ISYE 6501 - Homework 8

Justin Lewis

February 28, 2019

Start by clearing the environment and importing the relevant packages. Still struggling to prevent these annoying outputs.

```
rm(list=ls())
library(kernlab, quietly=TRUE)
```

```
## Warning: package 'kernlab' was built under R version 3.5.2
```

```
library(knitr, quietly=TRUE)
```

```
## Warning: package 'knitr' was built under R version 3.5.2
```

```
library(ggplot2, quietly=TRUE)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
library(glmnet, quietly=TRUE)
```

```
## Warning: package 'glmnet' was built under R version 3.5.2
```

```
## Warning: package 'Matrix' was built under R version 3.5.2
```

```
## Warning: package 'foreach' was built under R version 3.5.2
```

```
## Loaded glmnet 2.0-16
```

Question 11.1

Q:

Using the crime data set uscrime.txt from Questions 8.2, 9.1, and 10.1, build a regression model using: 1. Stepwise regression 2. Lasso 3. Elastic net For Parts 2 and 3, remember to scale the data first - otherwise, the regression coefficients will be on different scales and the constraint won't have the desired effect.

A:

Start by importing the data to be used and splitting it up into train/test sets

```
uscrime <- as.data.frame(read.csv('uscrime.txt', header=TRUE, sep='\t'))

#Split data up into train/test
train_size <- floor(0.75*nrow(uscrime))
set.seed(42)
train_indices <- sample(seq_len(nrow(uscrime)), size=train_size)
train <- uscrime[train_indices, ]
test <- uscrime[-train_indices, ]
```

Now going to begin by using stepwise regression. I am going to start with every factor, and then iterate through the predictors which give a high p-value.

```
#1. Stepwise Regression

#First need to fit the model using all the variables
lmall <- lm(train$Crime ~., train[1:15])

#Now check the summary of the model
lmallsummary <- summary(lmall)
print(lmallsummary)
```

```
##
## Call:
## lm(formula = train$Crime ~ ., data = train[1:15])
##
## Residuals:

##     Min      1Q  Median      3Q     Max
## -303.81  -63.54   -7.89   69.83  262.07
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.096e+03  1.768e+03  -3.449  0.00269 **
## M            1.297e+02  5.541e+01   2.341  0.03032 *
## So          -7.567e+01  1.790e+02  -0.423  0.67728
## Ed           1.982e+02  7.233e+01   2.740  0.01301 *
## Po1          2.216e+02  1.146e+02   1.933  0.06828 .
## Po2         -1.388e+02  1.287e+02  -1.079  0.29415
## LF          -5.134e+02  1.437e+03  -0.357  0.72478
## M.F          2.735e+00  2.265e+01   0.121  0.90517
## Pop          2.474e-03  1.403e+00   0.002  0.99861
## NW           3.260e+00  8.623e+00   0.378  0.70957
## U1          -8.382e+02  5.738e+03  -0.146  0.88540
## U2           8.001e+01  1.232e+02   0.650  0.52367
## Wealth       1.596e-01  1.491e-01   1.071  0.29771
## Ineq         9.194e+01  2.414e+01   3.808  0.00119 **
## Prob        -5.579e+03  3.622e+03  -1.541  0.13990
## Time        -9.187e+00  9.367e+00  -0.981  0.33902
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 188.9 on 19 degrees of freedom
## Multiple R-squared:  0.868,  Adjusted R-squared:  0.7639
## F-statistic: 8.332 on 15 and 19 DF,  p-value: 1.91e-05
```

From the summary of the model with every variable, we see there are several variables with p-values over the threshold of 0.05. Now the iteration over all of the variables is repetitive, so I am going to define a function to do it for me. The function takes in a summary, a set of attributes, and a response variable. If there are any p-values over 0.05 in the coefficients of the summary, they are ordered and the highest is removed. Then the new set of attributes are fit to the response for a new model, and back elim is run again until there are no p-values over 0.05.

