HDS Exercise set 4

Shabbeer Hassan

Problem 1 – Solution

```r
# Libraries
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 3.0-1
```

```r
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```r
library(plotmo)
```

```
## Loading required package: Formula
## Loading required package: plotrix
## Loading required package: TeachingDemos
```

```r
library(car)
```

```
## Loading required package: carData
```

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------ tidyverse 1.2.1 --
## v tibble  2.1.3     v purrr   0.3.2
## v tidyr   0.8.3     v dplyr   0.8.1
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0
## -- Conflicts --------------------------------------------------------- tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## x dplyr::recode() masks car::recode()
## x purrr::some()   masks car::some()
```

```r
library(ggplot2)
library(MASS)
```

```
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(data.table)
```

```
##
## Attaching package: 'data.table'
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(dplyr)

tr.ind <- read.csv("~/Dropbox/Important_Documents/Doctoral_Work/Courses/High Dimensional Stats/2019/Weel

# Train & Test dataset
data(Boston)
train <- Boston[tr.ind$X1,]
test <- Boston[-tr.ind$X1,]

# Predictor variables
x <- model.matrix(medv~., train)[,-1]
x.test <- model.matrix(medv~., test)[,-1]

# Outcome variable
y <- train$medv
y.test <- test$medv
```

(a)

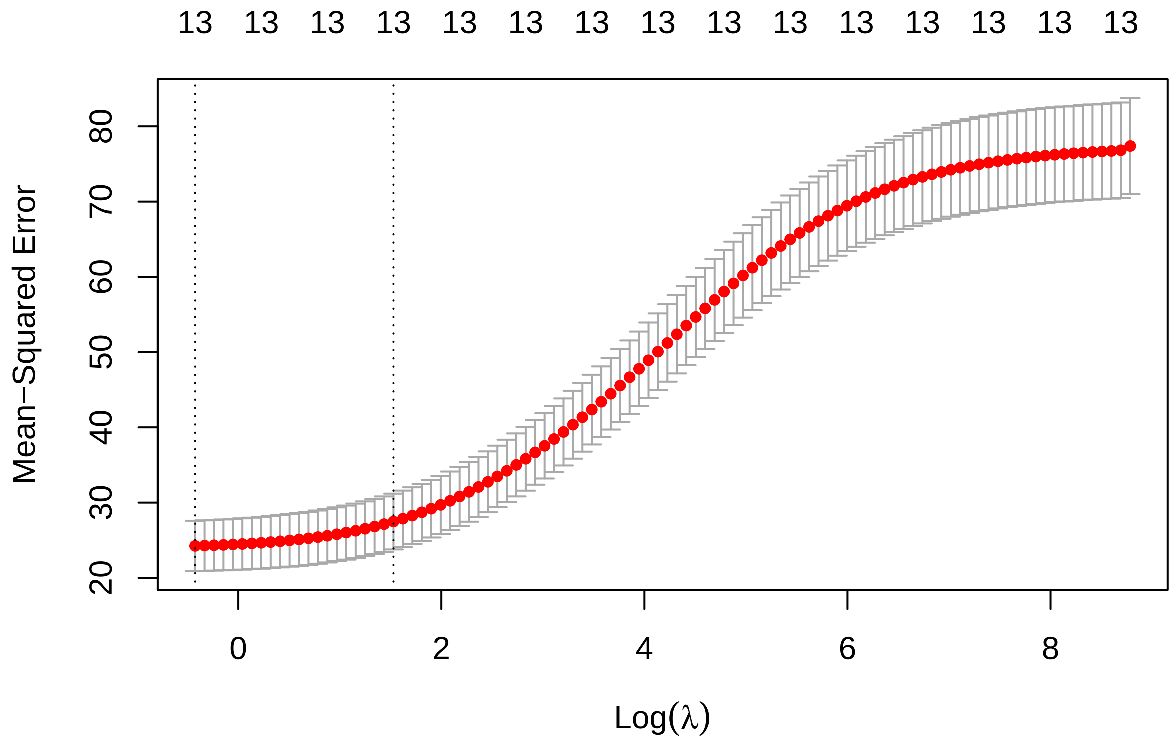
```
# Finding the best lambda using cross-validation
cv_glmnet <- cv.glmnet(x, y, alpha = 0, type.measure = "mse")

c(cv_glmnet$lambda.min, cv_glmnet$lambda.1se)
```

```
## [1] 0.6536359 4.6112717
```

```
round(log(c(cv_glmnet$lambda.min, cv_glmnet$lambda.1se)), 2)
```

```
## [1] -0.43  1.53
```

```
plot(cv_glmnet) # PLotting model
```



According to the cv-plot, we see that normal regression aka OLS would do fine here since @ it corresponds to lambda = 0 and from the ridge reg above we see that lambda.min = 0.6521009. As the lambda increases, model predictions do not become better since MSE also increases.

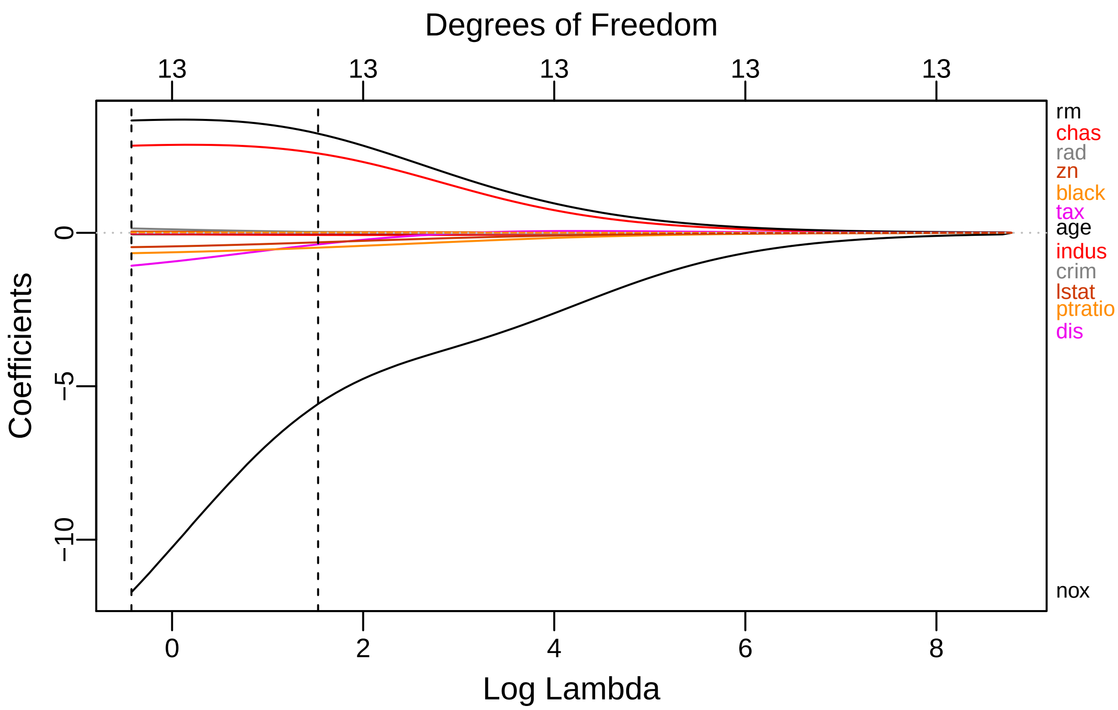
The lambda.min option refers to value of at the lowest CV error. The error at this value of is the average of the errors over the k folds and hence this estimate of the error is uncertain. The lambda.1se represents the value of in the search that was simpler than the best model (lambda.min), but which has error within 1 standard error of the best model.

In other words, using the value of lambda.1se as the selected value for results in a model that is slightly simpler than the best model but which cannot be distinguished from the best model in terms of error given the uncertainty in the k-fold CV estimate of the error of the best model.

Hence, if we use lambda.min, we might get the best model that may be too complex and slightly overfitted. BUt lambda.1se gives us the simplest model that has comparable error to the best model given the uncertainty.

(b)

