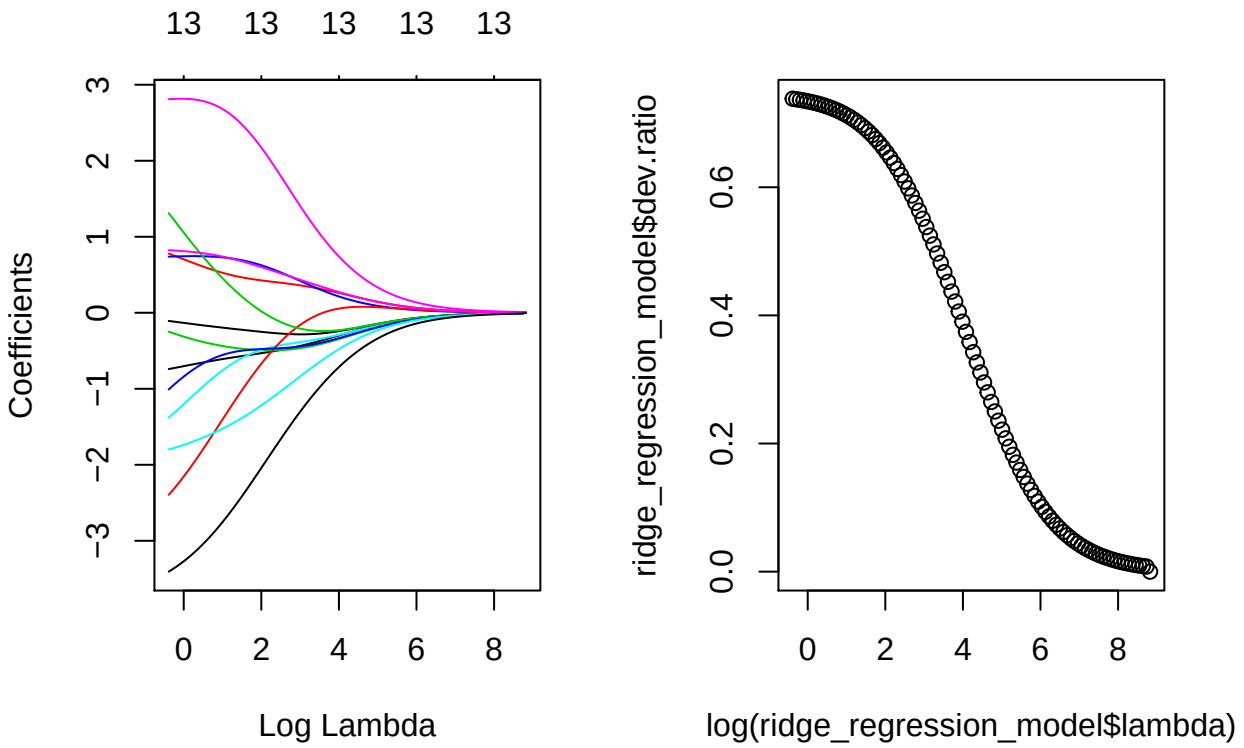
In the first plot, we can see each different coefficient for each color. We can say that we have a bigger penalization if we use more variables. If we increase lambda, less variables are used.

Use glmnet to fit the previous model using ridge regression. Compare the 10-fold cross validation results from function cv.glmnet with those you obtained in the previous practice with your own functions.

```
ridge_regression_model <- glmnet(X, Y, alpha=0)
par(mfrow=c(1,2))
summary(ridge_regression_model)
```

```
##           Length Class     Mode
## a0         100   -none-    numeric
## beta      1300   dgCMatrix S4
## df         100   -none-    numeric
## dim          2   -none-    numeric
## lambda     100   -none-    numeric
## dev.ratio  100   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