This is the opposite method to what was seen in the lectures, where we began with no factors and added them in individually. Instead I am starting with ALL the factors, and removing them one-by-one.

```
back_elim <- function(summ, atts, response){
  #Take in a summary, get the coefficients, sort the p-values, and remove the highest

  #Create a new df with the coefficients table in the provided summary
  ms_df <- as.data.frame(summ['coefficients'])[2:nrow(as.data.frame(summ['coefficients'])),]

  #Order the df based on the values in the p-values column
  ms_df <- ms_df[order(ms_df$coefficients.Pr...t..),]

  #Reverse the result so the highest are at the top. Could not make descending option work?
  ms_df <- ms_df[dim(ms_df)[1]:1,]

  #Get the subset with p-values over 0.05
  high_pvs <- ms_df[ms_df['coefficients.Pr...t..']>0.05,]['coefficients.Pr...t..']

  #If there are any p-values over 0.05, remove the highest, then fit a new model with the remain
ing attributes
  if (nrow(high_pvs)>0){
    print(paste('Removing:', row.names(high_pvs)[[1]], 'from attributes.'))
    atts <- atts[, !(names(atts) %in% c(row.names(high_pvs)[[1]]))]
    new_m <- lm(response ~., atts)
    new_s <- summary(new_m)
    back_elim(new_s, atts, response)
  }
  #Otherwise, return all the good attributes to use
  else if (nrow(high_pvs) == 0){
    print(paste('Remaining attributes:', list(colnames(atts))))
    return(atts)
  }
}
```

Now lets try it out with the summary and attributes from the first model with all of the predictors used.

```
#Store the results of the backwards elimination
stepwiseatts <- back_elim(summ=lmallsummary, atts=train[1:15], response=train$Crime)
```

```
## [1] "Removing: Pop from attributes."
## [1] "Removing: M.F from attributes."
## [1] "Removing: U1 from attributes."
## [1] "Removing: LF from attributes."
## [1] "Removing: NW from attributes."
## [1] "Removing: So from attributes."
## [1] "Removing: Wealth from attributes."
## [1] "Removing: Time from attributes."
## [1] "Removing: Po2 from attributes."
## [1] "Removing: U2 from attributes."
## [1] "Remaining attributes: c(\"M\", \"Ed\", \"Po1\", \"Ineq\", \"Prob\")"
```

```
#Fit a new model with the remaining attributes
lmswreg <- lm(train$Crime ~., stepwiseatts)
summary(lmswreg)
```

```
##
## Call:
## lm(formula = train$Crime ~ ., data = stepwiseatts)
##
## Residuals:

##     Min      1Q  Median      3Q     Max
## -457.29  -61.61   15.72   93.41  268.68
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4815.66     743.71  -6.475 4.36e-07 ***
## M              88.31      30.27   2.917 0.006755 **
## Ed            186.03      41.81   4.449 0.000117 ***
## Po1           133.03      15.06   8.832 1.02e-09 ***
## Ineq           82.13      14.24   5.769 3.01e-06 ***
## Prob        -4950.51    2122.23  -2.333 0.026810 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 171.9 on 29 degrees of freedom
## Multiple R-squared:  0.8333, Adjusted R-squared:  0.8046
## F-statistic:    29 on 5 and 29 DF,  p-value: 1.902e-10
```

Through the stepwise regression method the number of attributes/factors in our model was reduced to 5 from 15 and while the multiple R-squared dropped a little bit, the resulting adjusted R-squared actually increased. Due to the increased simplicity and small amount of change in metrics, I would consider this a much better model, but it should be tested on the test set as well.

```
#Make the predictions
singlepredict <- predict.lm(lmswreg, test[1:15])

#Calculate the R-squared
sst <- sum((test$Crime-mean(test$Crime))^2)
sse <- sum((test$Crime-singlepredict)^2)
r2 <- 1 - sse/sst
print(paste('R-squared value on test data:' , r2))
```

```
## [1] "R-squared value on test data: 0.163372781104892"
```

