STAT 542 / CS 598: Homework 2

Fall 2019, by John Moran (jfmoran2)

Due: Monday, Sep 23 by 11:59 PM Pacific Time

Contents

Question 1 [20 Points] Linear Model Selection

We will use the Boston Housing data again. This time, we do not scale the covariate. We will still remove `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. If you do not use R, you can download a '.csv' file from the course website.

```
library(mlbench)
data(BostonHousing2)
BH = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract"))]
full.model <- lm(cmedv ~ ., data=BH)
```

Answer the following questions:

a. [5 Points] Report the most significant variable from this full model with all features.

```
pvalues <- summary(full.model)$coefficients[,4]
most.sig.var <- pvalues[which.min(pvalues)]
print(sprintf("Most significant variable: %s with value: %g",
              names(pvalues[which.min(pvalues)]), most.sig.var))
```

```
## [1] "Most significant variable: lstat with value: 5.27442e-24"
```

b. [5 Points] Starting from this full model, use stepwise regression with both forward and backward and BIC criterion to select the best model. Which variables are removed from the full model?

```
# n = nrows in Boston Housing Data
n=dim(BH)[1]

# k=log(n) does BIC
stepBIC = step(full.model, direction="both", k=log(n), trace=0)
sel.var.BIC = attr(stepBIC$terms, "term.labels")

# names(BH) %in% sel.var.BIC will give T/F for each name in BH that is or isn't in sel.var.BC
# find names that are in BH but not in sel.var.BIC
removed <- names(BH[!(names(BH) %in% sel.var.BIC)])
removed <- removed[removed != "cmedv"]
print("Vars removed from full model: ")
```

```
## [1] "Vars removed from full model: "
```

```
print(removed)
```

```
## [1] "lon"   "lat"   "indus" "age"
```

c. [5 Points] Starting from this full model, use the best subset selection and list the best model of each model size.

```r
library(leaps)
p=dim(BH)[2] - 1 # don't count cmedv
b = regsubsets(cmedv ~ ., data=BH, nvmax = p)
rs = summary(b)

#which: A logical matrix indicating which elements are in each model
model.elements <- rs$which

# take away intercept since it is in all models
model.elements <- model.elements[,-1]
best.models.names <- apply(model.elements, 1, function(x) names(which(x)))
best.models.names
```

```
## $`1`
## [1] "lstat"
##
## $`2`
## [1] "rm"    "lstat"
##
## $`3`
## [1] "rm"      "ptratio" "lstat"
##
## $`4`
## [1] "rm"      "dis"     "ptratio" "lstat"
##
## $`5`
## [1] "nox"     "rm"      "dis"     "ptratio" "lstat"
##
## $`6`
## [1] "chas1"   "nox"     "rm"      "dis"     "ptratio" "lstat"
##
## $`7`
## [1] "chas1"   "nox"     "rm"      "dis"     "ptratio" "b"       "lstat"
##
## $`8`
## [1] "zn"      "chas1"   "nox"     "rm"      "dis"     "ptratio" "b"       "lstat"
##
## $`9`
## [1] "chas1"   "nox"     "rm"      "dis"     "rad"     "tax"     "ptratio" "b"
## [9] "lstat"
##
## $`10`
##  [1] "crim"    "zn"      "nox"     "rm"      "dis"     "rad"     "tax"     "ptratio"
##  [9] "b"       "lstat"
##
## $`11`
##  [1] "crim"    "zn"      "chas1"   "nox"     "rm"      "dis"     "rad"     "tax"
##  [9] "ptratio" "b"       "lstat"
##
## $`12`
##  [1] "lat"     "crim"    "zn"      "chas1"   "nox"     "rm"      "dis"     "rad"
##  [9] "tax"     "ptratio" "b"       "lstat"
```

```
##
## $`13`
##  [1] "lon"     "lat"     "crim"    "zn"      "chas1"   "nox"     "rm"      "dis"
##  [9] "rad"     "tax"     "ptratio" "b"       "lstat"
##
## $`14`
##  [1] "lon"     "lat"     "crim"    "zn"      "indus"   "chas1"   "nox"     "rm"
##  [9] "dis"     "rad"     "tax"     "ptratio" "b"       "lstat"
##
## $`15`
##  [1] "lon"     "lat"     "crim"    "zn"      "indus"   "chas1"   "nox"     "rm"
##  [9] "age"     "dis"     "rad"     "tax"     "ptratio" "b"       "lstat"
```

