lab5

Dr. Purna Gamage

2/23/2021

```r
library(ISLR)
summary(Hitters)
```

```
##      AtBat            Hits         HmRun            Runs       
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00  
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25  
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00  
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91  
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00  
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00  
##                                                                
##       RBI             Walks            Years            CAtBat       
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0  
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8  
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0  
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7  
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2  
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0  
##                                                                      
##      CHits            CHmRun           CRuns             CRBI        
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00  
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75  
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50  
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12  
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25  
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00  
##                                                                      
##      CWalks          League  Division    PutOuts          Assists     
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0  
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0  
##  Median : 170.50                    Median : 212.0   Median : 39.5  
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9  
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0  
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0  
##                                                                     
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176    
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146    
##  Median : 6.00   Median : 425.0            
##  Mean   : 8.04   Mean   : 535.9            
##  3rd Qu.:11.00   3rd Qu.: 750.0            
##  Max.   :32.00   Max.   :2460.0
##                   NA's   :59
```

```
set.seed(330)
```

There are some miising values here, so before we proceed we will remove them: and predict the salary of the player

```
Hitters=na.omit(Hitters)
with(Hitters,sum(is.na(Salary)))
```

```
## [1] 0
```

Ridge regression and Lasso

We will use the package 'glmnet', which does not use the model formula language, so we will set up an "x" and "y".

```
library(glmnet)
```

```
## Loading required package: Matrix
```

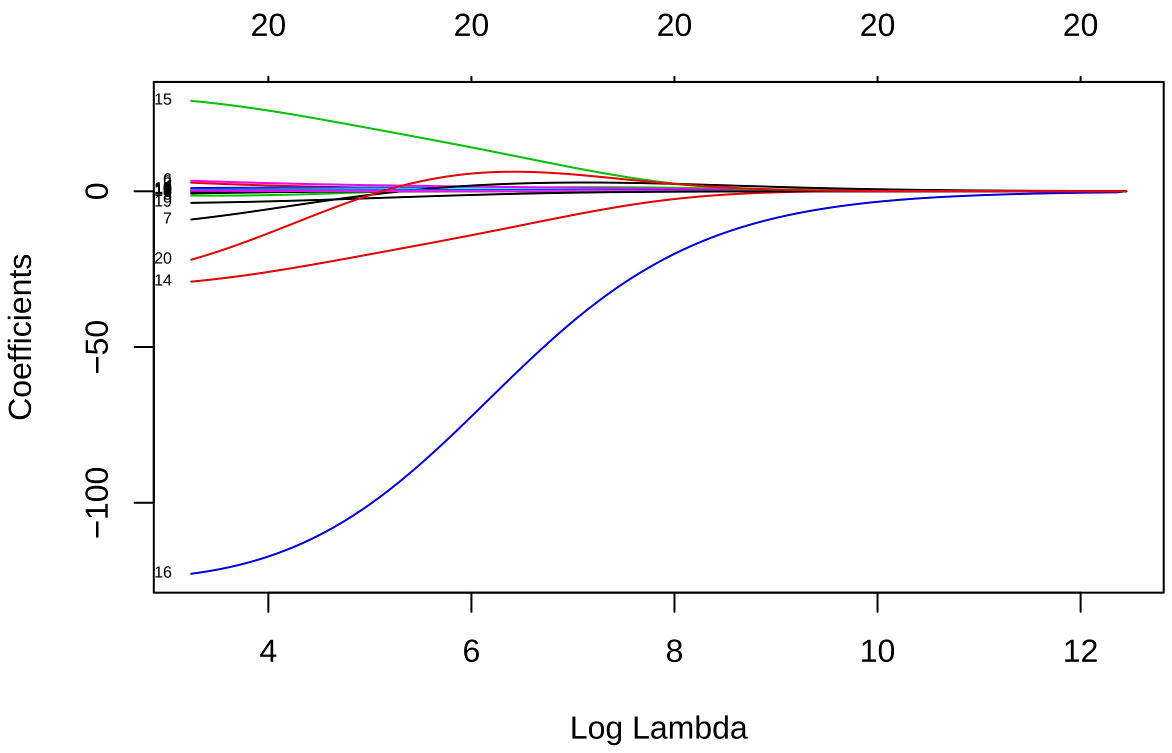
```
## Loaded glmnet 4.0-2
```

```
x=model.matrix(Salary~.-1,Hitters)
y=Hitters$Salary
```

First we will fit a ridge-regression model. This is achieved by calling "glmnet" with "alpha=0" (see the help file). There is also a 'cv.glmnet' function which will do the cross-validation for us.

$$RSS + ((1-\alpha)l_2(\beta) + \alpha l_1(\beta)$$

```
fit.ridge=glmnet(x,y,alpha=0)
plot(fit.ridge,xvar="lambda",label = TRUE)
```



this gives a plot of log lamda and what it is plotting are the coefficients.

Recall: Ridge regression models are penalized by the sum of squares of the coefficients and that's controlled by the parameter lambda.

So,the criterian for Ridge regression is:

$$RSS + \lambda \sum_j (\beta_j)^2$$

When lambda gets larger coefficients are pushed to be 0. You can see that in the plot.

When lambda log equals 12 all the coefficients are essentially 0.

As we relax the lambda, coefficients move away from lambda smoothly and sum of squares of coefficients are geting bigger.