```
plot(ridge_regression_model,  xvar="lambda")
plot(log(ridge_regression_model$lambda), ridge_regression_model$dev.ratio, type='b')
```



As expected we can see how the error goes down the higher the value of the lamdas.

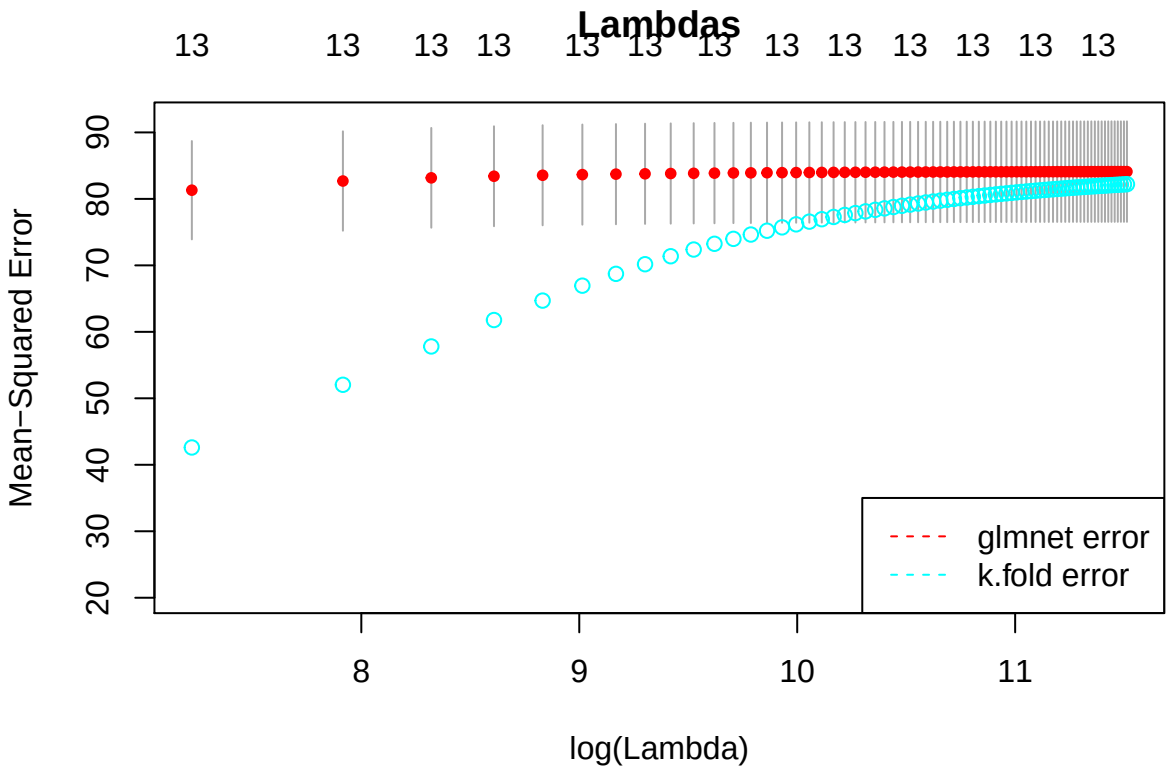
In order to compare the two models we will use the same lambdas that we have generated and feed them to both methods, thus ensuring that the results are not tricked by using differnet values of lambdas.

We will also set the **standardize** flag to false since our data is already standardized as seen in the first block of code. Opting to use the original that without centering and standarizing it could have also been an option, too.

We have tried sending to glmnet the data without centering it ourselves and letting the function standardize it itself, but the results have been the same.

```
m3.rr <- cv.glmnet(X, Y, nfolds=10, lambda = lambda.v, standardize=FALSE, intercept=FALSE, alpha=0)
plot(m3.rr, main="Lambdas")
abline(v=log(m3.rr$lambda.min),col=2,lty=2)
abline(v=log(m3.rr$lambda.1se),col=2,lty=2)

cvm <- rev(m3.rr$cvm)
p <- dim(X)[2]
m4.rr <-  MSPEkfold(X, Y, 10, n.lambdas, lambda.v, dim(X)[1])
points(log(lambda.v), m4.rr, col=5)
legend("bottomright", c("glmnet error", "k.fold error"), col=c(2,5), lty=c(2,2), lwd=c(1,1)) # FIX
```