d. [5 Points] Use the Cp criterion to select the best model from part c). Which variables are removed from the full model? What is the most significant variable?

```
cp.best.names <- best.models.names[which.min(rs$cp)]

print(sprintf("Best model from part c) is the one with %d vars", which.min(rs$cp)))
```

```
## [1] "Best model from part c) is the one with 11 vars"
```

```
cp.removed <- names(BH[!(names(BH) %in% cp.best.names)])
cp.removed <- removed[removed != "cmedv"]
print("The following variables are removed from the full model:")
```

```
## [1] "The following variables are removed from the full model:"
```

```
print(cp.removed)
```

```
## [1] "lon"   "lat"   "indus" "age"
```

```
param.names <- as.data.frame(cp.removed)
p <- NULL
for (tmp in param.names) {
  p <- paste(p, tmp, sep="-", collapse="")
}

f <-as.formula(paste("cmedv ~ . ", p, collapse=""))

cp.model <- lm(f, data=BH)

pvalues <- summary(cp.model)$coefficients[,4]
most.sig.var <- pvalues[which.min(pvalues)]

print(sprintf("Most significant variable: %s with value: %g",
              names(pvalues[which.min(pvalues)]), most.sig.var))
```

```
## [1] "Most significant variable: lstat with value: 2.85504e-26"
```

Question 2 (50 Points) Code Your Own Lasso

For this question, we will write our own Lasso code. You are not allowed to use any built-in package that already implements Lasso. First, we will generate simulated data. Here, only X_1, X_2 and X_3 are important, and we will not consider the intercept term.

```r
library(MASS)
set.seed(1)
n = 200
p = 200

# generate data
V = matrix(0.2, p, p)
diag(V) = 1
X = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
y = X[, 1] + 0.5*X[, 2] + 0.25*X[, 3] + rnorm(n)

# we will use a scaled version
X = scale(X)
y = scale(y)
```

As we already know, coordinate descent is an efficient approach for solving Lasso. The algorithm works by updating one parameter at a time, and loop around all parameters until convergence.

a. [10 Points] Hence, we need first to write a function that updates just one parameter, which is also known as the soft-thresholding function. Construct the function in the form of `soft_th <- function(b, lambda)`, where `b` is a number that represents the one-dimensional linear regression solution, and `lambda` is the penalty level. The function should output a scaler, which is the minimizer of

$$(x - b)^2 + \lambda|b|$$

```r
soft_th <- function(b, lambda) {
  if (b < -(lambda / 2)) {
    retval <- b + (lambda / 2)
  } else if (b > (lambda / 2)) {
    retval <- b - ((lambda / 2))
  } else {
    retval <- 0
  }

  return(retval)
}
```

b. [10 Points] Now lets pretend that at an iteration, the current parameter $\boldsymbol{\beta}$ value is given below (as `beta_old`, i.e., $\boldsymbol{\beta}^{\text{old}}$). Apply the above soft-thresholding function to update all p parameters sequencially one by one to complete one "loop" of the updating scheme. Please note that we use the Gauss-Seidel style coordinate descent, in which the update of the next parameter is based on the new values of previous entries. Hence, each time a parameter is updated, you should re-calculate the residual