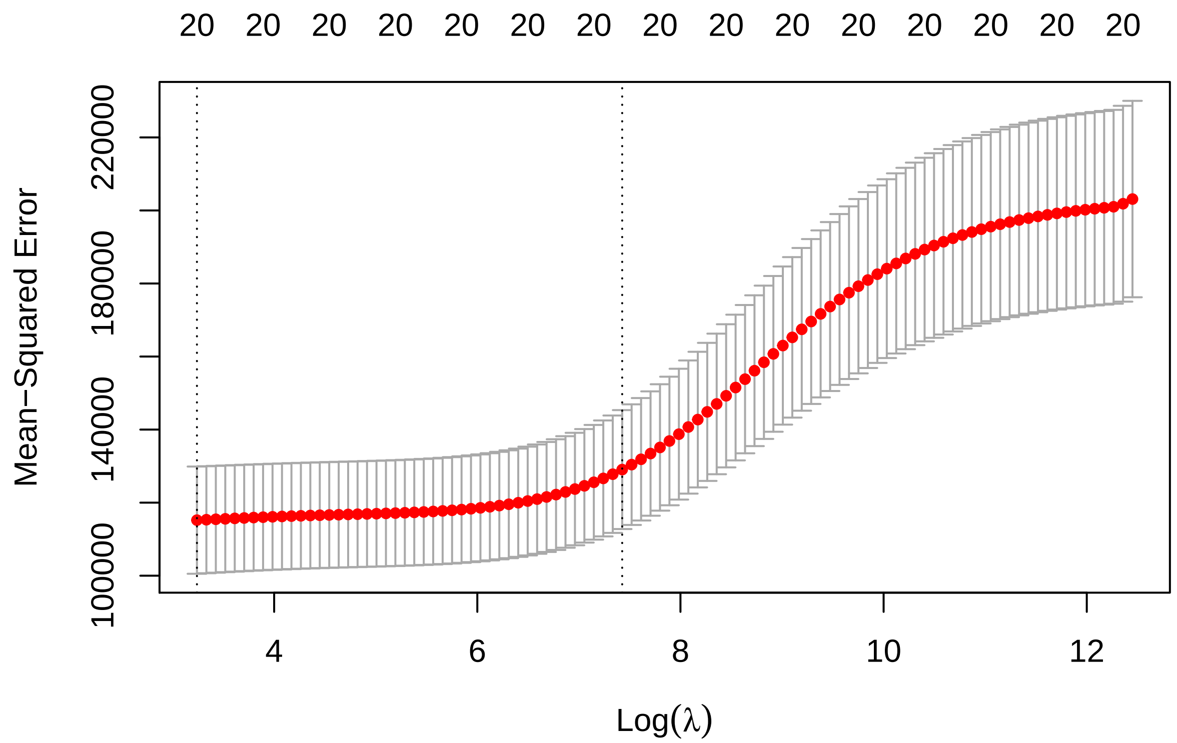
We reach a point where lamba is effectively 0 and the coefficients are unregularized.

These are the coefficients you get from an ordinary least squares fit of these variables.

Unlike stepwise methods where they control the complexity of model by restricting the number of variables, Ridge regression keeps all the variables in and shrinks the coefficients towards 0.

A buit-in function called 'cv.glmnet' will do cross validation stand by:10 fold CV

```
cv.ridge=cv.glmnet(x,y,alpha=0)
plot(cv.ridge)
```



plot of the cross validated mean squared errors.

You can see for higher lambda, (coefficients are restricted to be small) high MSE and at some pont it knids of levels off, saying that the full model is doing a good job.

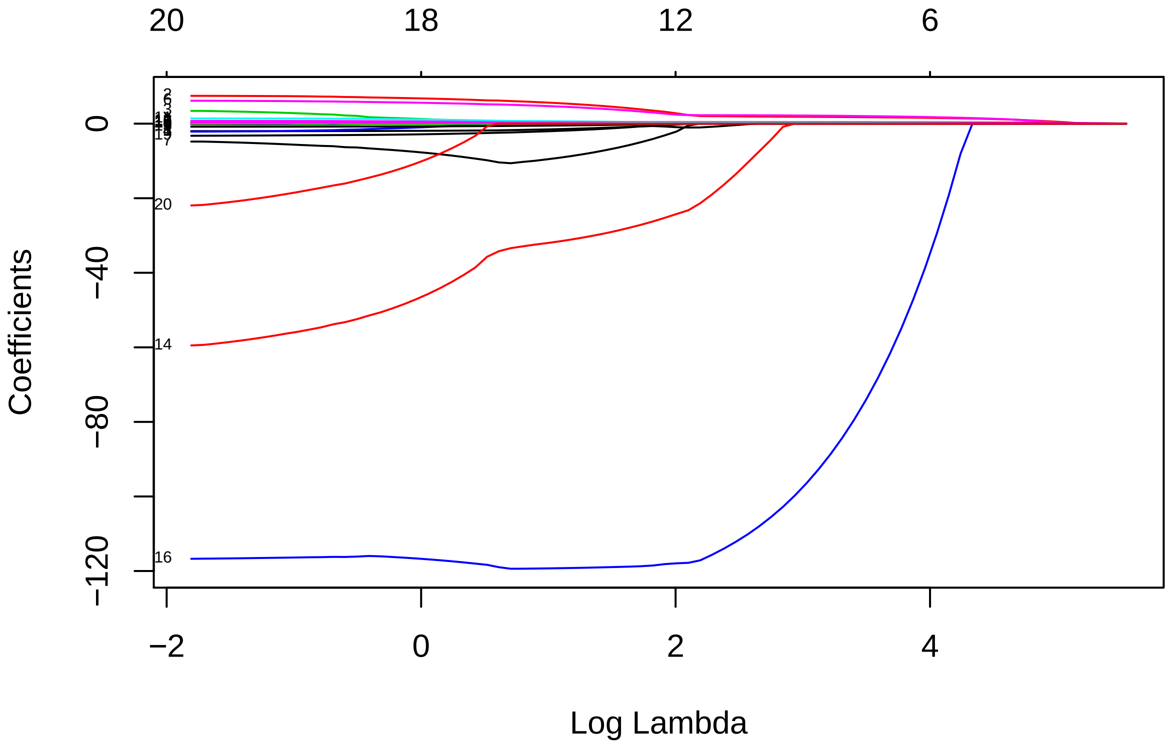
Two verticle lines: one at the minimimum and the other is at one std. error of the minimum within.