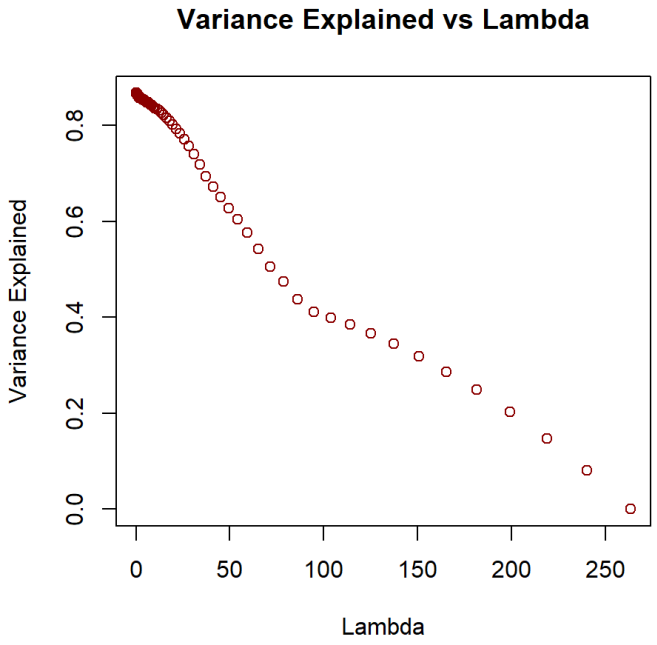
We did not perform nearly as well on the test data. This is likely due to overfitting on the training set since our data set is so small.

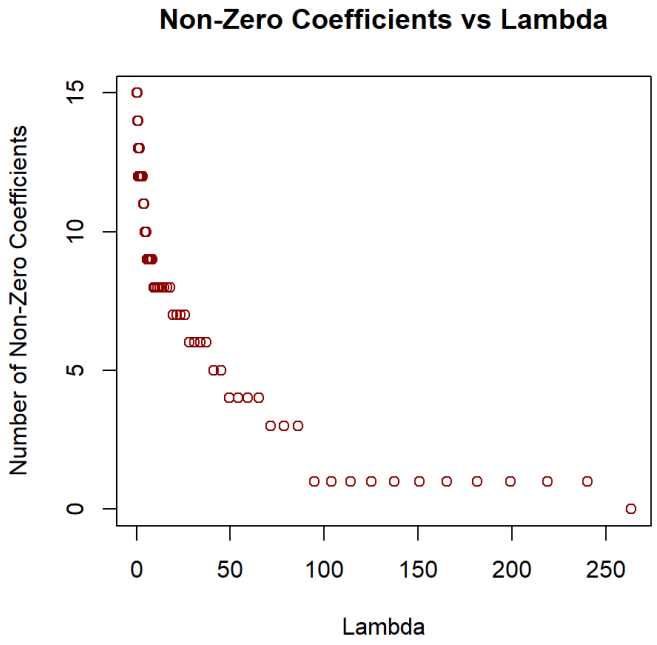
Before we move on to LASSO and Elastic Net, we need to first create a scaled version of our data.

```
#Need to create a scaled version of the data, but not the target column
uscrimesc <- data.frame(cbind(scale(uscrime[1:15]), uscrime$Crime))

#And make new training and test sets
trainsc <- uscrimesc[train_indices, ]
testsc <- uscrimesc[-train_indices, ]
```

With the data scaled and split, we can begin using LASSO to select variables. Here we use an alpha of 1 which corresponds to a LASSO method as we saw in the lectures. In the later portion we can adjust alpha to various values between 0 and 1, which is all effectively different Elastic Net implementations.

```
#Perform the method with alpha=1 (LASSO)
lasso <- glmnet(x=as.matrix(trainsc[1:15]), y=trainsc$V16, family='mgaussian', alpha=1)

#Check the variance explained vs the lambda parameter
par(mfrow=c(1,2))
plot(lasso$lambda, lasso$dev.ratio, xlab='Lambda', ylab='Variance Explained', main='Variance Exp
lained vs Lambda', col='darkred')

#And the number of non-zero coefficients vs the lambda parameter
plot(lasso$lambda, lasso$df, xlab='Lambda', ylab='Number of Non-Zero Coefficients', main='Non-Ze
ro Coefficients vs Lambda', col='darkred')
```





Here we see the effect of the lambda parameter. With no penalty (lambda=0) we have nearly all the variance in our data explained and all 15 predictors have non-zero coefficients. As the penalty multiplier increases, we have fewer non-zero coefficients and as a result we get less variance explained.