$$\mathbf{r} = \mathbf{y} - \mathbf{X}^{\text{T}}\boldsymbol{\beta}$$

so that the next parameter update reflects this change. After completing this one enrire loop, print out the first 3 observations of $\mathbf{r}$ and the nonzero entries in the updated $\boldsymbol{\beta}^{\text{new}}$ vector. For this question, use `lambda` $= 0.7$ and

```r
isNonZero <- function(x) {
  ifelse (abs(x) > 1e-10, TRUE, FALSE)
}

lambda = 0.7

beta_old <- rep(0, p)
```

```r
beta_new <- rep(0, p)
residuals <- y - X %*% beta_old

xcolsum.2 <- colSums(X^2)

for (j in 1:p) {
  beta_old <- beta_new

  # exclude effect of current j by adding back
  residuals <- residuals + X[,j]*beta_old[j]

  ro <- sum(X[,j] * residuals) / xcolsum.2[j]

  beta_new[j] <- soft_th(ro, lambda)

  # restore and recalculate residuals with effect of newly calculated beta
  residuals <- residuals - X[,j]*beta_new[j]
}

print("First three residuals:")
```

```
## [1] "First three residuals:"
```

```r
print(residuals[1:3])
```

```
## [1] -0.07604338  0.14677403  0.15625677
```

```r
my.idx <- which(isNonZero(beta_new))

for (i in 1:length(my.idx)) {
  print(sprintf("Non-zero entry beta index: %d with value: %f",
                my.idx[i], beta_new[my.idx][i]))
}
```

```
## [1] "Non-zero entry beta index: 1 with value: 0.352963"
## [1] "Non-zero entry beta index: 2 with value: 0.090293"
```

c. [25 Points] Now, let us finish the entire Lasso algorithm. We will write a function `myLasso(X, y, lambda, tol, maxitr)`. Set the tolerance level `tol` = 1e-5, and `maxitr` = 100 as the default value. Use the "one loop" code that you just wrote in the previous question, and integrate that into a grand for-loop that will continue updating the parameters up to `maxitr` runs. Check your parameter updates once in this grand loop and stop the algorithm once the ℓ_1 distance between $\boldsymbol{\beta}^{\text{new}}$ and $\boldsymbol{\beta}^{\text{old}}$ is smaller than `tol`. Use `beta_old = rep(0, p)` as the initial value, and `lambda` = 0.3. After the algorithm converges, report the following: i) the number of iterations took; ii) the nonzero entries in the final beta parameter estimate, and iii) the first three observations of the residual. Please write your algorithm as efficient as possible.

```r
L1.distance <- function(a, b){
  distance <- abs(a-b)
  distance <- sum(distance)
  return(distance)
}

myLasso <- function(X, y, lambda, tol, maxitr) {

  beta_curr <- rep(0, p)
```

```r
  beta_prev <- rep(0, p)
  residuals <- y - X %*% beta_curr

  xcolsum.2 <- colSums(X^2)

  eval.tolerance <- 10^10
  tol <- 1e-5
  maxiter <- 100
  count <- 0
  tolerance.met <- FALSE

  while (!tolerance.met > tol && count < maxiter) {
    for (j in 1:p) {
      # exclude effect of current j by adding back
      residuals <- residuals + X[,j]*beta_curr[j]

      ro <- sum(X[,j] * residuals) / xcolsum.2[j]
      beta_curr[j] <- soft_th(ro, lambda)

      # restore and recalculate residuals with effect of newly calculated beta
      residuals <- residuals - X[,j]*beta_curr[j]
    }

    count <- count + 1

    eval.tolerance <- L1.distance(beta_curr, beta_prev)

    if (eval.tolerance < tol) {
      tolerance.met <- TRUE
    } else
    {
      beta_prev <- beta_curr
    }

  }

  results <- list()
  results$beta <- beta_curr
  results$residuals <- residuals
  results$iterations <- count

  return(results)
}

lambda = .3

tol <- 1e-5
maxiter <- 100
results <- myLasso(X, y, lambda, tol, maxitr)

beta <- results$beta
residuals <- results$residuals
my.idx <- which(isNonZero(beta))
```

```r
print(sprintf("Number of iterations: %d", results$iterations))
```

```
## [1] "Number of iterations: 9"
```

```r
for (i in 1:length(my.idx)) {
    print(sprintf("Non-zero entry beta index: %d with value: %f",
                  my.idx[i], beta[my.idx][i]))
}
```

```
## [1] "Non-zero entry beta index: 1 with value: 0.457802"
## [1] "Non-zero entry beta index: 2 with value: 0.226116"
## [1] "Non-zero entry beta index: 3 with value: 0.114400"
## [1] "Non-zero entry beta index: 14 with value: 0.001019"
## [1] "Non-zero entry beta index: 118 with value: 0.011551"
## [1] "Non-zero entry beta index: 137 with value: 0.004669"
```

```r
print("First three residuals:")
```

```
## [1] "First three residuals:"
```

```r
print(residuals[1:3])
```

```
## [1] -0.1757378  0.2262848  0.1912103
```