LASSO

Recall: LASSO is similar to Ridge regression but the difference was the penalty. By panelizing the coefficients with absolute function pushes some of the coefficients to be exactly 0.

$$RSS + \lambda \sum_j |\beta_j|$$

```
fit.lasso=glmnet(x,y)
plot(fit.lasso, xvar="lambda", label=TRUE)
```

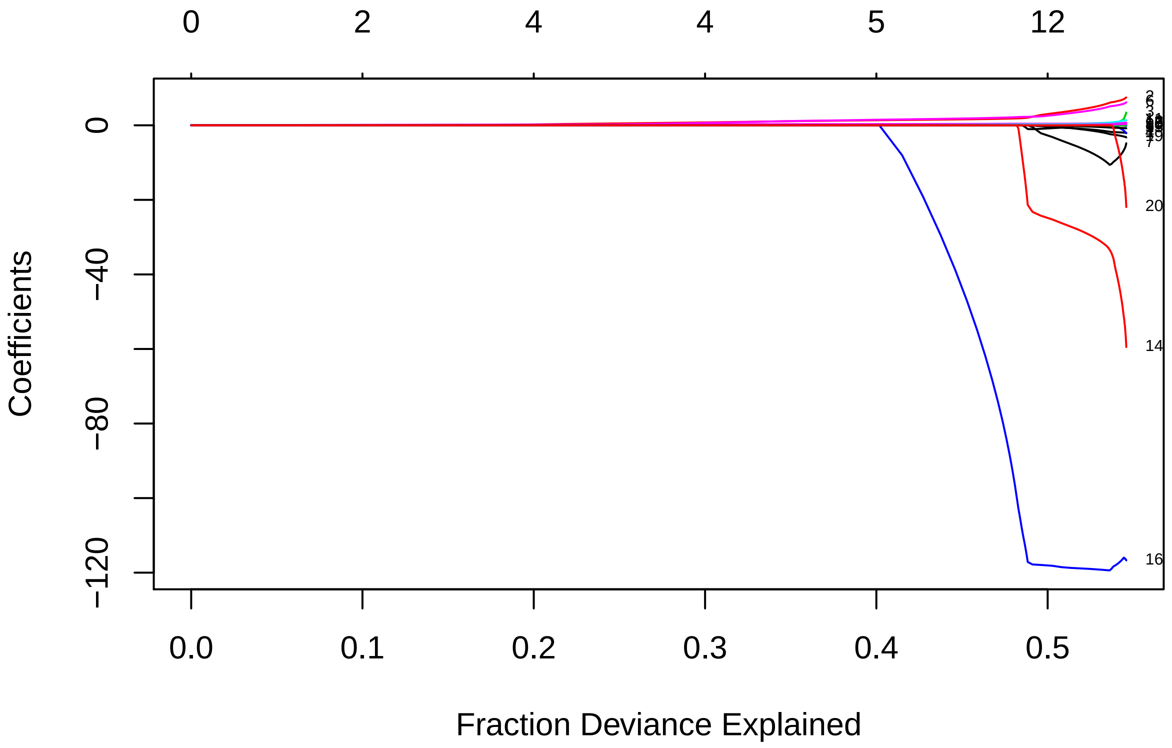


Plot: Top of the plot; how many non zero variables are in the model.

LASSO is doing both shrinkage and variable selection.

Default alpha =1 for 'glmnet'.

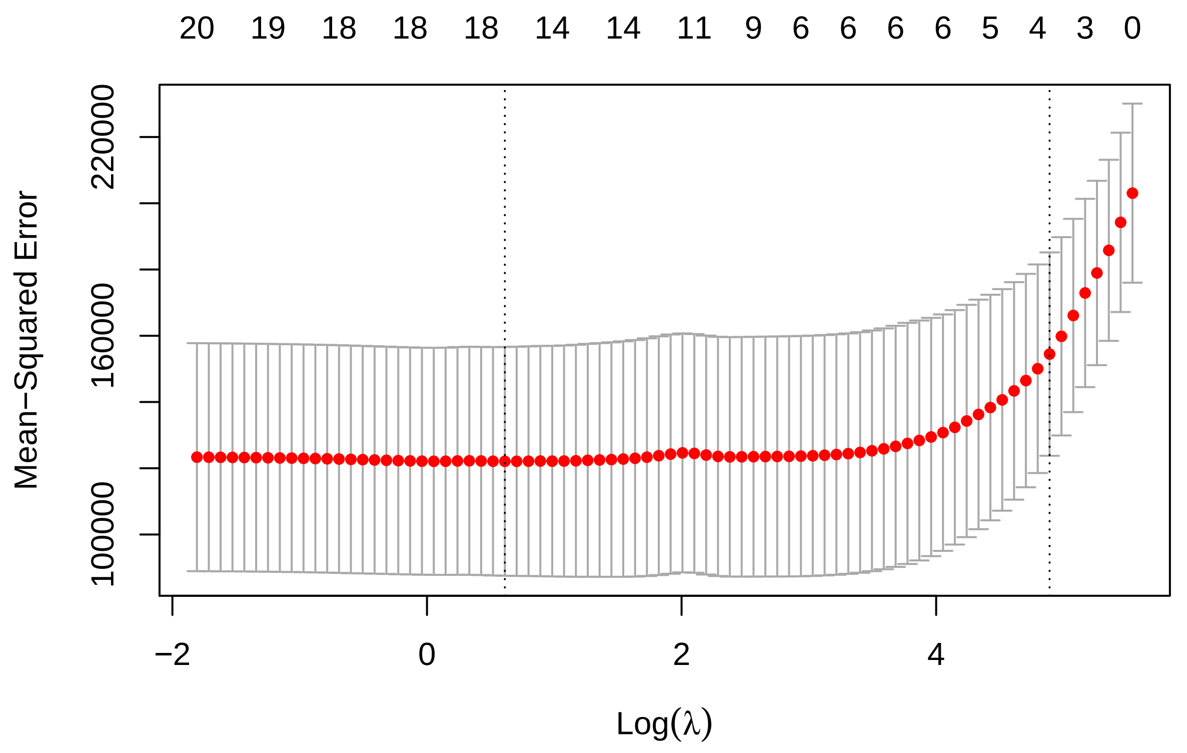
```
plot(fit.lasso, xvar="dev", label=TRUE)
```