```
plot_glmnet(cv_glmnet$glmnet.fit, label = T, xvar = "lambda")
abline(v = log(cv_glmnet$lambda.min), lty = 2)
abline(v = log(cv_glmnet$lambda.1se), lty = 2)
```




```r
# Run lm
fit_lm <- lm(medv~., train)
summary(fit_lm)
```

```
##
## Call:
## lm(formula = medv ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.2569  -2.7731  -0.5727   1.4788  25.8668
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  35.600400   6.200853   5.741 2.11e-08 ***
## crim         -0.072705   0.053498  -1.359 0.175051
## zn            0.057290   0.016161   3.545 0.000449 ***
## indus         0.035552   0.080274   0.443 0.658138
## chas          2.550825   0.955445   2.670 0.007960 **
## nox         -16.904750   4.524228  -3.736 0.000219 ***
## rm            3.381526   0.538233   6.283 1.03e-09 ***
## age          -0.003004   0.016036  -0.187 0.851492
## dis          -1.452771   0.242733  -5.985 5.56e-09 ***
## rad           0.300110   0.084084   3.569 0.000410 ***
## tax          -0.013904   0.004780  -2.909 0.003871 **
## ptratio      -0.753162   0.158022  -4.766 2.80e-06 ***
## black         0.008903   0.003348   2.659 0.008212 **
## lstat        -0.530554   0.064581  -8.215 4.67e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.767 on 335 degrees of freedom
```

```
## Multiple R-squared:  0.716,  Adjusted R-squared:  0.7049
## F-statistic: 64.95 on 13 and 335 DF,  p-value: < 2.2e-16
```

```r
vif(fit_lm)
```

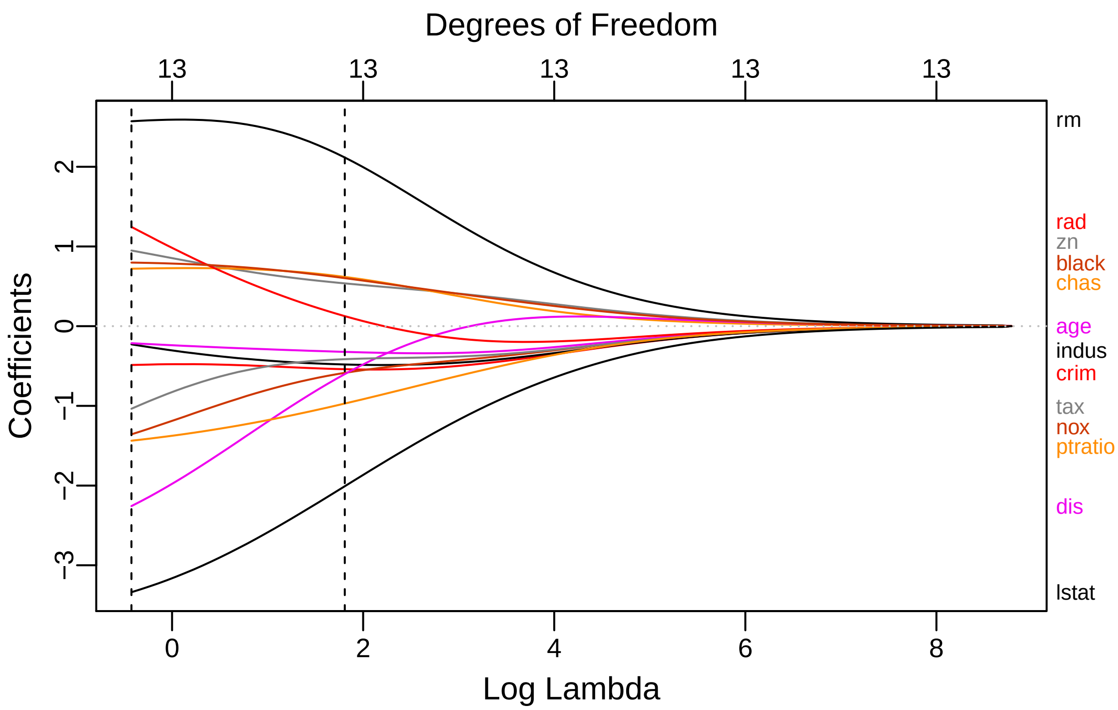
```
##     crim       zn    indus     chas      nox       rm      age      dis
## 2.281083 2.434418 4.580449 1.101435 4.451223 2.027185 3.234339 4.103866
##      rad      tax  ptratio    black    lstat
## 8.248746 9.766272 1.792897 1.305496 3.356000
```

In glmnet(), the ones with the greatest effects are as from the plot: "rm", "chas" and "nox".

We see that age and indus have been shrinked to zero and the ones remaining are included in the model. In lm(), we see that lstat, dis, rm and ptratio are amongst the most highly signififcant variables. However, in glmnet() we see that "dis" is not to be seen at all despite having 5.56e-09 p-value. Also nox and chas are not among the highly significant variables in the linear regression but are seen to have greater effects in glmnet()

(c)

```r
Boston$chas<-as.numeric(Boston$chas)

#Standardize covariates before fitting
Boston.X.std<- scale(dplyr::select(Boston,-medv))
X.train<- as.matrix(Boston.X.std)[tr.ind$X1,]
X.test<-  as.matrix(Boston.X.std)[-tr.ind$X1,]
Y.train<- Boston[tr.ind$X1, "medv"]
Y.test<- Boston[-tr.ind$X1, "medv"]