d. [5 Points] Now we have our own Lasso function, let's check the result and compare it with the `glmnet` package. Note that for the glmnet package, their `lambda` should be set as half of ours. Comment on the accuracy of the algorithm that we wrote. Please note that the distance of the two solutions should not be larger than 0.005.

```r
library(glmnet)
library(knitr)

lasso <- glmnet(X, y, alpha = 1,  lambda = .15, standardize = T)
glm.idx <- which(isNonZero(lasso$beta))

df <- data.frame(matrix(ncol = 4, nrow = length(my.idx)))
colnames(df) <- c("var index","mylasso", "glmnet", "difference")
df[,"var index"] <- my.idx
df[,"mylasso"] <- beta[my.idx]
df[,"glmnet"] <- lasso$beta[glm.idx]
df[,"difference"] <- abs(beta[my.idx] - lasso$beta[glm.idx])

kable(df, digits=9, caption="Compare mylasso and glmnet")
```

Table 1: Compare mylasso and glmnet

var index	mylasso	glmnet	difference
1	0.457802236	0.457611354	0.000190882
2	0.226116017	0.225945438	0.000170579
3	0.114399954	0.114280738	0.000119217
14	0.001018992	0.000796767	0.000222225
118	0.011551407	0.011335213	0.000216194
137	0.004669249	0.004493090	0.000176159

```r
diff <- L1.distance(beta[my.idx], lasso$beta[glm.idx])
print(sprintf("The L1 distance between mylasso and glmnet is: %f", diff))
```

```
## [1] "The L1 distance between mylasso and glmnet is: 0.001095"
```

The difference of the coefficient values that mylasso produces and the coefficients that glmnet produces is less than 0.0003 for each variable, with a total L1 distance of less than 0.002 between the two algorithms. The total RMSE between the coefficient values is: 0.000186.

Question 3 (30 Points) Cross-Validation for Model Selection

We will use the Walmart Sales data provided on Kaggle. For this question, we will use only the Train.csv file. The file is also available at here.