deviance -> percentage of deviance explained or in regression R^2.

Lot of R^2 was explained for quite heavily shrunk coefficients towards the end with relatively small increase in R^2 from between 0.4 and 0.5. coefficients grow very large and this may be an indication of that the end of path is overfitting.

```r
cv.lasso=cv.glmnet(x,y)
plot(cv.lasso)
```



this tells us that the minimum cross validation error is for size 7 and within one std. error we have model of size 5.

coeff() function will pick out the coefficients corresponding to the best model, which in this case is:

```r
coef(cv.lasso)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 318.7356100
## AtBat         .
## Hits          0.7092108
## HmRun         .
## Runs          .
## RBI           .
## Walks         0.5516924
## Years         .
## CAtBat        .
## CHits         .
## CHmRun        .
## CRuns         0.0960854
## CRBI          0.2521850
## CWalks        .
## LeagueA       .
## LeagueN       .
## DivisionW     .
## PutOuts       .
## Assists       .
## Errors        .
## NewLeagueN    .
```

*) Find the 1SE value of λ, using 10-fold cross-validation. What is the cross validation estimate for the residual standard error?

```
cv.lasso=cv.glmnet(x,y) #default is 10
names(cv.lasso)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se"
```

```
lambda.1se = cv.lasso$lambda.1se
i0 <- which(cv.lasso$lambda == lambda.1se)
rmse <- cv.lasso$cvm[i0]
rmse
```

```
## [1] 135044.3
```

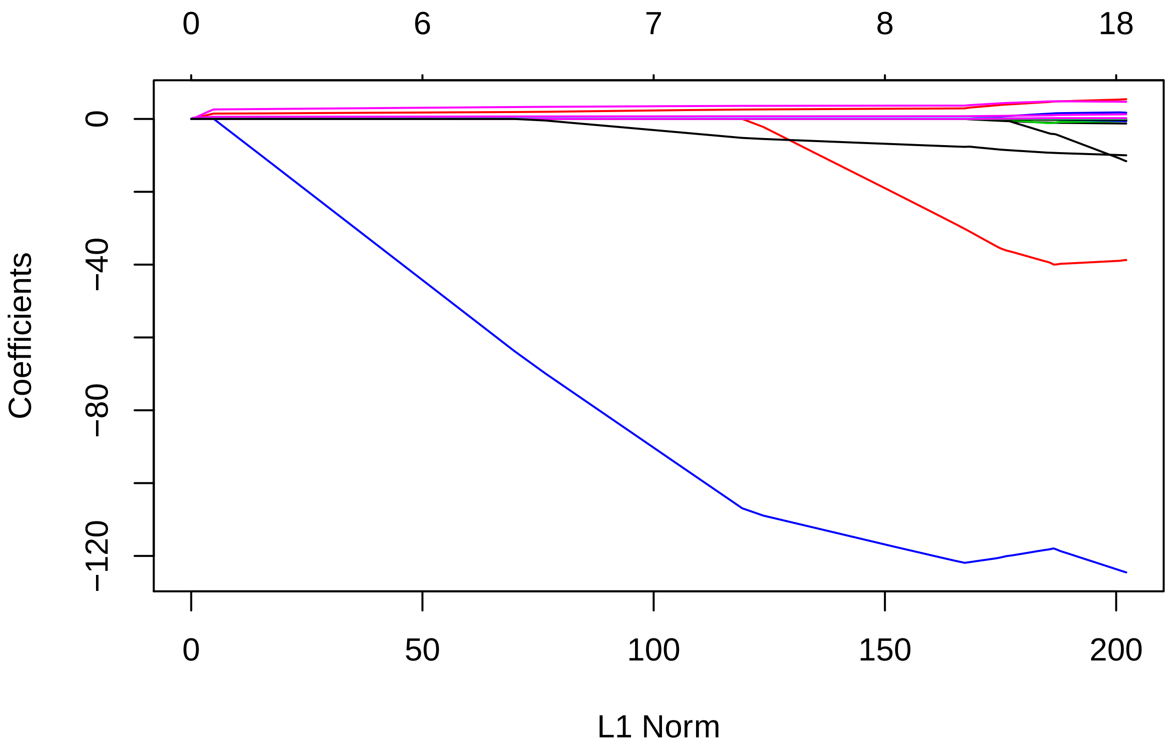
Validation test approach

Suppose we want to use train/validation division to select the 'lambda' for the lasso.