cv_glmnet_std = cv.glmnet( x = X.train, y = Y.train, alpha = 0, type.measure = "mse")
plot_glmnet(cv_glmnet_std$glmnet.fit, label = T, xvar = "lambda")
abline(v = log(cv_glmnet_std$lambda.min), lty = 2)
abline(v = log(cv_glmnet_std$lambda.1se), lty = 2)
```



Interestingly, upon standardization the top three variables with the largest effects in glmnet() become rm, dis and lstat, which also are among the ones with lowest p-values in the linear regression.

(d)

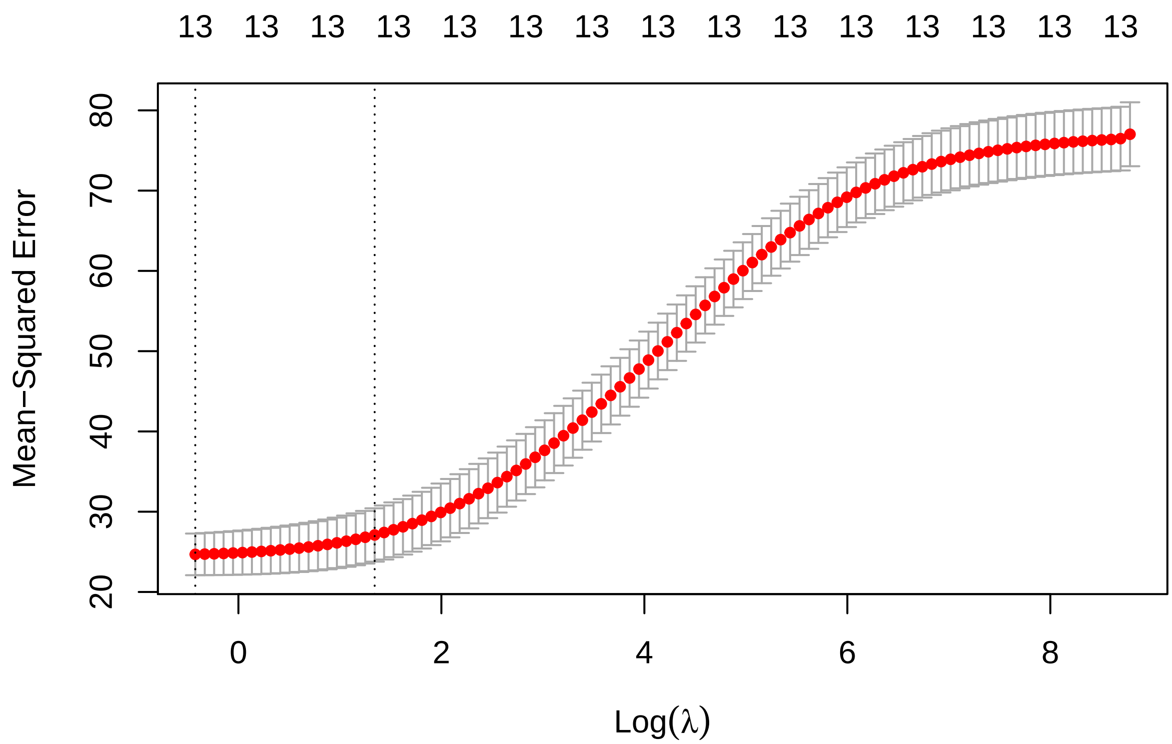
```
#### Ridge Reg

# Cross Validation to find lamda.min
cv1<- cv.glmnet(x=x, y=y, family = "gaussian", alpha = 0, nfolds = 10, type.measure = "mse")

# Predictions
pred1.min<- predict(cv1, newx = x.test, s = "lambda.min")

# MSPE (prediction error)
cv1_mse <- mean((y.test-pred1.min)^2)
cv1_mse
```

```
## [1] 24.6015
```

```
# Deviance explained
plot(cv1, xvar = "dev", label = TRUE)
```

```
## Warning in plot.window(...): "xvar" is not a graphical parameter

## Warning in plot.window(...): "label" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "xvar" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "label" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not
```

```
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not
## a graphical parameter

## Warning in box(...): "xvar" is not a graphical parameter

## Warning in box(...): "label" is not a graphical parameter

## Warning in title(...): "xvar" is not a graphical parameter

## Warning in title(...): "label" is not a graphical parameter
```




```
cv1_var_expl <- 100-(cv1_mse*100 )/var(y )
cv1_var_expl
```

```
## [1] 68.0595
```

Variance explained is around 68%

Problem 2 – Solution

(e)

```
###### LASSO

library(coefplot)

# Cross Validation to find lamda.min
```

```r
cv2<- cv.glmnet(x=x, y=y, family = "gaussian", alpha = 1, nfolds = 10,  type.measure = "mse")
cv2$lambda.min
```

```
## [1] 0.03918444
```

```r
cv2$lambda.1se
```

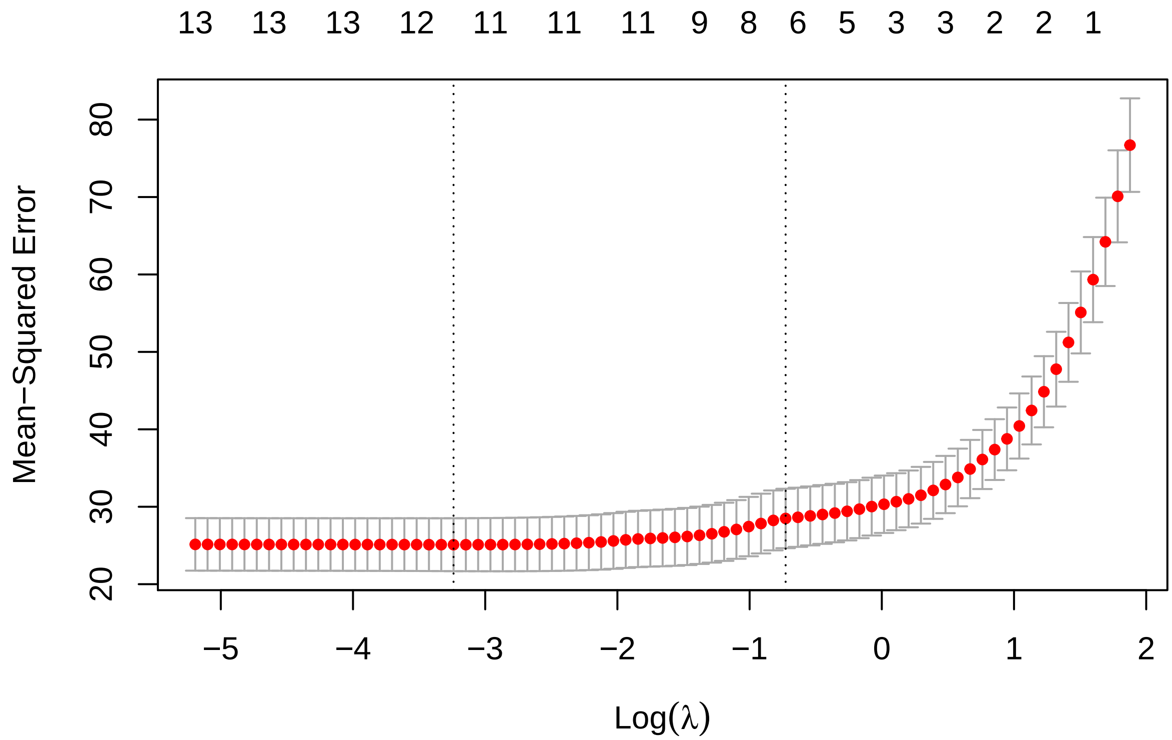
```
## [1] 0.4830841
```

```r
# Predictions
pred2.1se<- predict(cv2, newx = x.test, s = "lambda.1se")

# MSPE (prediction error)
cv2_mse <- mean((y.test-pred2.1se)^2)
cv2_mse
```

```
## [1] 30.36194
```

```r
# Deviance explained
plot(cv2, xvar = "dev", label = TRUE)
```

```
## Warning in plot.window(...): "xvar" is not a graphical parameter

## Warning in plot.window(...): "label" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "xvar" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "label" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not
## a graphical parameter

## Warning in box(...): "xvar" is not a graphical parameter

## Warning in box(...): "label" is not a graphical parameter

## Warning in title(...): "xvar" is not a graphical parameter

## Warning in title(...): "label" is not a graphical parameter
```




```
cv2_var_expl <- 100-(cv2_mse*100 )/var(y )
cv2_var_expl
```

```
## [1] 60.58064
```

The MSE is smaller in ridge reg than lasso, which means that ridge explains teh test data better

(f).