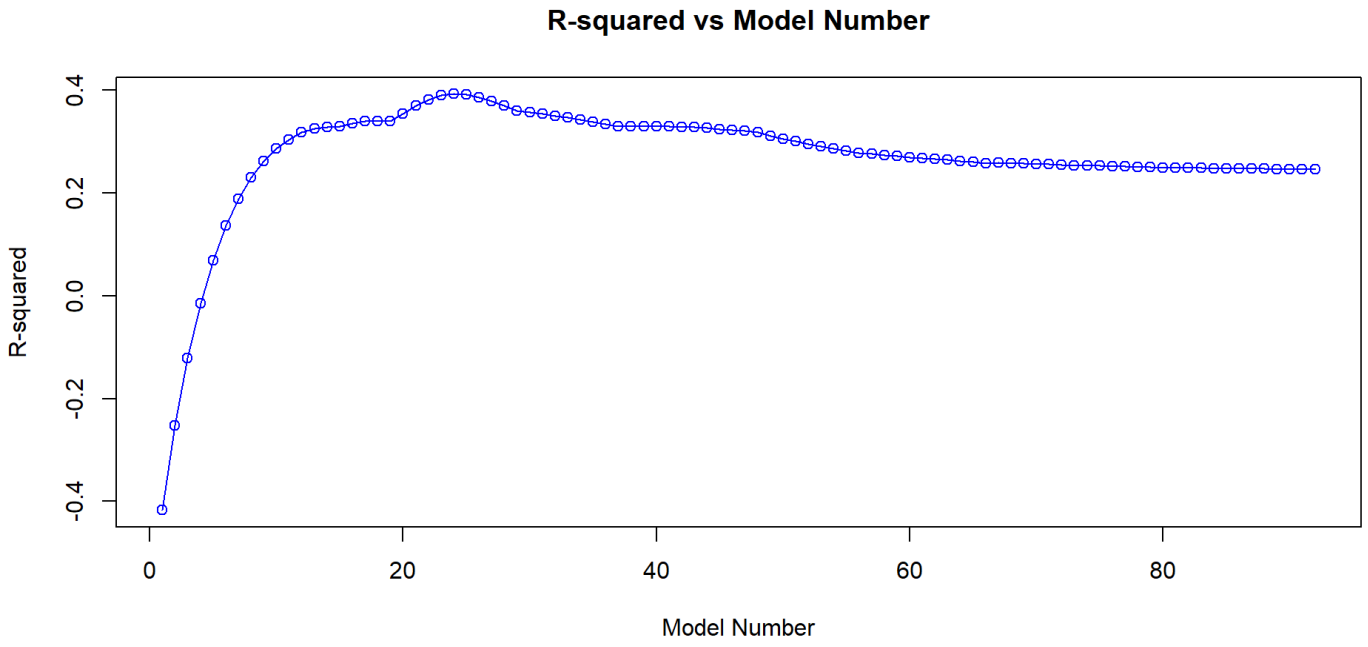
```
#This will create predictions using every single option from above.
lasso_preds <- predict.glmnet(lasso, newx=as.matrix(testsc[1:15]))
r2s <- list()
#Now make a list of the R-squared value on the test set for each model
for (i in 1:ncol(lasso_preds)){
  #Calculate the R-squared
  sst <- sum((testsc$V16-mean(testsc$V16))^2)
  sse <- sum((testsc$V16-lasso_preds[, i])^2)
  r2 <- 1 - sse/sst
  r2s[[i]] <- r2
}
#So the best model is
print(which.max(r2s))
```

```
## [1] 24
```

```
#With an R-squared of
r2s[which.max(r2s)]
```

```
## [[1]]
## [1] 0.392073
```

```
#Visualizing how each model did
plot(1:ncol(lasso_preds), r2s, xlab='Model Number', ylab='R-squared', main='R-squared vs Model N
umber', type='o', col='blue')
```



We know which model performed best on the test data, but we have not checked the actual coefficients. To do this we go back to the LASSO portion and can grab the coefficient matrix corresponding the the best model.

```
#Store the index of the best model
best <- which.max(r2s)
#Print the coefficients
lasso$beta[, best]
```