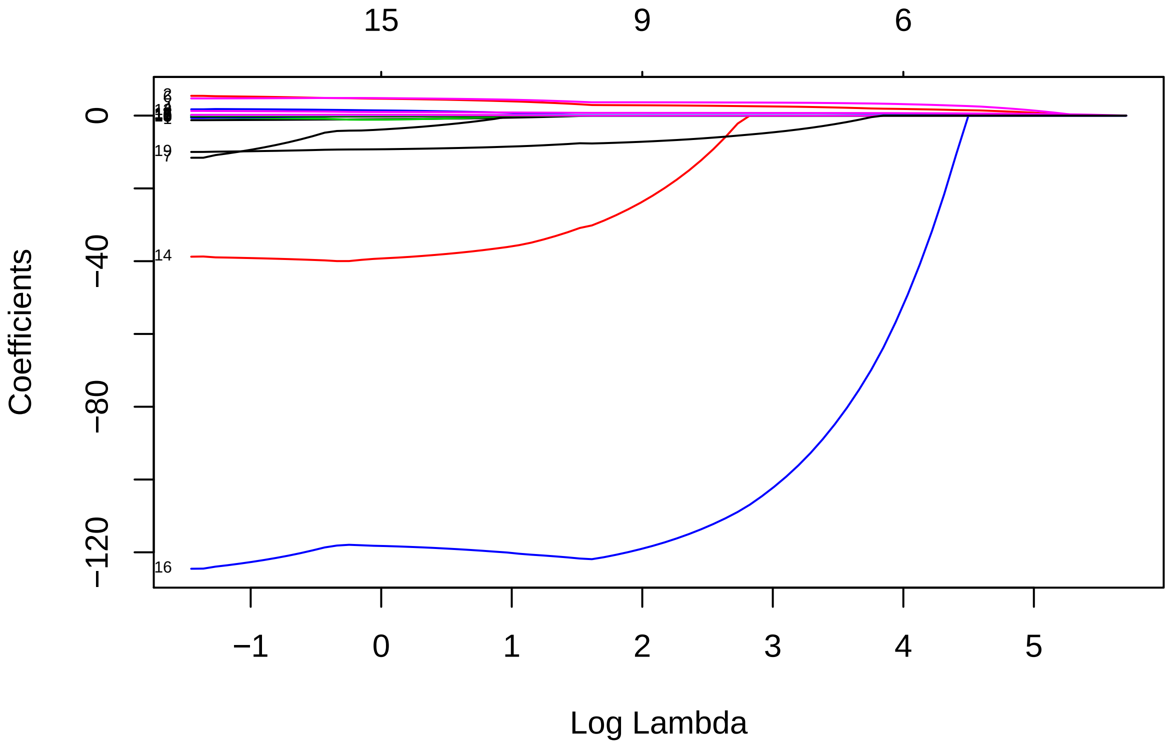
```
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)

lasso.tr=glmnet(x[train,],y[train])
lasso.tr
```

```
##
## Call:  glmnet(x = x[train, ], y = y[train])
##
##    Df  %Dev  Lambda
## 1   0  0.00 301.200
## 2   1  7.71 274.500
## 3   1 14.11 250.100
## 4   1 19.43 227.900
## 5   2 23.92 207.600
## 6   3 30.33 189.200
## 7   3 35.73 172.400
## 8   3 40.22 157.100
## 9   4 43.94 143.100
## 10  4 47.04 130.400
## 11  4 49.61 118.800
## 12  4 51.74 108.300
## 13  4 53.51  98.640
## 14  5 55.17  89.870
## 15  6 56.90  81.890
## 16  6 58.37  74.620
## 17  6 59.60  67.990
## 18  6 60.62  61.950
## 19  6 61.46  56.440
## 20  6 62.16  51.430
## 21  6 62.74  46.860
## 22  7 63.36  42.700
## 23  7 63.98  38.900
## 24  7 64.49  35.450
## 25  7 64.91  32.300
## 26  7 65.27  29.430
## 27  7 65.56  26.820
## 28  6 65.80  24.430
## 29  6 66.00  22.260
## 30  6 66.17  20.280
## 31  6 66.31  18.480
## 32  6 66.42  16.840
## 33  8 66.54  15.340
## 34  8 66.65  13.980
## 35  8 66.75  12.740
## 36  8 66.82  11.610
## 37  8 66.89  10.580
## 38  8 66.94   9.637
## 39  8 66.99   8.781
## 40  9 67.02   8.001
## 41  9 67.06   7.290
## 42  9 67.08   6.642
## 43  9 67.10   6.052
## 44  9 67.12   5.515
## 45 11 67.14   5.025
## 46 12 67.33   4.578
## 47 11 67.54   4.172
## 48 11 67.70   3.801
## 49 11 67.84   3.463
## 50 11 67.95   3.156
## 51 12 68.05   2.875
## 52 13 68.15   2.620
## 53 13 68.22   2.387
## 54 13 68.29   2.175
## 55 13 68.34   1.982
## 56 13 68.39   1.806
## 57 13 68.43   1.645
## 58 13 68.46   1.499
## 59 13 68.48   1.366
## 60 13 68.50   1.245
## 61 14 68.52   1.134
## 62 14 68.54   1.033
## 63 15 68.55   0.942
## 64 15 68.57   0.858
## 65 16 68.58   0.782
## 66 16 68.61   0.712
## 67 17 68.64   0.649
## 68 17 68.70   0.591
## 69 17 68.75   0.539
## 70 17 68.80   0.491
## 71 17 68.83   0.447
## 72 17 68.86   0.408
## 73 17 68.89   0.371
## 74 17 68.91   0.338
## 75 17 68.93   0.308
## 76 18 68.94   0.281
## 77 18 68.98   0.256
## 78 18 68.98   0.233
```

```r
plot(lasso.tr)
```



```r
plot(lasso.tr, xvar="lambda", label=TRUE)
```

