Lasso estimation in LM and GLM

Joel Cantero Priego and Ricard Meyerhofer Parra

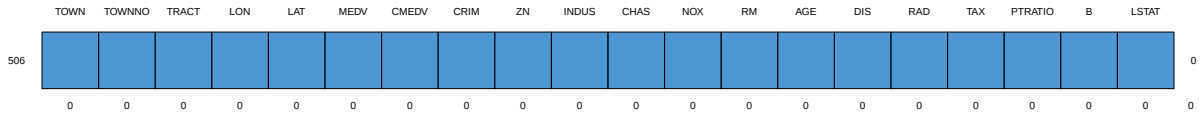
19/11/2019

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)^2 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.

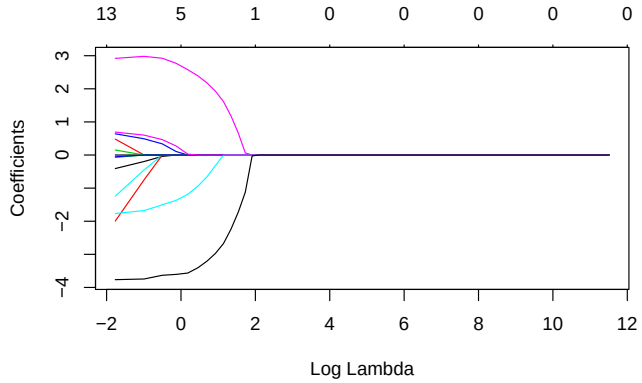


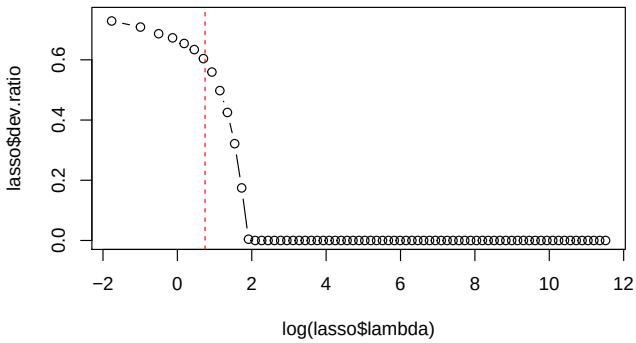
As we can see with the mice package, there is any missing value so we can continue and it is not necessary to deal with imputation.

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.

```
lambda.max <- 1e5
n.lambdas <- 74 #tuned
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)
for (l in 1:n.lambdas){
  lambda <- lambda.values[l]
  df.v[l] <- sum(d2/(d2+lambda))
}
# Lasso estimation in glmnet
lasso <- glmnet(X, Y, lambda=lambda.values)
```

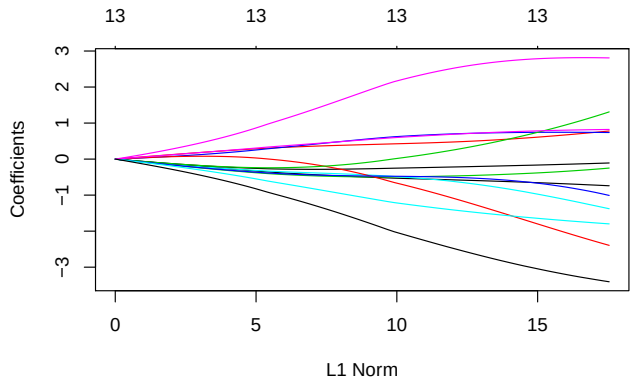


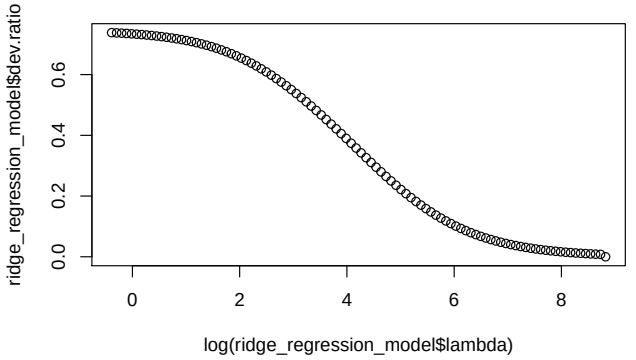


In the first plot, we can see each different coefficient for each color. We can say that we have a bigger penalization if we use more variables. If we increase lambda, less variables are used.