```
# Run lm
fit_lm <- lm(medv~., train)
mse_lm <- mean((test$medv - predict.lm(fit_lm, test)) ^ 2)
mse_lm
```

```
## [1] 23.21952
```

LM has lowest mse, followed by ridge and then LASSO, which means linear model should be used here instead of penalised reg.

(g).

```
# Using lamda.min (ridge)
ridge_fit <- glmnet( x, y, lambda = cv1$lambda.min, alpha = 0 )

# Using lamda.1se (LASSO)
lasso_fit <- glmnet( x, y, lambda = cv2$lambda.1se, alpha = 1 )

# Coefficient table
coef_tab <- data.frame(name = c('Intercept', as.vector(colnames(x))),
                       lm = as.vector(coefficients(fit_lm)),
                       ridge = as.vector(coef(ridge_fit)),
```

```r
                        lasso = as.vector(coef(lasso_fit)))
coef_tab
```

```
##          name          lm        ridge        lasso
## 1  Intercept  35.600399934  26.661237462 13.772362002
## 2       crim  -0.072705090  -0.057161649  0.000000000
## 3         zn   0.057289804   0.040626772  0.000000000
## 4      indus   0.035551758  -0.033684787  0.000000000
## 5       chas   2.550824500   2.841137485  1.846564691
## 6        nox -16.904750167 -11.782140303 -1.104124125
## 7         rm   3.381525918   3.662805725  3.806331521
## 8        age  -0.003004460  -0.007749735  0.000000000
## 9        dis  -1.452770852  -1.073484118  0.000000000
## 10       rad   0.300110341   0.143328363  0.000000000
## 11       tax  -0.013903871  -0.006121596  0.000000000
## 12   ptratio  -0.753161956  -0.664618333 -0.538755269
## 13     black   0.008903039   0.008739638  0.004914422
## 14     lstat  -0.530553503  -0.466429559 -0.517688079
```

There are 5 such variables for whom the coefficients are set to 0 in lasso model - crim, indus, age, rad, tax

The selected lamda values chosen for the models were not so large that the penalization could be heavier, leading the coefficients towards zero. So, if not large lamda values, then perhaps variables were correlated towards wach other (multicollinearity) which is tended to be ignored by lasso but worked on heavily by ridge reg. Thats why much of teh coeff in lasso are set to zero.

Problem 3 – Solution

```r
new_dat <- read.csv("~/Dropbox/Important_Documents/Doctoral_Work/Courses/High Dimensional Stats/2019/We

# Train & Test dataset
train <- new_dat[which(new_dat$train=='1'), ]
test <- new_dat[which(new_dat$train=='0'), ]

#Standardize covariates before fitting
train_std <- scale(dplyr::select(train,-c(y1, y2, train)))
test_std <- scale(dplyr::select(test,-c(y1, y2, train)))

# Predictors only
x_pred_train <- train[, -c(1:3)]
x_pred_test <- test[, -c(1:3)]

# Outcome vars
Y1.train<- train$y1
Y2.train<- train$y2
Y1.test<- test$y1
Y2.test<- test$y2
Y.test<- Boston[-tr.ind$X1, "medv"]
```

```
##### RIDGE ####

# Ridge reg using default lambda from cv for y1
cv.y1_ridge <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 0, nfolds =
cv.y1_ridge$lambda.min # minimum lambda during CV
```

```
## [1] 7.266715
```

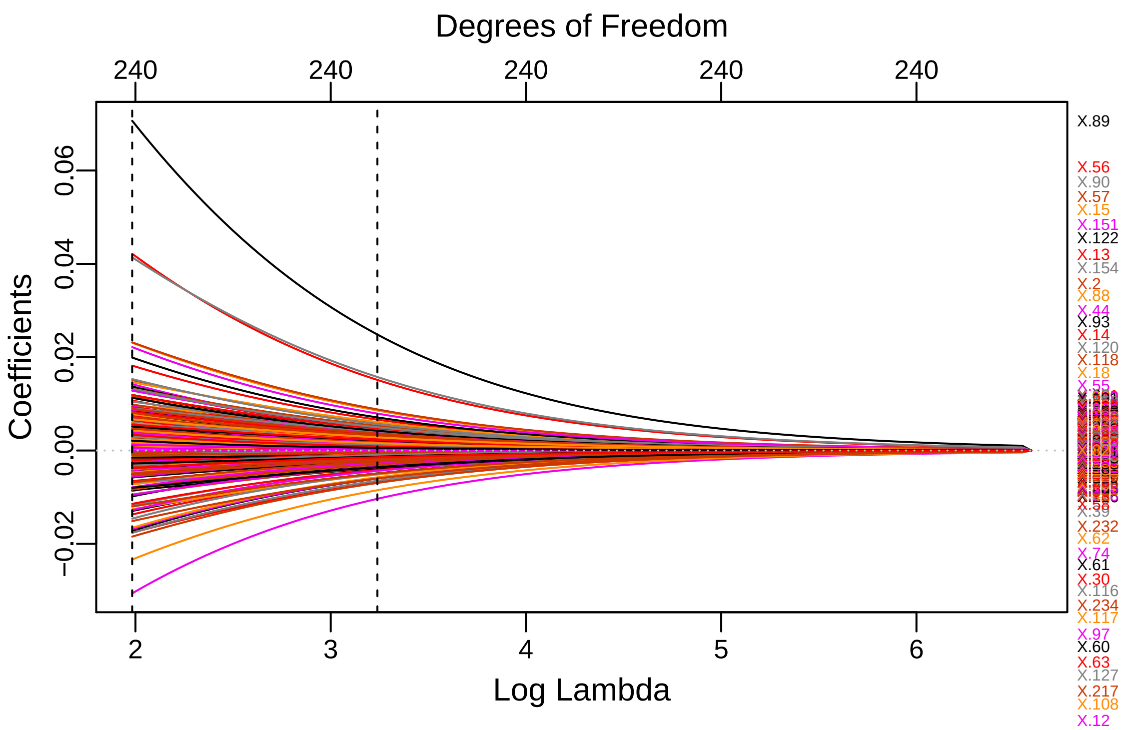
```
cv.y1_ridge$lambda.1se
```

```
## [1] 25.51483
```

```
plot_glmnet(cv.y1_ridge$glmnet.fit, label = T, xvar = "lambda")
```

```
## Warning in TeachingDemos::spread.labs(beta[iname, ncol(beta)], mindiff =
## 1.2 * : Maximum iterations reached
```