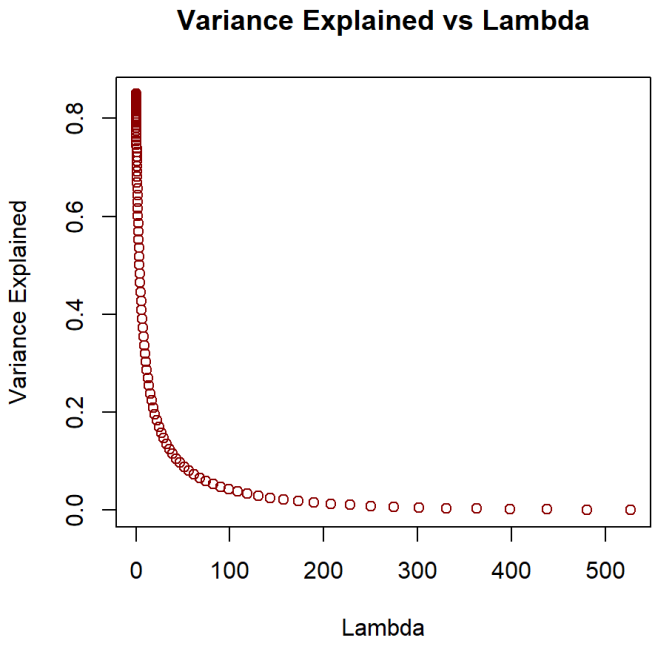
```
##         M        So        Ed       Po1       Po2        LF       M.F
##  66.43808   0.00000  34.10584 333.33707   0.00000   0.00000  45.83498
##       Pop        NW        U1        U2    Wealth      Ineq      Prob
##   0.00000   0.00000   0.00000   0.00000   0.00000 130.17852 -41.69158
##      Time
##   0.00000
```

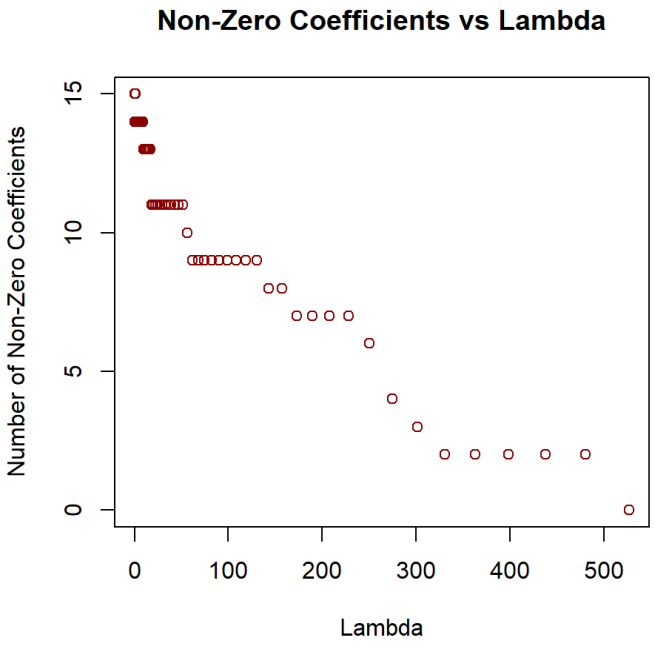
So our best model results in one extra factor compared to the step-wise method. However we got a much better metric value, with the higher R-squared.

One thing to note, is that this method has a flaw. By using the test data to determine which model to use from the LASSO method, we actually used it as a validation set. To truly know the effectiveness of the model we would want to use a third, non-used set for a true test. As it is we fit the data to the train set, then used the test set to choose the coefficients/model. But with so little data I am going to leave it as it is.