Compare the 10-fold cv results with the ones you implemented in your previously

```
ridge_regression_model <- glmnet(X, Y, alpha=0, standardize=FALSE)
```

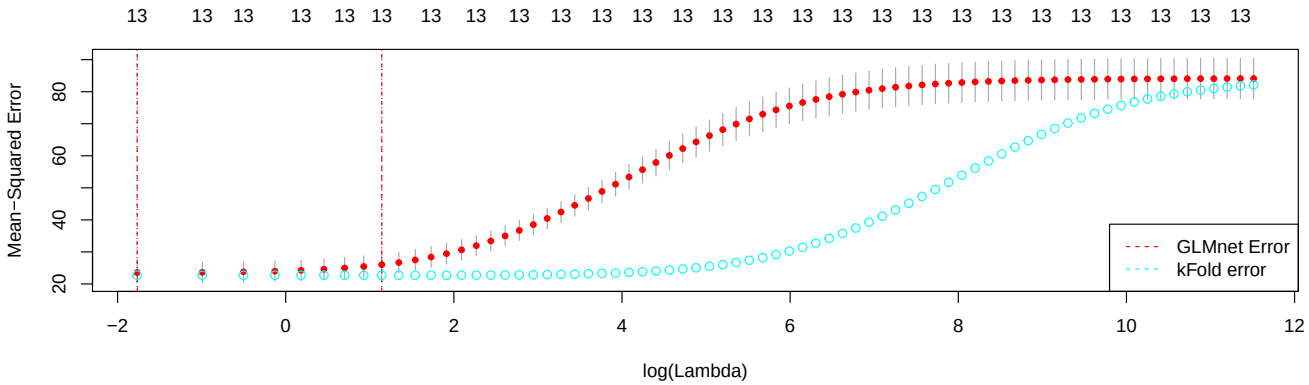




First of all, we are going to select the same lambdas that we used in the previous method just to compare both of them. As we can see in the plots, the error decreases when the values of the lambdas increases.

We need to set standardize paramater to false because our data is already centered and scaled.

```
glmnet.cv <- cv.glmnet(X, Y, nfolds=10, lambda = lambda.values,
                       standardize=FALSE, intercept=FALSE, alpha=0)
plot(glmnet.cv)
abline(v=log(glmnet.cv$lambda.min),col=2,lty=2)
abline(v=log(glmnet.cv$lambda.1se),col=2,lty=2)
cvm <- rev(glmnet.cv$cvm)
kFold <-  MSPEkfold(X, Y, 10, n.lambdas, lambda.values, dim(X)[1])
points(log(lambda.values), kFold, col=5)
legend("bottomright", c("GLMnet Error", "kFold error"), col=c(2,5), lty=c(2,2), lwd=c(1,1))
```

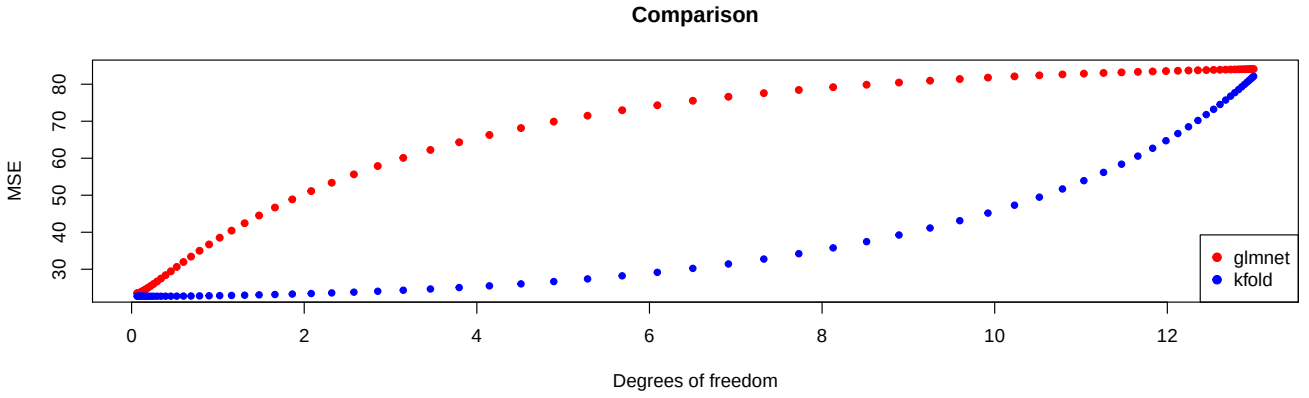



```
min.lambda <- data.frame("kFold" = lambda.values[which.min(kFold)], "GLMnet" = glmnet.cv$lambda.min)
min.df <- data.frame("kFold" = df.v[which.min(kFold)],
                     "GLMnet" = df.v[which(lambda.values == glmnet.cv$lambda.min)])
result <- rbind(min.df, min.lambda)
row.names(result) <- c("df", "lambda")
```

	kFold	GLMnet
df	12.78238	12.9847773
lambda	2.53139	0.1708283

The previous plot shows us that both methods follows the same curve but kFold method starts the curve later, it starts to increase MSE until lambda is around 6-8.

```
lambda.glm <- glmnet.cv$lambda
degrees_of_freedom <- numeric(n.lambdas)
for (l in 1:n.lambdas){
    lambda <- lambda.glm[l]
    degrees_of_freedom[l] <- sum(d2/(d2+lambda))
}
```



In the previous plot we can see in red points the values of the degrees of freedom against the MSE from GLMnet, and the blue points representing the kFold method. As we can see, with 0 and 12 degrees of freedom the MSE is equal for both methods, as we expect, but each one gets a different curves.

kFold method gets the error lower as less variables are used (more d.f.) The glmnet function proves to be getting more error around 4-8 degrees of freedom, so for this reason, we discard it. Otherwise, kFold method proves to be getting less error with the same degrees of freedom, so this would be our final method.