```
abline(v = log(cv.y1_ridge$lambda.min), lty = 2)
abline(v = log(cv.y1_ridge$lambda.1se), lty = 2)
```



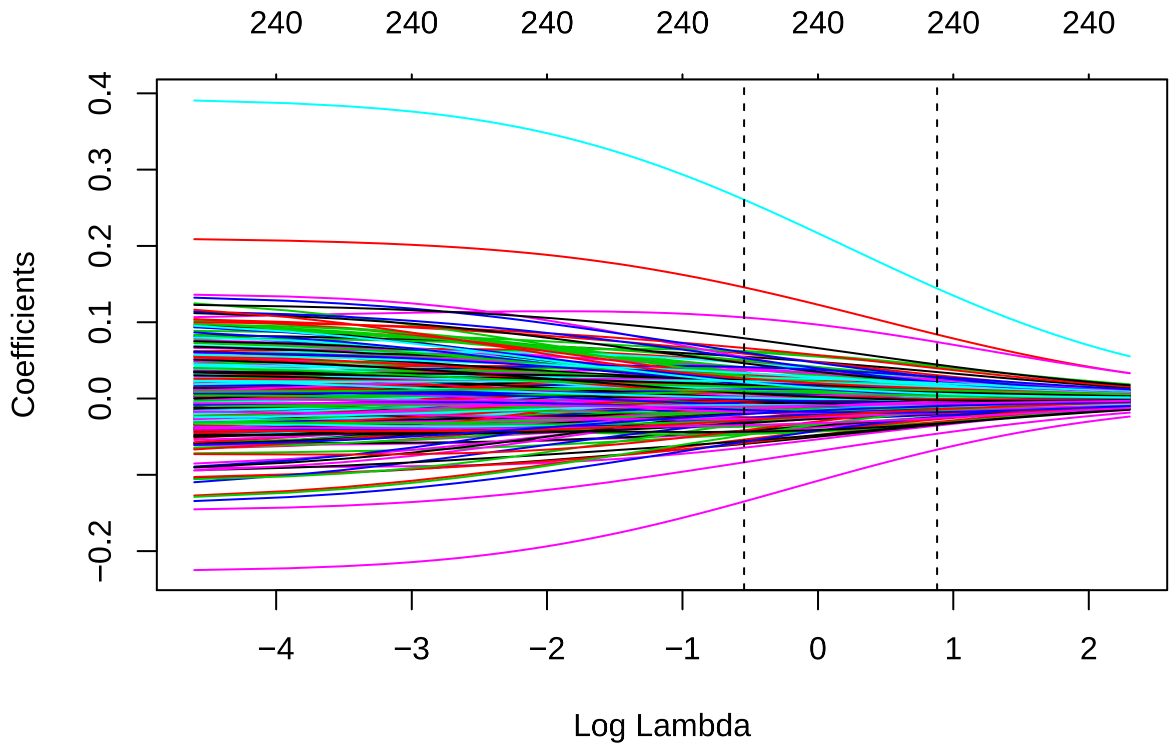

```
# Ridge reg after adjusting lambda for y1
cv.y1_ridge_adjust <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 0, r
cv.y1_ridge_adjust$lambda.min # minimum lambda during CV
```

```
## [1] 0.58
```

```
cv.y1_ridge_adjust$lambda.1se
```

```
## [1] 2.41
```

```
plot(cv.y1_ridge_adjust$glmnet.fit, label = F, xvar = "lambda")
abline(v = log(cv.y1_ridge_adjust$lambda.min), lty = 2)
abline(v = log(cv.y1_ridge_adjust$lambda.1se), lty = 2)
```




```
# MSE
pred_ridge <- predict(cv.y1_ridge_adjust, newx = as.matrix(x_pred_test))
ridge.MSE.y1 <- mean((as.vector(Y1.test) - pred_ridge)^2)
ridge.MSE.y1
```

```
## [1] 0.7479603
```

```
# Ridge reg using default lambda from cv for y2
cv.y2_ridge <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 0, nfolds =
cv.y2_ridge$lambda.min # minimum lambda during CV
```

```
## [1] 7.266715
```

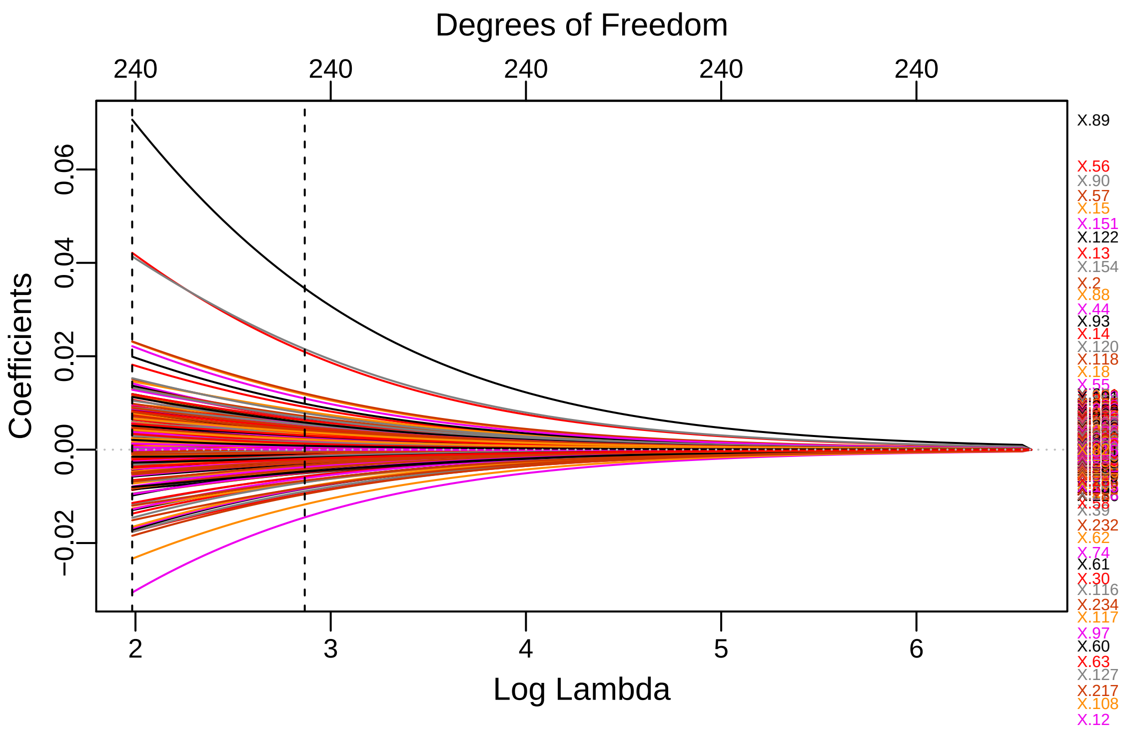
```
cv.y2_ridge$lambda.1se
```

```
## [1] 17.58638
```

```
plot_glmnet(cv.y2_ridge$glmnet.fit, label = T, xvar = "lambda")
```

```
## Warning in TeachingDemos::spread.labs(beta[iname, ncol(beta)], mindiff =
## 1.2 * : Maximum iterations reached
```