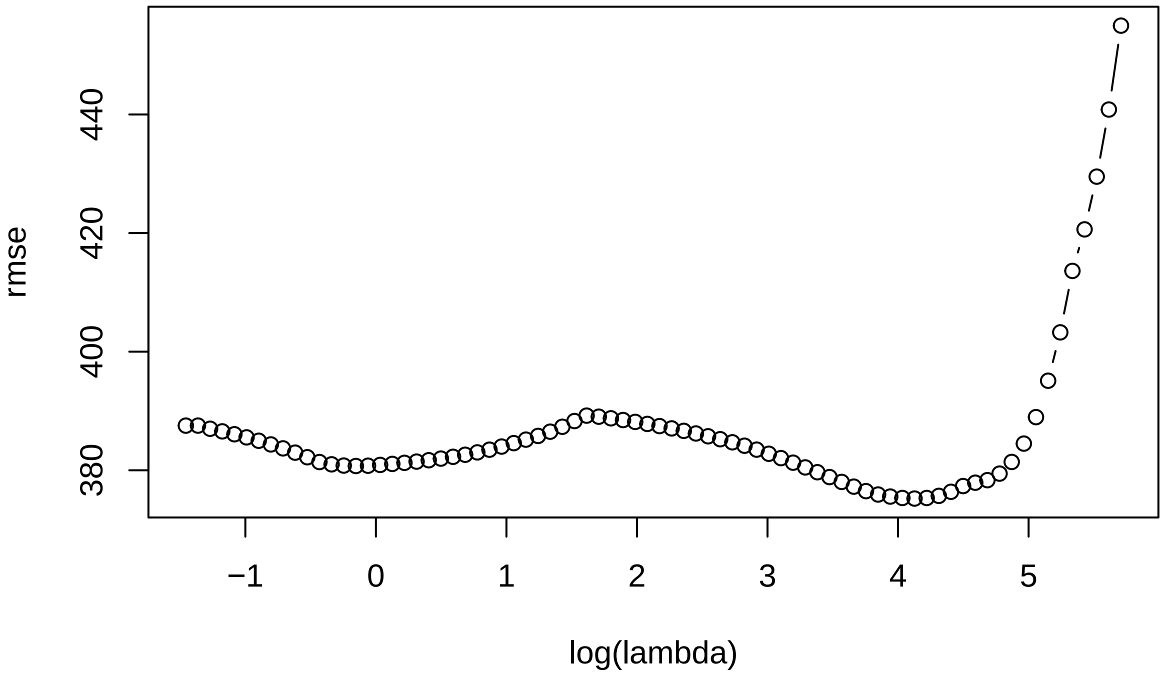


```
pred=predict(lasso.tr, x[-train,])
#dim(pred)
```

we then square those errors and use apply to compute the means in each column of the squared errors.

Then we take the square root. = RMSE (Root MSE)

```
rmse=sqrt(apply((y[-train]-pred)^2,2,mean))
plot(log(lasso.tr$lambda),rmse,type="b",xlab="log(lambda)")
```



```
lam.best=lasso.tr$lambda[order(rmse)[1]]
lam.best
```

```
## [1] 61.94722
```

```
coef(lasso.tr,s=lam.best)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)   3.02089945
## AtBat         .
## Hits          1.74666354
## HmRun         .
## Runs          0.19643837
## RBI           .
## Walks         3.04183261
## Years         .
## CAtBat        .
## CHits         .
## CHmRun        .
## CRuns         .
## CRBI          0.65257702
## CWalks        .
## LeagueA       .
## LeagueN       .
## DivisionW   -40.87078003
## PutOuts       0.06169089
## Assists       .
## Errors        .
## NewLeagueN    .
```

extract the best lambda by indexing:order() put them in ascending order. and we need the index of the smallest value and that will pick up the best lambda.

coefficients: dots indicate the coefficints that are 0. (Sparse matrix format)

Example

http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(glmnet)
```

We'll use the Boston data set [in MASS package], for predicting the median house value (mdev), in Boston Suburbs, based on multiple predictor variables.

```r
# Load the data
data("Boston", package = "MASS")
# Split the data into training and test set
set.seed(123)
training.samples <- Boston$medv %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
```

need to create two objects:

y - for storing the outcome variable x - for holding the predictor variables.

This should be created using the function model.matrix() allowing to automatically transform any qualitative variables (if any) into dummy variables, which is important because glmnet() can only take numerical, quantitative inputs.

After creating the model matrix, we remove the intercept component at index = 1.

```r
# Predictor variables
x <- model.matrix(medv~., train.data)[,-1]
# Outcome variable
y <- train.data$medv
```

We'll use the R function glmnet() [glmnet package] for computing penalized linear regression models.