```
min.lambda <- data.frame("K.fold" = lambda.v[which.min(m4.rr)], "glmnet" = m3.rr$lambda.min)
min.df <- data.frame("K.fold" = df.v[which.min(m4.rr)],"glmnet" =df.v[which(lambda.v == m3.rr$lambda.mi
result <- rbind(min.df, min.lambda)
row.names(result) <- c("df", "lambda")

kable(result)
```

	K.fold	glmnet
df	13	13
lambda	0	0

Interestingly enough, we see how the values differ from our method to the one implemented by **glmnet**. Taking a look at the table, the most noticeable thing is that the optimal lambda is increased by more than 4 units, yet the value of the degrees of freedom remains almost the same.

We are not sure if our model is wrongly implemented or it is simply that the glmnet method is doing things differently from what we did in our implementation.

Yet if we take a look at the plot which compares the Mean Square error to with the values of the lambdas we can see how both models seem to follow the same shape.

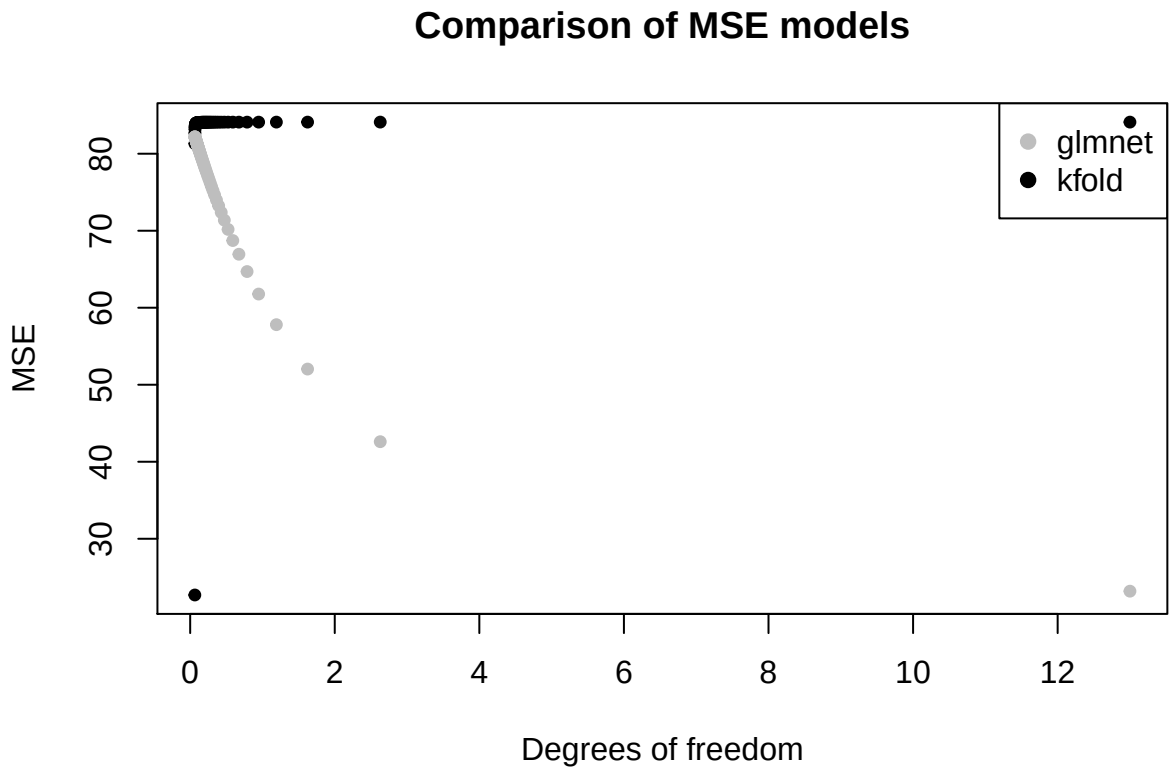
However, our model starts the curve later on (around 4 units), meaning that the Mean Squared Error does not start increasing until reaching higher values of λ.