```
abline(v = log(cv.y2_ridge$lambda.min), lty = 2)
abline(v = log(cv.y2_ridge$lambda.1se), lty = 2)
```



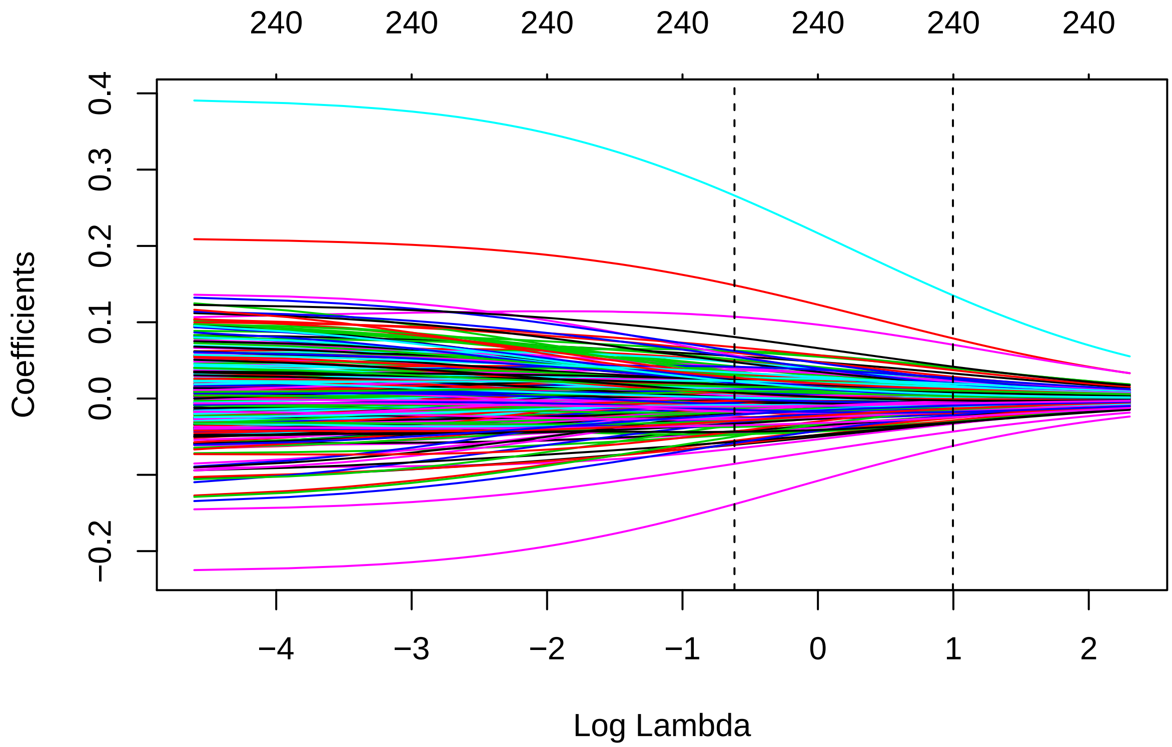

```
# Ridge reg after adjusting lambda for y2
cv.y2_ridge_adjust <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 0,
cv.y2_ridge_adjust$lambda.min # minimum lambda during CV
```

```
## [1] 0.54
```

```
cv.y2_ridge_adjust$lambda.1se
```

```
## [1] 2.71
```

```
plot(cv.y2_ridge_adjust$glmnet.fit, label = F, xvar = "lambda")
abline(v = log(cv.y2_ridge_adjust$lambda.min), lty = 2)
abline(v = log(cv.y2_ridge_adjust$lambda.1se), lty = 2)
```




```
# MSE
pred_ridge <- predict(cv.y2_ridge_adjust, newx = as.matrix(x_pred_test))
ridge.MSE.y2 <- mean((as.vector(Y1.test) - pred_ridge)^2)
ridge.MSE.y2
```

```
## [1] 0.7590902
```

```
##### LASSO ####

# LASSO using default lambda from cv for y1
cv.y1_lasso <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 1, nfolds =
cv.y1_lasso$lambda.min # minimum lambda during CV
```

```
## [1] 0.07437708
```

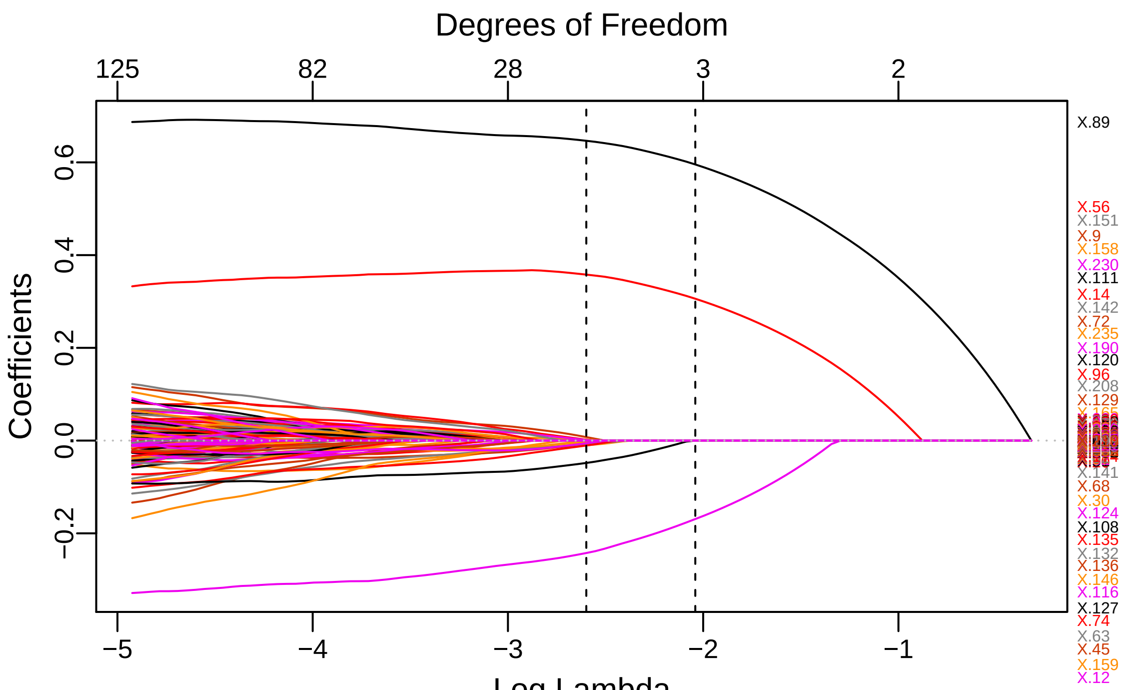
```
cv.y1_lasso$lambda.1se
```

```
## [1] 0.1299761
```

```
plot_glmnet(cv.y1_lasso$glmnet.fit, label = T, xvar = "lambda")
```

```
## Warning in TeachingDemos::spread.labs(beta[iname, ncol(beta)], mindiff =
## 1.2 * : Maximum iterations reached
```