```r
glmnet(x, y, alpha = 1, lambda = NULL)
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 1, lambda = NULL)
##
##    Df  %Dev Lambda
## 1   0  0.00 6.8040
## 2   1  9.37 6.1990
## 3   1 17.15 5.6480
## 4   2 24.48 5.1470
## 5   2 31.12 4.6890
## 6   2 36.62 4.2730
## 7   2 41.19 3.8930
## 8   2 44.98 3.5470
## 9   2 48.13 3.2320
## 10  3 51.03 2.9450
## 11  3 53.83 2.6830
## 12  3 56.16 2.4450
## 13  3 58.09 2.2280
## 14  3 59.69 2.0300
## 15  3 61.02 1.8500
## 16  3 62.13 1.6850
## 17  3 63.04 1.5360
## 18  3 63.81 1.3990
## 19  4 64.45 1.2750
## 20  4 65.13 1.1620
## 21  4 65.69 1.0580
## 22  4 66.16 0.9644
## 23  5 66.63 0.8787
## 24  5 67.06 0.8007
## 25  5 67.42 0.7295
## 26  5 67.72 0.6647
## 27  5 67.97 0.6057
## 28  6 68.38 0.5519
## 29  6 68.73 0.5028
## 30  6 69.03 0.4582
## 31  7 69.38 0.4175
## 32  8 69.78 0.3804
## 33  8 70.11 0.3466
## 34  8 70.38 0.3158
## 35  8 70.61 0.2877
## 36 10 70.80 0.2622
## 37 10 71.05 0.2389
## 38 11 71.27 0.2177
## 39 11 71.57 0.1983
## 40 11 71.79 0.1807
## 41 11 71.98 0.1647
## 42 12 72.14 0.1500
## 43 12 72.36 0.1367
## 44 12 72.54 0.1246
## 45 12 72.69 0.1135
## 46 12 72.81 0.1034
## 47 12 72.92 0.0942
## 48 12 73.01 0.0858
## 49 12 73.08 0.0782
## 50 12 73.14 0.0713
## 51 12 73.19 0.0649
## 52 13 73.23 0.0592
## 53 13 73.27 0.0539
## 54 13 73.30 0.0491
## 55 13 73.33 0.0448
## 56 13 73.35 0.0408
## 57 13 73.37 0.0372
## 58 13 73.38 0.0339
## 59 13 73.40 0.0308
## 60 13 73.41 0.0281
## 61 13 73.42 0.0256
## 62 13 73.42 0.0233
## 63 13 73.43 0.0213
## 64 13 73.43 0.0194
## 65 13 73.44 0.0177
## 66 13 73.44 0.0161
## 67 13 73.44 0.0147
## 68 13 73.45 0.0134
## 69 13 73.45 0.0122
## 70 13 73.45 0.0111
## 71 13 73.45 0.0101
## 72 13 73.45 0.0092
## 73 13 73.45 0.0084
## 74 13 73.46 0.0076
## 75 13 73.46 0.0070
```

x: matrix of predictor variables y: the response or outcome variable, which is a binary variable. alpha: the elasticnet mixing parameter. Allowed values include:

"1": for lasso regression

"0": for ridge regression

a value between 0 and 1 (say 0.3) for elastic net regression.

lamba: a numeric value defining the amount of shrinkage. Should be specify by analyst.

In penalized regression, you need to specify a constant lambda to adjust the amount of the coefficient shrinkage. The best lambda for your data, can be defined as the lambda that minimize the cross-validation prediction error rate. This can be determined automatically using the function cv.glmnet().

1. Ridge

```r
# Find the best lambda using cross-validation
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 0)
# Display the best lambda value
cv$lambda.min
```

```
## [1] 0.6803611
```

```r
# Fit the final model on the training data
model <- glmnet(x, y, alpha = 0, lambda = cv$lambda.min)
# Display regression coefficients
coef(model)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)  29.238345649
## crim         -0.076748233
## zn            0.026643601
## indus        -0.062378025
## chas          2.528882454
## nox         -11.485535312
## rm            3.728871579
## age           0.001995118
## dis          -1.059940154
## rad           0.166148350
## tax          -0.005328358
## ptratio      -0.863241387
## black         0.009608300
## lstat        -0.497753813
```

```r
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
```

```r
predictions <- model %>% predict(x.test) %>% as.vector()
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##       RMSE   Rsquare
## 1 4.645389 0.7657226
```

2. LASSO

The only difference between the R code used for ridge regression is that, for lasso regression you need to specify the argument alpha = 1 instead of alpha = 0 (for ridge regression).

```r
# Find the best lambda using cross-validation
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 1)
# Display the best lambda value
cv$lambda.min
```

```
## [1] 0.006963707
```

```r
# Fit the final model on the training data
model <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)
# Dsiplay regression coefficients
coef(model)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  37.225146230
## crim         -0.091182130
## zn            0.038065055
## indus        -0.015345392
## chas          2.294708256
## nox         -16.795593125
## rm            3.521945836
## age           0.008557073
## dis          -1.375034888
## rad           0.315378144
## tax          -0.011681132
## ptratio      -0.955435022
## black         0.009782030
## lstat        -0.560017642
```

```r
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
predictions <- model %>% predict(x.test) %>% as.vector()
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##      RMSE   Rsquare
## 1 4.58664 0.7618357
```

Computing elastic net regession

The elastic net regression can be easily computed using the caret workflow, which invokes the glmnet package.

We use caret to automatically select the best tuning parameters alpha and lambda. The caret packages tests a range of possible alpha and lambda values, then selects the best values for lambda and alpha, resulting to a final model that is an elastic net model.

Here, we'll test the combination of 10 different values for alpha and lambda. This is specified using the option tuneLength.

The best alpha and lambda values are those values that minimize the cross-validation error.

```
# Build the model using the training set
set.seed(123)
model <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# Best tuning parameter
model$bestTune
```

```
##   alpha     lambda
## 4   0.1 0.03875385
```

```
# Coefficient of the final model. You need
# to specify the best lambda
coef(model$finalModel, model$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  36.727488405
## crim         -0.090703096
## zn            0.037446370
## indus        -0.020415545
## chas          2.320231739
## nox         -16.457742172
## rm            3.533267286
## age           0.008434395
## dis          -1.358834790
## rad           0.305260173
## tax          -0.011175268
## ptratio      -0.950290918
## black         0.009799131
## lstat        -0.557178910
```

```
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
predictions <- model %>% predict(x.test)
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##       RMSE   Rsquare
## 1 4.587382 0.7621341
```

Comparing the different models:

The different models performance metrics are comparable. Using lasso or elastic net regression set the coefficient of the predictor variable age to zero, leading to a simpler model compared to the ridge regression, which include all predictor variables.

All things equal, we should go for the simpler model. In our example, we can choose the lasso or the elastic net regression models.

Note that, we can easily compute and compare ridge, lasso and elastic net regression using the caret workflow.

caret will automatically choose the best tuning parameter values, compute the final model and evaluate the model performance using cross-validation techniques.

Using caret package

a. Setup a grid range of lambda values:

```
lambda <- 10^seq(-3, 3, length = 100)
```

1. Compute ridge regression:

```
# Build the model
set.seed(123)
ridge <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 0, lambda = lambda)
  )
# Model coefficients
coef(ridge$finalModel, ridge$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  29.199522433
## crim         -0.076456377
## zn            0.026794700
## indus        -0.062277495
## chas          2.525893584
## nox         -11.388819192
## rm            3.723870172
## age           0.002204446
## dis          -1.058455239
## rad           0.165876918
## tax          -0.005354632
## ptratio      -0.862205203
## black         0.009610093
## lstat        -0.498825572
```

```
# Make predictions
predictions <- ridge %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##       RMSE   Rsquare
## 1 4.646655 0.7655889
```

2. Compute lasso regression:

```
# Build the model
set.seed(123)
lasso <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda)
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
# Model coefficients
coef(lasso$finalModel, lasso$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  37.199465655
## crim         -0.091088918
## zn            0.038010919
## indus        -0.014083319
## chas          2.291379000
## nox         -16.745915299
## rm            3.520653532
## age           0.008671758
## dis          -1.370592996
## rad           0.316149936
## tax          -0.011743014
## ptratio      -0.955953859
## black         0.009790457
## lstat        -0.560615758
```

```
# Make predictions
predictions <- lasso %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##       RMSE  Rsquare
## 1 4.587305 0.761773
```

3. Elastic net regression:

```
# Build the model
set.seed(123)
elastic <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
# Model coefficients
coef(elastic$finalModel, elastic$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  36.727488405
## crim         -0.090703096
## zn            0.037446370
## indus        -0.020415545
## chas          2.320231739
## nox         -16.457742172
## rm            3.533267286
## age           0.008434395
## dis          -1.358834790
## rad           0.305260173
## tax          -0.011175268
## ptratio      -0.950290918
## black         0.009799131
## lstat        -0.557178910
```

```r
# Make predictions
predictions <- elastic %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##       RMSE   Rsquare
## 1 4.587382 0.7621341
```

4. Comparing models performance:

The performance of the different models - ridge, lasso and elastic net - can be easily compared using caret. The best model is defined as the one that minimizes the prediction error.

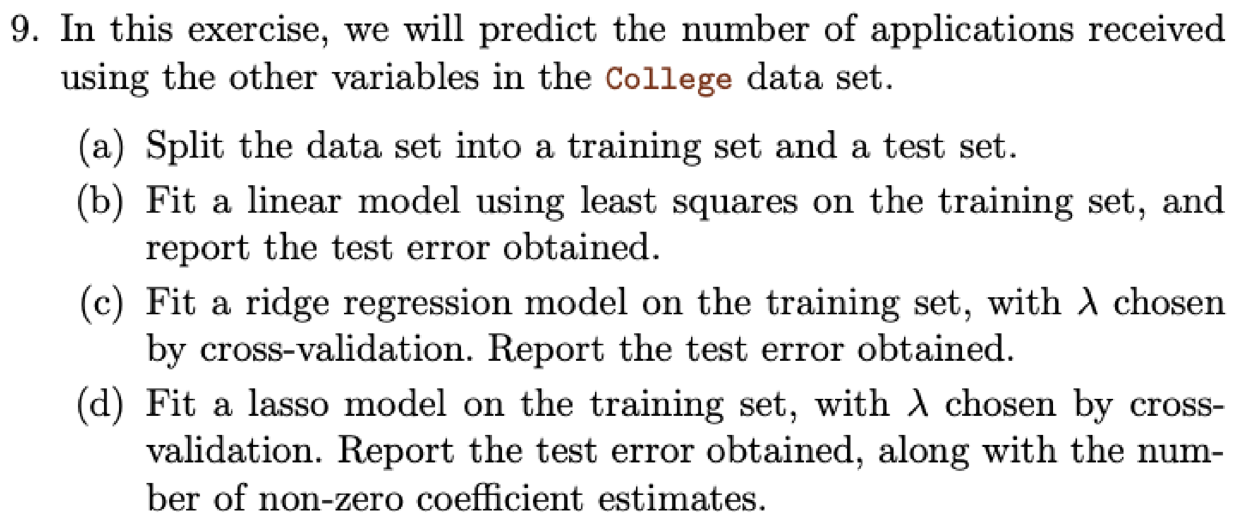
```r
models <- list(ridge = ridge, lasso = lasso, elastic = elastic)
resamples(models) %>% summary( metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = ., metric = "RMSE")
##
## Models: ridge, lasso, elastic
## Number of resamples: 10
##
## RMSE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ridge   3.497664 4.194848 4.716501 4.846001 5.460015 6.389243    0
## lasso   3.639341 4.156179 4.693779 4.817913 5.406915 6.161518    0
## elastic 3.630082 4.148174 4.692444 4.816828 5.404555 6.163305    0
```

It can be seen that the elastic net model has the lowest median RMSE.

Lab Assignment

Question 1: ISLR ch. 6 #9abcd



9. In this exercise, we will predict the number of applications received using the other variables in the `College` data set.

   (a) Split the data set into a training set and a test set.

   (b) Fit a linear model using least squares on the training set, and report the test error obtained.

   (c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

   (d) Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

Question 2:

Consider the **Boston** data. We want to predict **medv** from all other predictors, using the LASSO.

a) Set up the LASSO and plot the trajectories of all coefficients. What are the last five variables to remain in the model?

b) Find the 1SE value of λ, using 10-fold cross-validation. What is the cross validation estimate for the residual standard error?

c) Rescale all predictors so that their mean is zero and their standard deviation is 1. Then set up the LASSO and plot the trajectories of all coefficients. What are the last five variables to remain in the model? Compare your answer to part a).

d) Find the 1SE value of λ, using 10-fold cross-validation. What is the cross validation estimate for the residual standard error now? Does rescaling lead to a better performing model?