```
lambda.glm <- m3.rr$lambda
df.v.g <- numeric(n.lambdas)
for (l in 1:n.lambdas){
  lambda <- lambda.glm[l]
  df.v.g[l] <- sum(d2/(d2+lambda))
```

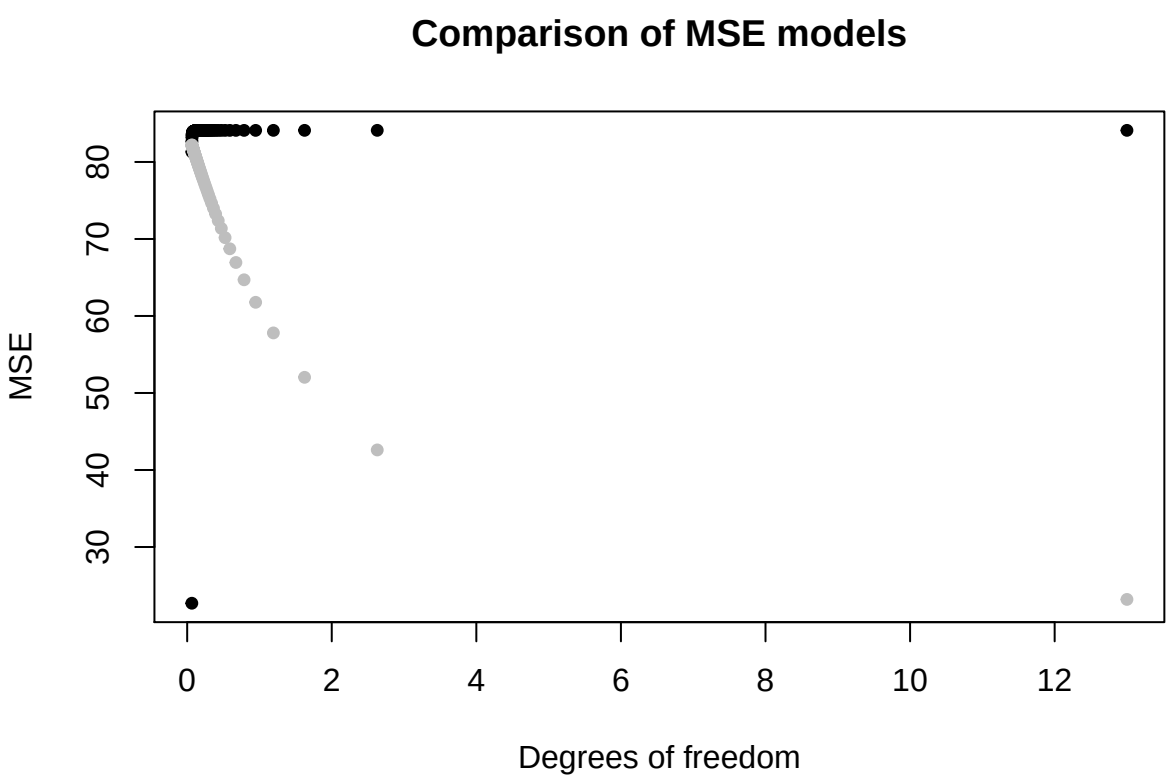
```
}

plot(df.v, cvm, col=1,pch=19,cex=.75, ylab="MSE", xlab="Degrees of freedom", main="Comparison of MSE mod
points(df.v, m4.rr,col=8,pch=19,cex=.75)

legend("topright",
       c("glmnet","kfold"),
       pch=c(19, 19),
       col=c(8, 1)
       )
```




```
## Pedro correction. Use df.v.g
lambda.glm <- m3.rr$lambda
df.v.g <- numeric(n.lambdas)
for (l in 1:n.lambdas){
    lambda <- lambda.glm[l]
    df.v.g[l] <- sum(d2/(d2+lambda))
}
plot(df.v.g, cvm, col=1,pch=19,cex=.75, ylab="MSE", xlab="Degrees of freedom", main="Comparison of MSE m
points(df.v.g, m4.rr,col=8,pch=19,cex=.75)
```



In this plot we are comparing the values of the degrees of freedom against the mean square error.

We can clearly see how both methods start and finish in the same values, which is expected.

Yet there is a big difference in the values between: our method lowers the error slower as more degrees of freedom are reached (so less variables are used).

Meanwhile the glmnet method follows more of a curve on the opposite sides towards lower error. It seems to stagnate when reaching higher degrees of freedom.

Assuming our method is correct, it seems that our implementation of k-fold is worse, since unless we reach 13 degrees of freedom, the MSE does not lower below 40.

The glmnet function proves to be getting less error around 4/6 degrees of freedom, so this would be our preferred method.

Conclusion

Our conclussion is that maybe our method has not been properly implemented, since we are having the same issues when passing the data untouched to glmnet and telling it to standardize it itself.

This tells us that the root of the discrepancies of the results emerge probably from a bad implementation from our side of the k-Fold function. However, we have rechecked it and we cannot pinpoint our fault in the implementation.

Thus, we will remain waiting for the correction of the first lab to correct this one.