```
abline(v = log(cv.y1_lasso$lambda.min), lty = 2)
abline(v = log(cv.y1_lasso$lambda.1se), lty = 2)
```



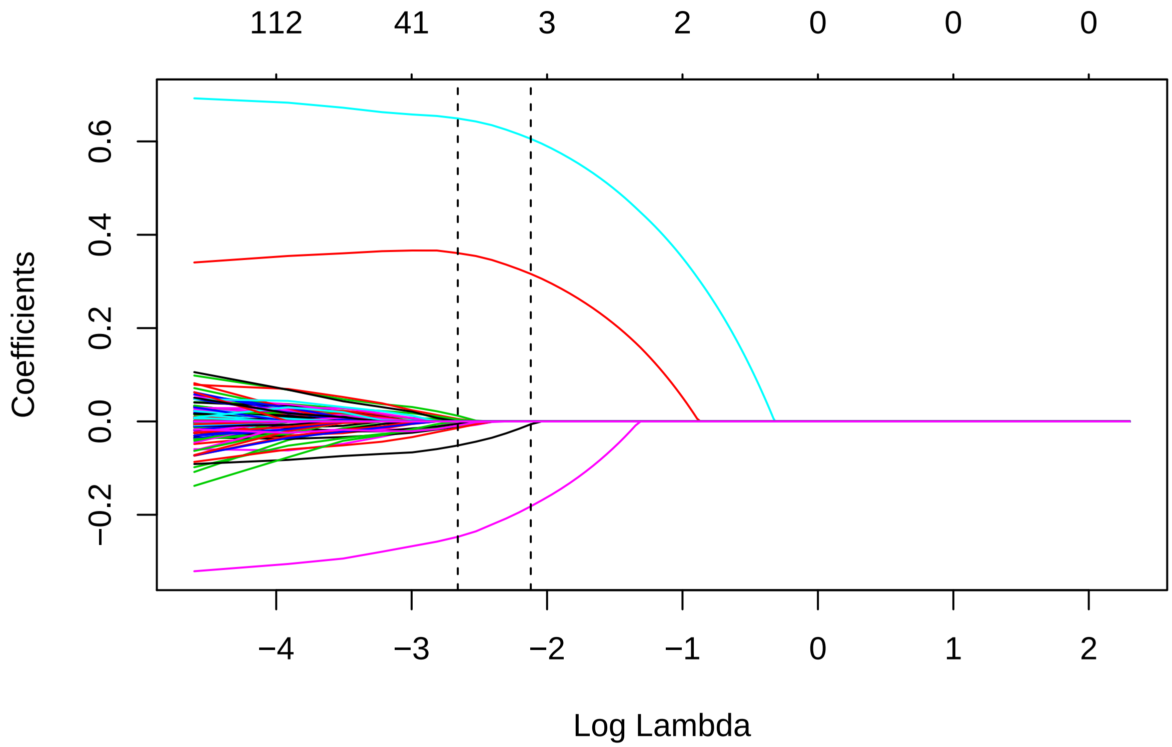

```
# LASSO after adjusting lambda for y1
cv.y1_lasso_adjust <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 1, r
cv.y1_lasso_adjust$lambda.min # minimum lambda during CV
```

```
## [1] 0.07
```

```
cv.y1_lasso_adjust$lambda.1se
```

```
## [1] 0.12
```

```
plot(cv.y1_lasso_adjust$glmnet.fit, label = F, xvar = "lambda")
abline(v = log(cv.y1_lasso_adjust$lambda.min), lty = 2)
abline(v = log(cv.y1_lasso_adjust$lambda.1se), lty = 2)
```




```
# MSE
pred_lasso <- predict(cv.y1_lasso_adjust, newx = as.matrix(x_pred_test))
lasso.MSE.y1 <- mean((as.vector(Y1.test) - pred_lasso)^2)
lasso.MSE.y1
```

```
## [1] 0.2456486
```

```
# LASSO using default lambda from cv for y2
cv.y2_lasso <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 1, nfolds =
cv.y2_lasso$lambda.min # minimum lambda during CV
```

```
## [1] 0.08162875
```

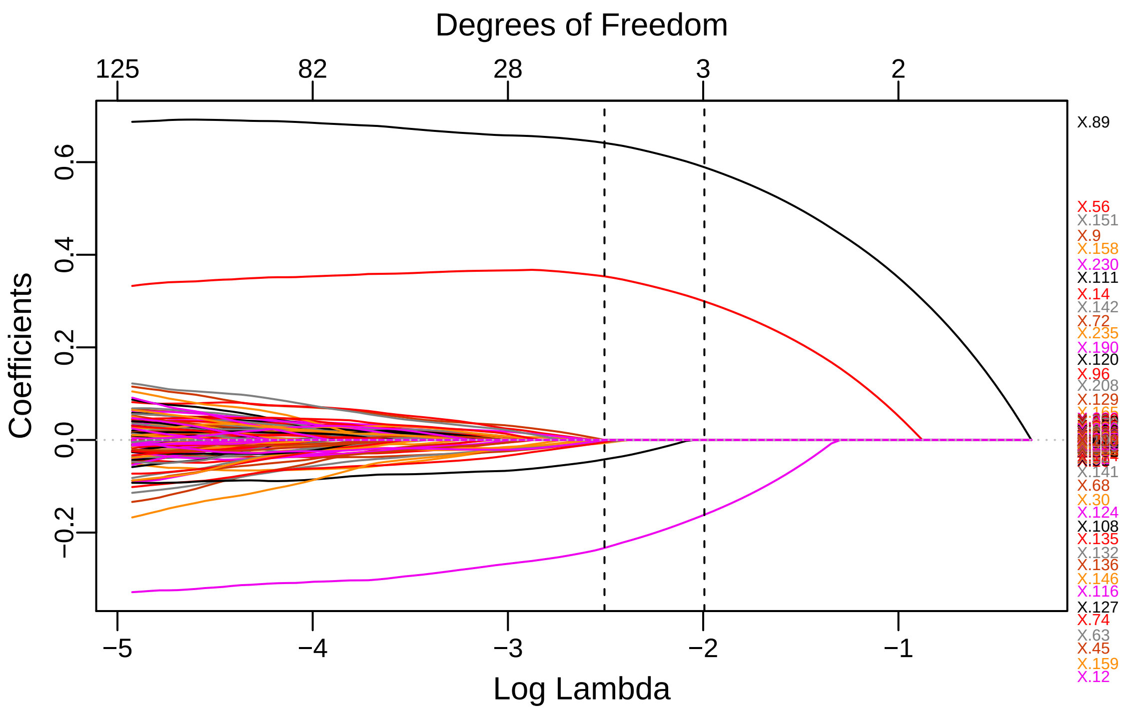
```
cv.y2_lasso$lambda.1se
```

```
## [1] 0.136165
```

```
plot_glmnet(cv.y2_lasso$glmnet.fit, label = T, xvar = "lambda")
```

```
## Warning in TeachingDemos::spread.labs(beta[iname, ncol(beta)], mindiff =
## 1.2 * : Maximum iterations reached
```