Now by adjusting alpha, we can build a model using ElasticNet.

```
#Try starting with a conservative value of 0.5, and repeat similar method as above
elastic <- glmnet(x=as.matrix(trainsc[1:15]), y=trainsc$V16, family='mgaussian', alpha=0.5)

#Check the variance explained vs the lambda parameter
par(mfrow=c(1,2))
plot(elastic$lambda, elastic$dev.ratio, xlab='Lambda', ylab='Variance Explained', main='Variance
Explained vs Lambda', col='darkred')

#And the number of non-zero coefficients vs the lambda parameter
plot(elastic$lambda, elastic$df, xlab='Lambda', ylab='Number of Non-Zero Coefficients', main='No
n-Zero Coefficients vs Lambda', col='darkred')
```





We see a significantly different result with the adjusted alpha value. Our variance drops off much faster, and the number of non-zero coefficients remains high much longer.

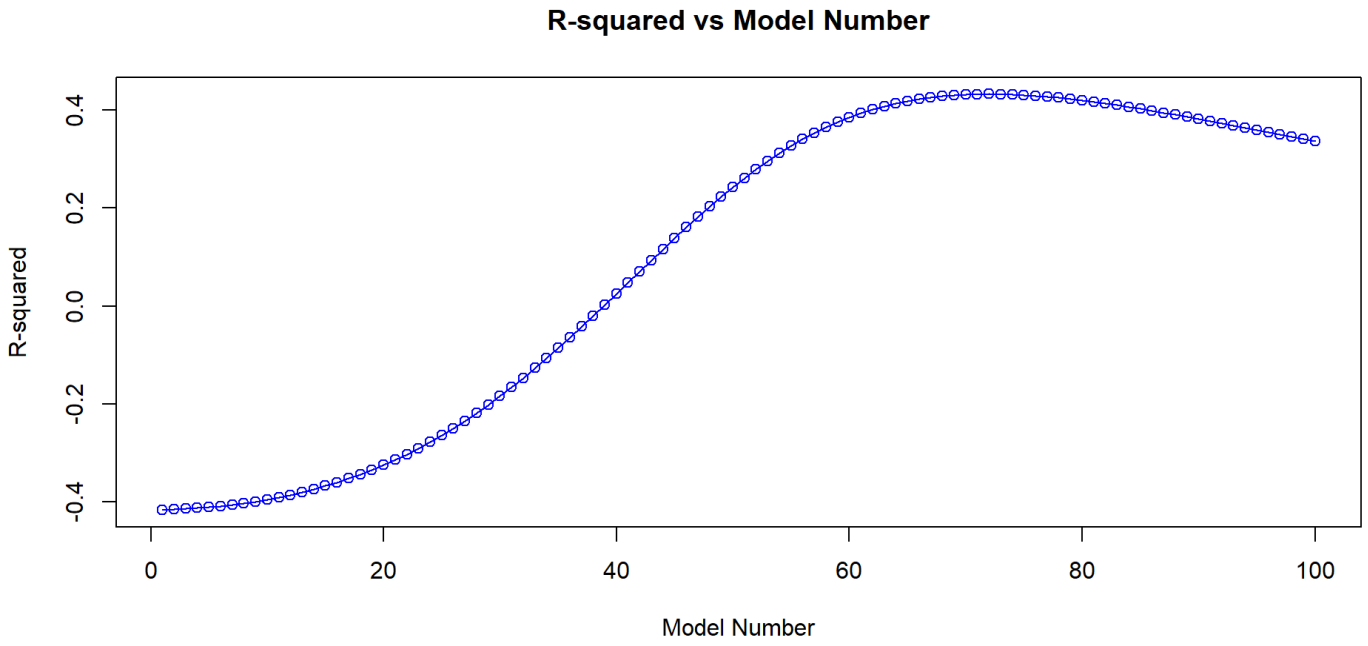
```
#This will create predictions using every single option from above.
elastic_preds <- predict.glmnet(elastic, newx=as.matrix(testsc[1:15]))
r2s <- list()
#Now make a list of the R-squared value on the test set for each model
for (i in 1:ncol(elastic_preds)){
  #Calculate the R-squared
  sst <- sum((testsc$V16-mean(testsc$V16))^2)
  sse <- sum((testsc$V16-elastic_preds[, i])^2)
  r2 <- 1 - sse/sst
  r2s[[i]] <- r2
}
#So the best model is
print(which.max(r2s))
```

```
## [1] 72
```

```
#With an R-squared of
r2s[which.max(r2s)]
```

```
## [[1]]
## [1] 0.4329387
```

```
#Visualizing how each model did
plot(1:ncol(elastic_preds), r2s, xlab='Model Number', ylab='R-squared', main='R-squared vs Model
Number', type='o', col='blue')
```