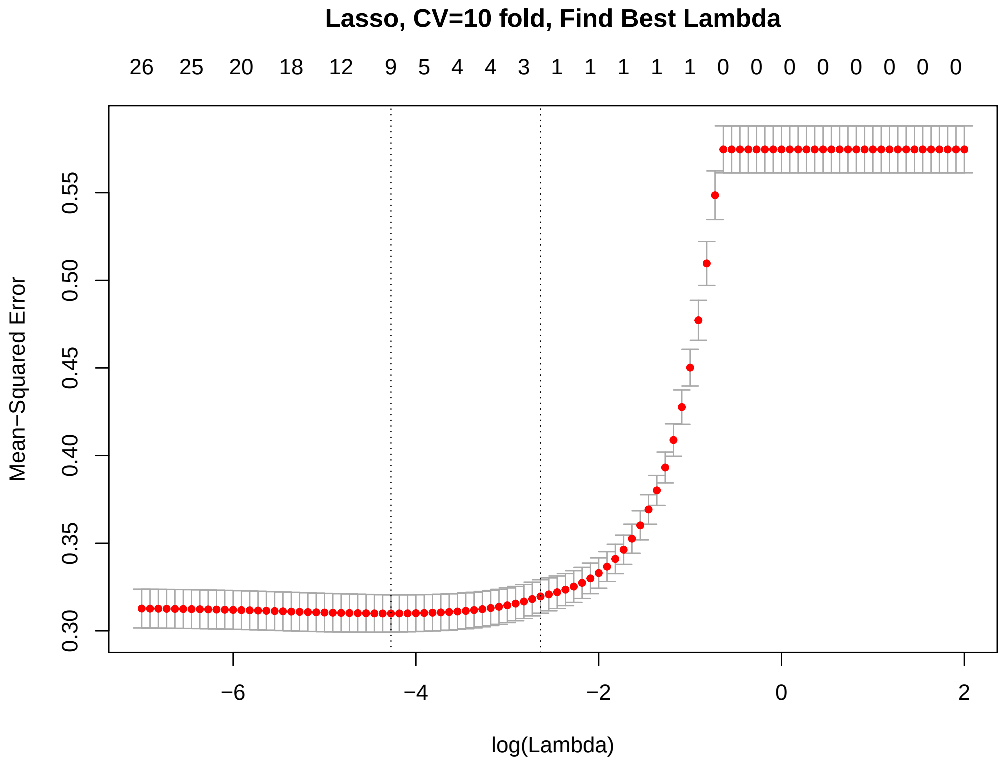
a. [10 Points] Do the following to process the data:
   - Read data into R
   - Convert character variables into factors
   - Remove `Item_Identifier`
   - Further convert all factors into dummy variables

```r
library(tidyverse)
library(knitr)

#walmart.sales <- read.csv(file="Train.csv",header=TRUE, sep=",")
walmart.sales <- read_csv("Train.csv")

# find character variables
char.vars <- c("Item_Fat_Content", "Item_Type", "Outlet_Identifier",
          "Outlet_Size", "Outlet_Location_Type", "Outlet_Type")

walmart.sales[char.vars] = lapply(walmart.sales[char.vars], factor)

# convert factors into dummies and remove Item_Identifier AND Outlet_Identifier,
# "~ . -1" = no intercept
walmart.data <- model.matrix( ~ . -1, data = walmart.sales[, c(-1,-7)])
```

b. [20 Points] Use all variables to model the outcome `Item_Outlet_Sales` in its *log* scale. First, we randomly split the data into two parts with equal size. Make sure that you set a random seed so that the result can be replicated. Treat one as the training data, and the other one as the testing data. For the training data, perform the following:
   - Use cross-validation to select the best Lasso model. Consider both `lambda.min` and `lambda.min`. Provide additional information to summarize the model fitting result
   - Use cross-validation to select the best Ridge model. Consider both `lambda.min` and `lambda.min`. Provide additional information to summarize the model fitting result
   - Test these four models on the testing data and report and compare the prediction accuracy

```r
set.seed(1)
num.rows <- nrow(walmart.data)

train.ind <- sample(1:num.rows, size=floor(num.rows/2))

train.data <- walmart.data[train.ind,]
train.X <- train.data[, !colnames(train.data) %in% "Item_Outlet_Sales"]
train.Y <- train.data[, "Item_Outlet_Sales"]
```

```r
test.data <- walmart.data[-train.ind,]
test.X <- test.data[, !colnames(test.data) %in% "Item_Outlet_Sales"]
test.Y <- test.data[, "Item_Outlet_Sales"]

lambda.seq <- exp(seq(-7,2, length.out = 100))

#alpha 1: lasso, alpha 0: ridge
lasso <- cv.glmnet(train.X, log(train.Y), alpha = 1, nfolds=10, lambda = lambda.seq)

plot(lasso)
title("Lasso, CV=10 fold, Find Best Lambda", line=3) # raise the title higher
```




```r
print(sprintf("Results: Lasso lambda.min: %0.7f and lambda.1se: %0.7f",
              lasso$lambda.min, lasso$lambda.1se))
```

```
## [1] "Results: Lasso lambda.min: 0.0139437 and lambda.1se: 0.0716212"
```