```
abline(v = log(cv.y2_lasso$lambda.min), lty = 2)
abline(v = log(cv.y2_lasso$lambda.1se), lty = 2)
```



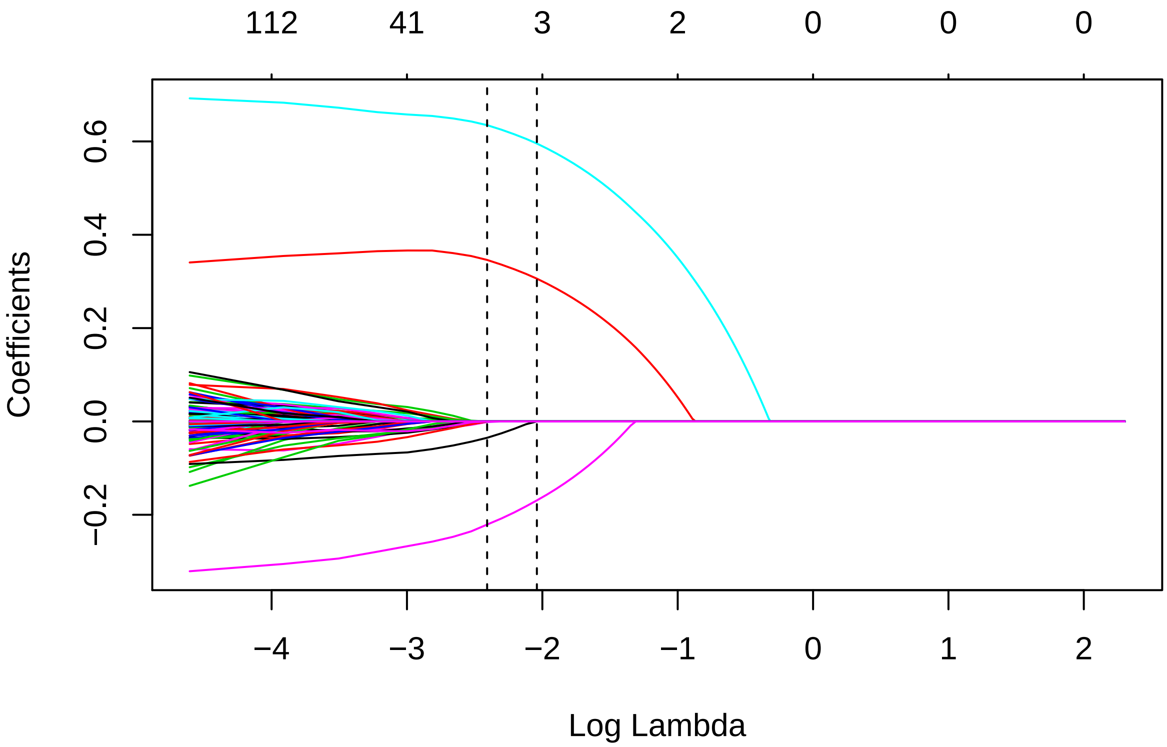

```
# LASSO after adjusting lambda for y2
cv.y2_lasso_adjust <- cv.glmnet(x=as.matrix(x_pred_train), y=Y1.train, family = "gaussian", alpha = 1, r
cv.y2_lasso_adjust$lambda.min # minimum lambda during CV
```

```
## [1] 0.09
```

```
cv.y2_lasso_adjust$lambda.1se
```

```
## [1] 0.13
```

```
plot(cv.y2_lasso_adjust$glmnet.fit, label = F, xvar = "lambda")
abline(v = log(cv.y2_lasso_adjust$lambda.min), lty = 2)
abline(v = log(cv.y2_lasso_adjust$lambda.1se), lty = 2)
```




```
# MSE
pred_lasso <- predict(cv.y2_lasso_adjust, newx = as.matrix(x_pred_test))
lasso.MSE.y2 <- mean((as.vector(Y1.test) - pred_lasso)^2)
lasso.MSE.y2
```

```
## [1] 0.2534553
```

```
### MSE from both methods
MSE_tot <- data.frame(method = c("ridge.y1", "ridge.y2", "lasso.y1", "lasso.y2"), MSE = c(ridge.MSE.y1,
MSE_tot
```

```
##     method       MSE
## 1 ridge.y1 0.7479603
## 2 ridge.y2 0.7590902
## 3 lasso.y1 0.2456486
## 4 lasso.y2 0.2534553
```

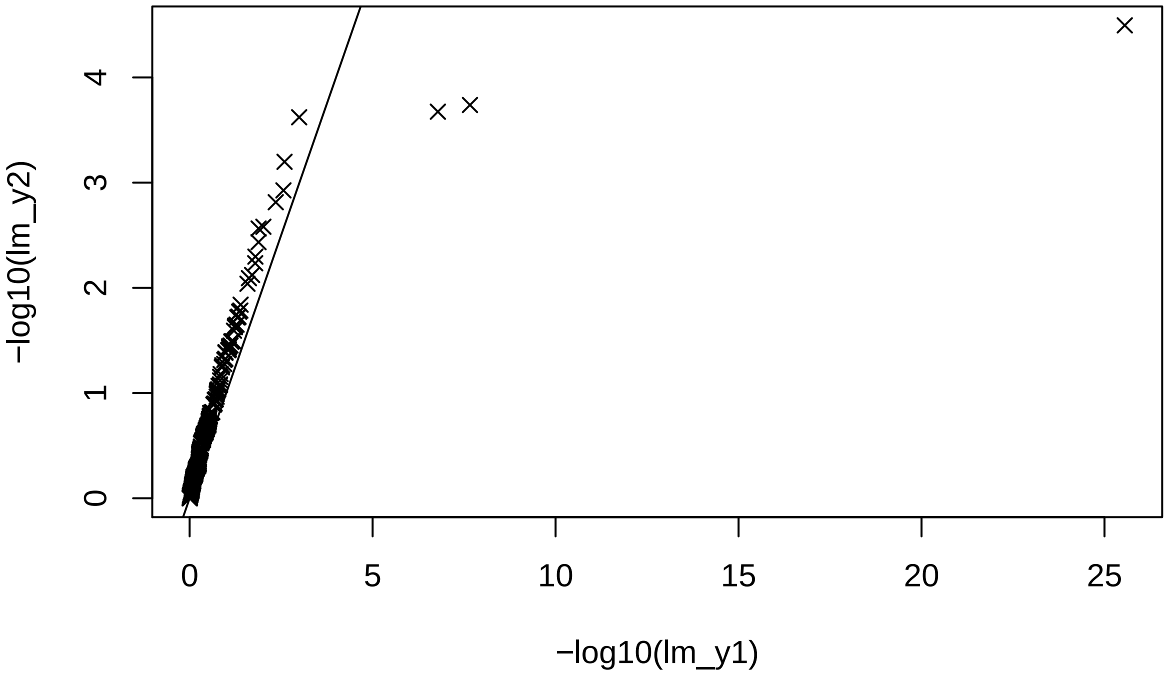
According to MSE table above, we see that lasso regression is preferred for y1 and ridge regression is preferred for y2

Problem 4 – Solution

(a)

```
dat <- read.csv("~/Dropbox/Important_Documents/Doctoral_Work/Courses/High Dimensional Stats/2019/Week 4,


lm_y1 <- apply(as.matrix(x_pred_train), 2, function(x)summary(lm(Y1.train ~ x ) )$coeff[2,4])
lm_y2 <- apply(as.matrix(x_pred_train), 2, function(x)summary(lm(Y2.train ~ x ) )$coeff[2,4])
```

```r
qqplot(-log10(lm_y1), -log10(lm_y2), pch = 4)
abline(0,1)
```



We see from the qqplot that p-values for y1 are higher than the for y2. As the p-values for y2 are smaller which would then mean that more significant predictors for y2 are found than for y1. Hence, we should use ridge regression for y2 and lasso for y1.

Junk COde

imp <- as.data.frame(varImp(cv_glmnet)) imp <- data.frame(overall = imp$Overall, names = rownames(imp))imp[order(imp$overall,decreasing = T),]

varImp <- function(object, lambda = NULL, …) {

skipping a few lines

beta <- predict(object, s = lambda, type = "coef") if(is.list(beta)) { out <- do.call("cbind", lapply(beta, function(x) x[,1])) out <- as.data.frame(out) } else out <- data.frame(Overall = beta[,1]) out <- abs(out[rownames(out) != "(Intercept)",,drop = FALSE]) out }

Compare the models and see which variables agree

var_step = names(fit_lm$coefficients)[-1]var_lasso = colnames(train)[which(coef(fit, s = cv.lasso$lambda.min)!=0)-1] intersect(var_step,var_lasso)