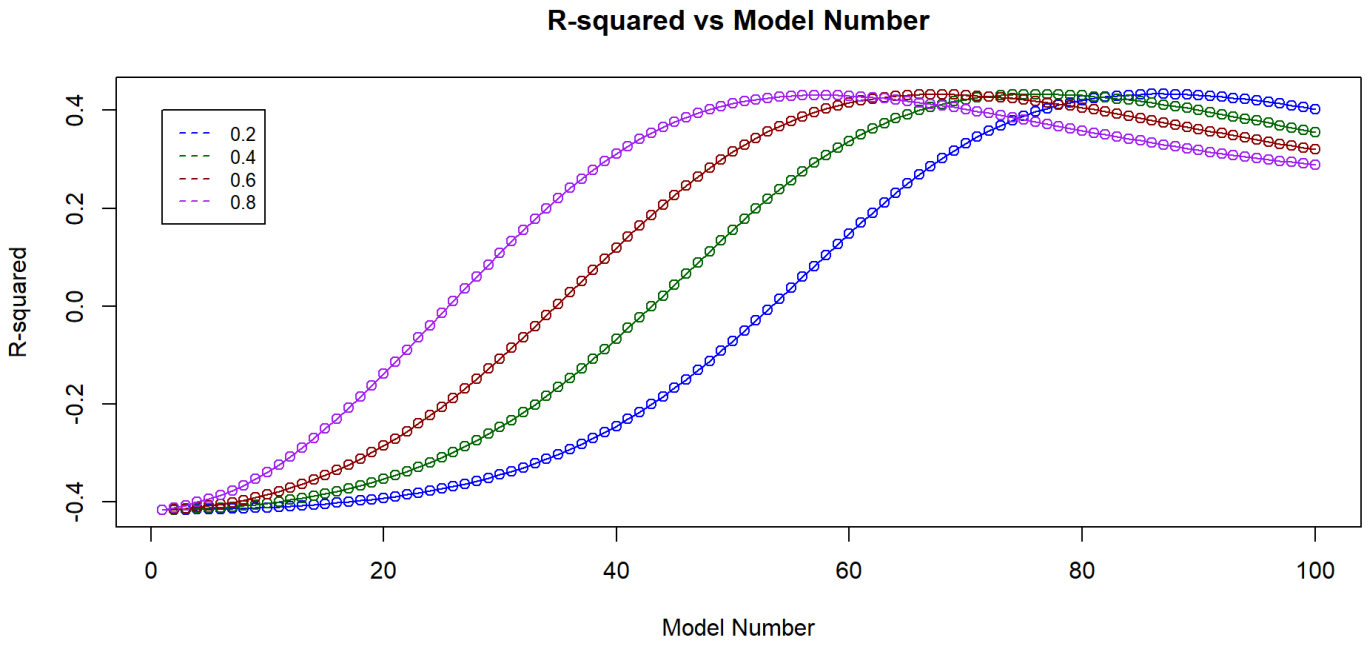
With the elastic net method we see a much smoother increase in the calculated R-squared value, with the peak much later on in the method. Now similar to above, we should also check what the coefficients of our model are.

```
#Store the index of the best model
best <- which.max(r2s)
#Print the coefficients
elastic$beta[, best]
```

```
##          M         So         Ed        Po1        Po2         LF
##  66.877973  30.908476  58.457066 128.033449 107.170696  28.282835
##        M.F        Pop         NW         U1         U2     Wealth
##  66.979256  40.635312  34.203197  -9.112282  34.371270  27.401378
##       Ineq       Prob       Time
##  95.686913 -69.098477  -1.562534
```