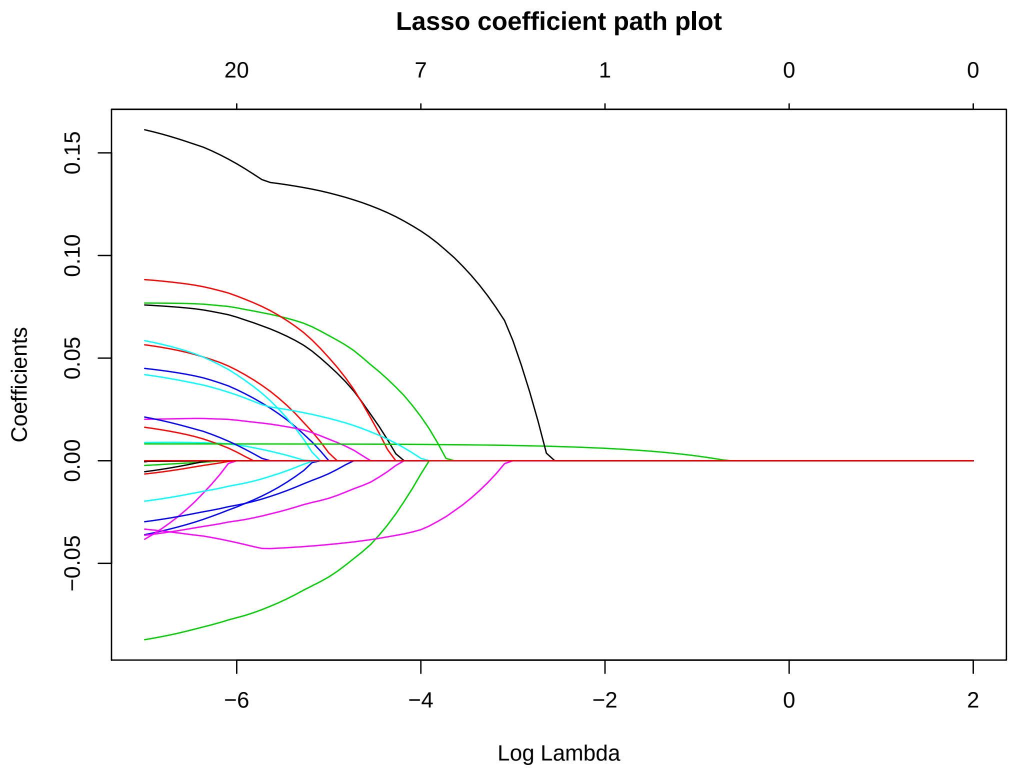
```r
print(sprintf("Degree of freedom at lambda.min: %d",
             lasso$glmnet.fit$df[which(lasso$glmnet.fit$lambda == lasso$lambda.min)] ))
```

```
## [1] "Degree of freedom at lambda.min: 9"
```

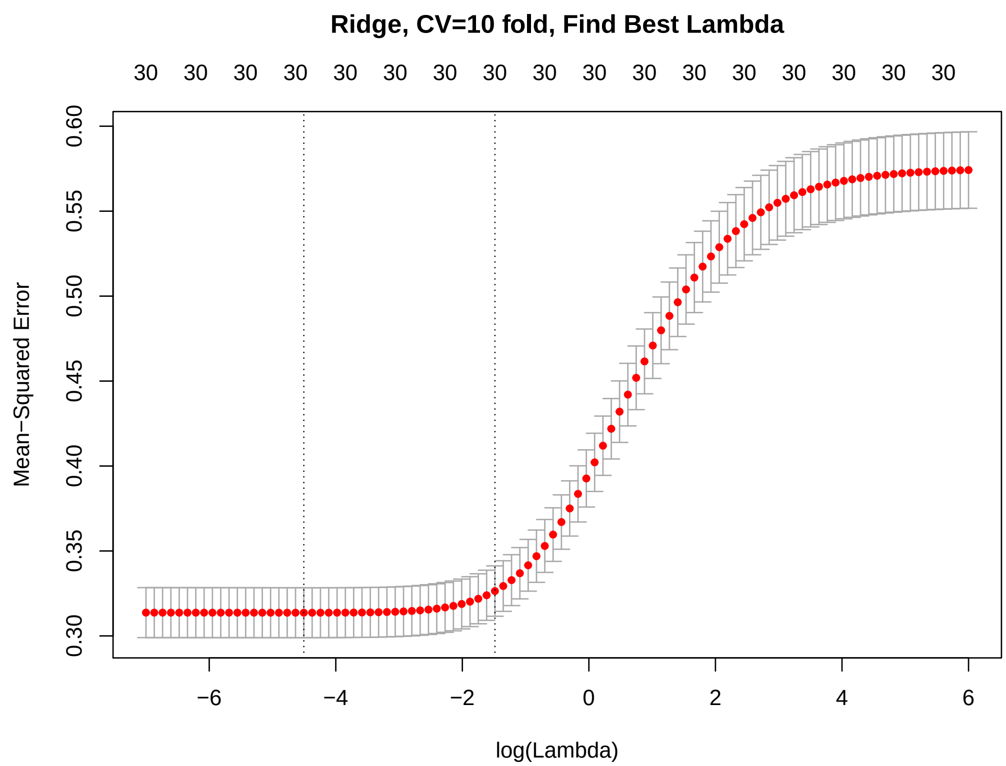
```r
print(sprintf("Degree of freedom at lambda.1se: %d",
             lasso$glmnet.fit$df[which(lasso$glmnet.fit$lambda == lasso$lambda.1se)] ))
```

```
## [1] "Degree of freedom at lambda.1se: 2"
```

```r
#coefficient path plot
plot.glmnet(lasso$glmnet.fit, label = FALSE, xvar = "lambda")
title("Lasso coefficient path plot", line=3) # raise the title higher
```