With this method we did not actually reduce any of the coefficients to zero, but our range of coefficient values is much smaller than with the LASSO method. To understand the effects of different values of alpha, we can iterate through multiple values.

```
#Define several alpha values, not including zero because ridge regression gives quite different
 results
alphas <- c(0.2, 0.4, 0.6, 0.8)

#Iterate through and store each model using 'assign'
for (a in alphas){
  assign(paste('fit', a, sep='_'), glmnet(x=as.matrix(trainsc[1:15]), y=as.matrix(trainsc$V16),
 alpha=a, family='mgaussian'))
}

#Now recreate the analysis for our original model, but for all of the above models
mods <- list(fit_0.2, fit_0.4, fit_0.6, fit_0.8)
ind <- 1
r2list<- list()
bests <- list()
#This will create predictions using every single option from above.
for (m in mods){
  preds <- predict.glmnet(m, newx=as.matrix(testsc[1:15]))
  r2s <- list()
  #Now make a list of the R-squared value on the test set for each model
  for (i in 1:ncol(preds)){
    #Calculate the R-squared
    sst <- sum((testsc[['V16']]-mean(testsc[['V16']]))^2)
    sse <- sum((testsc[['V16']]-preds[, i])^2)
    r2 <- 1 - sse/sst
    r2s[[i]] <- r2
  }
  r2list[[ind]] <- r2s
  bests[[ind]] <- which.max(r2s)
  ind = ind + 1
}

#Visualizing how each model did
plot(1:ncol(preds), r2list[[1]], xlab='Model Number', ylab='R-squared', main='R-squared vs Model
Number', type='o', col='blue')
lines(1:ncol(preds), r2list[[2]], type='o', col='darkgreen')
lines(1:ncol(preds), r2list[[3]], type='o', col='darkred')
lines(1:ncol(preds), r2list[[4]], type='o', col='purple')
legend(1, 0.4, legend=c('0.2', '0.4', '0.6', '0.8'), col=c('blue', 'darkgreen', 'darkred', 'purp
le'), lty=8, cex=0.8)
```



As we increase the alpha value, the resulting maximum R-squared values are reached earlier and earlier. Now if we look into the coefficients for each of these best performing models we can see how much of an affect the alpha parameter has.

```
#Go through each model and print the best performing coefficients
for (i in seq_along(mods)){
  print(paste('For alpha of', alphas[i], 'optimal coefficients are:'))
  print(mods[[i]]$beta[, bests[[i]]])
}
```

```
## [1] "For alpha of 0.2 optimal coefficients are:"
##          M         So         Ed        Po1        Po2         LF
##  67.358035  31.359775  59.157990 128.038758 107.325844  28.493192
##        M.F        Pop         NW         U1         U2     Wealth
##  67.216858  40.818690  34.445321  -9.556652  35.038874  28.111135
##       Ineq       Prob       Time
##  96.820699 -69.762577  -2.137282
## [1] "For alpha of 0.4 optimal coefficients are:"
##          M         So         Ed        Po1        Po2         LF
##  66.180578  31.011445  57.501942 126.805735 106.497558  28.342737
##        M.F        Pop         NW         U1         U2     Wealth
##  66.684002  40.717074  34.375988  -9.132620  34.263416  27.508218
##       Ineq       Prob       Time
##  93.647769 -68.602803  -1.690454
## [1] "For alpha of 0.6 optimal coefficients are:"
##          M         So         Ed        Po1        Po2         LF
##  67.529932  30.809328  59.336206 129.264730 107.831138  28.200564
##        M.F        Pop         NW         U1         U2     Wealth
##  67.213035  40.488966  33.961886  -8.997644  34.363027  27.236980
##       Ineq       Prob       Time
##  97.623818 -69.523431  -1.347253
## [1] "For alpha of 0.8 optimal coefficients are:"
##          M         So         Ed        Po1        Po2         LF
##  65.756008  29.638377  56.392613 128.141438 107.301619  27.599287
##        M.F        Pop         NW         U1         U2     Wealth
##  66.131464  39.887971  33.278223  -7.213396  31.881103  25.660349
##       Ineq       Prob       Time
##  93.164794 -67.242182   0.000000
```

We see with the increasing alpha parameter, the coefficients are shrinking slowly, with only an alpha of 0.8 resulting in any of the parameter coefficients shrinking to 0. With lower values like 0.2 we are introducing more bias into the model in order to reduce variances in the estimate, while with higher values like 0.8 we are accepting higher variance in order to remain less biased.