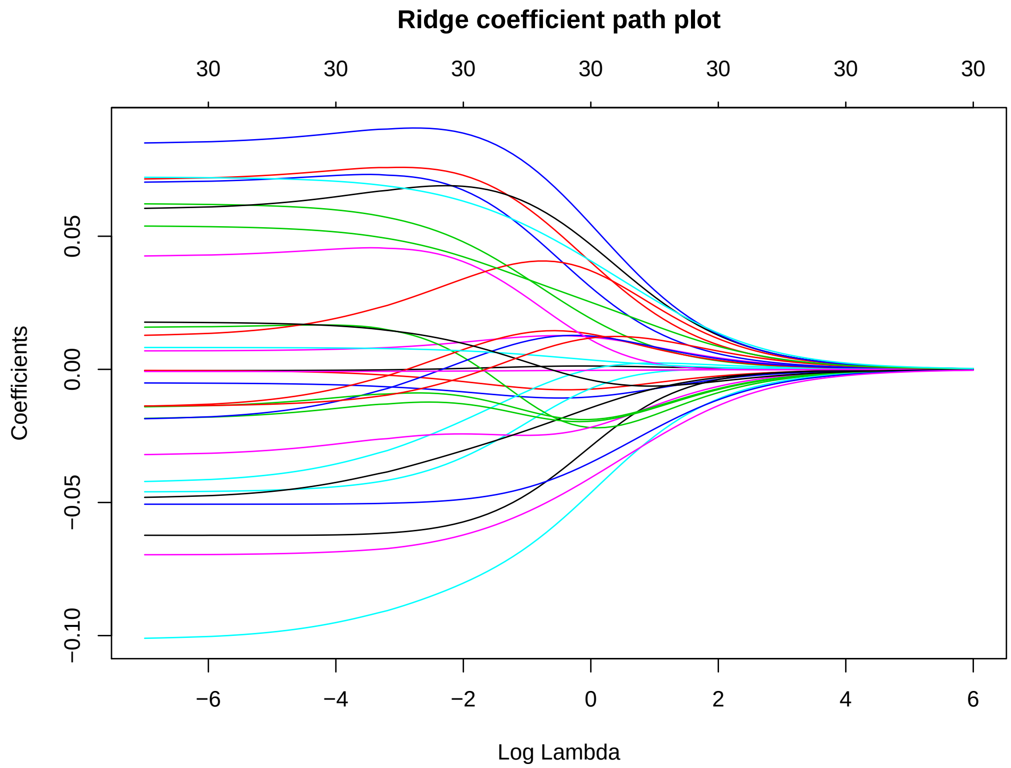

```
lambda.seq<- exp(seq(-7, 6, length.out = 100))
ridge <- cv.glmnet(train.X, log(train.Y), alpha = 0, nfolds=10, lambda = lambda.seq)
plot(ridge)
title("Ridge, CV=10 fold, Find Best Lambda", line=3) # raise the title higher
```



```r
print(sprintf("Results: Ridge lambda.min: %0.7f and lambda.1se: %0.7f",
              ridge$lambda.min, ridge$lambda.1se))
```

```
## [1] "Results: Ridge lambda.min: 0.0110530 and lambda.1se: 0.2265367"
```

```r
# coefficient path plot
plot.glmnet(ridge$glmnet.fit, label = FALSE, xvar = "lambda")
title("Ridge coefficient path plot", line=3) # raise the title higher
```




```r
lasso.lambda.min <- lasso$lambda.min
lasso.lambda.1se <- lasso$lambda.1se
ridge.lambda.min <- ridge$lambda.min
ridge.lambda.1se <- ridge$lambda.1se

Ypred.lasso.min <- predict(lasso, s=lasso.lambda.min, newx=test.X)
Ypred.lasso.min.mse <- mean((Ypred.lasso.min - log(test.Y))^2)

Ypred.lasso.1se <- predict(lasso, s=lasso.lambda.1se, newx=test.X)
Ypred.lasso.1se.mse <- mean((Ypred.lasso.1se - log(test.Y))^2)

Ypred.ridge.min <- predict(ridge, s=ridge.lambda.min, newx=test.X)
Ypred.ridge.min.mse <- mean((Ypred.ridge.min - log(test.Y))^2)

Ypred.ridge.1se <- predict(ridge, s=ridge.lambda.1se, newx=test.X)
Ypred.ridge.1se.mse <- mean((Ypred.ridge.1se - log(test.Y))^2)

df <- data.frame(matrix(ncol = 2, nrow = 4))

colnames(df) <- c("MSE", "RMSE")
rownames(df) <- c("Ypred.lasso.min",
```

```r
                  "Ypred.lasso.1se",
                  "Ypred.ridge.min",
                  "Ypred.ridge.1se")

df["Ypred.lasso.min","MSE"] <- Ypred.lasso.min.mse
df["Ypred.lasso.min","RMSE"] <- sqrt(Ypred.lasso.min.mse)
df["Ypred.lasso.1se","MSE"] <- Ypred.lasso.1se.mse
df["Ypred.lasso.1se","RMSE"] <- sqrt(Ypred.lasso.1se.mse)

df["Ypred.ridge.min","MSE"] <- Ypred.ridge.min.mse
df["Ypred.ridge.min","RMSE"] <- sqrt(Ypred.ridge.min.mse)
df["Ypred.ridge.1se","MSE"] <- Ypred.ridge.1se.mse
df["Ypred.ridge.1se","RMSE"] <- sqrt(Ypred.ridge.1se.mse)

kable(df, digits=9, caption="Result Table Lasso and Ridge")
```

Table 2: Result Table Lasso and Ridge

	MSE	RMSE
Ypred.lasso.min	0.2949604	0.5431025
Ypred.lasso.1se	0.3070113	0.5540860
Ypred.ridge.min	0.2955537	0.5436485
Ypred.ridge.1se	0.3134079	0.5598284

```r
min.MSE.index <- which.min(df[,'MSE'])
min.MSE.name <- rownames(df)[min.MSE.index]
print(sprintf("Best Model is: %s with MSE = %0.7f and RMSE = %0.7f",
              min.MSE.name, df[min.MSE.index,'MSE'], df[min.MSE.index,'RMSE']))
```

```
## [1] "Best Model is: Ypred.lasso.min with MSE = 0.2949604 and RMSE = 0.5431025"
```