STA 478 Fall 2025 Assignment #7

Dr. Robert Buscaglia

October 31, 2025

Due Date: Tuesday, Nobember 9th, 2021 before 8:00 AM.

Exam 2 Tentative Date: Weekend of November 12th, 2021 - Due Monday, 11/15/2021, before 5:00 PM

Exercises

Exercise 1

Construct a function that will calculate the Variance Inflation Factors of all predictors. You may choose how the data is read in, but I recommend that the function take a matrix or data.frame of which each predictor is a unique column. The VIF function should return the VIF score for each predictor. Recall that VIF is calculated as:

$$VIF = \frac{1}{1 - R_j^2}$$

where R_j^2 is the Coefficient of Determination for regression of predictor x_j against all other predictors.

Solution.

The goal here is to prepare a function that accepts only numerical predictors from a data set and produces a VIF output that can be used to selected out variables from the analysis. There are many ways this can be done, you can see my approach below. The difficulty of the function typically lies in producing the proper model fits. I use matrix forms to get this to work properly.

```
# VIF.single
#
# A function that takes a set of numerical predictors and determines there
# variance inflation factors. Can then be used to eliminate high VIF columns.
VIF.single <- function(predictor.frame)
{
  VIF.out<-numeric()
  n <- ncol(predictor.frame)
  for(j in 1:n)
  {
    work.frame <- data.frame(y = predictor.frame[,j], x=predictor.frame[,-j])
    R2.temp<-summary(lm(y ~ ., data=work.frame))$r.squared
    VIF.out[j] <- 1/(1-R2.temp)
  }
  return(data.frame(Predictor = colnames(predictor.frame), VIF=VIF.out) %>% arrange(desc(VIF)
}
```

You could then produce a "wrapper" function that could do the iterative removal of predictors based on a given VIF threshold. I will choose to do this by hand below so that I can review each VIF step.

Exercise 2

a

Return to the `Boston` data analyzed in a previous homework. Using your VIF function, calculate the VIF for each predictor in the `Boston` set (all variables but `crim`). Display a table of the VIF for each predictor.

Solution.

We have explored this data previously, so I omit any EDA and move directly to the use of VIF. We are asked to essentially show that our VIF function is working well. Here is my output for the `Boston` data set. We remove the predictor `crim` (as we will use this to produce there response variable `crim01`) and apply VIF!

```
data('Boston', package='MASS')
Boston.ws <- Boston %>% select(-crim)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
tax	9.195495
rad	7.158834
nox	4.551563
dis	4.289040
indus	3.987753
medv	3.772856
lstat	3.561476
age	3.100802
zn	2.325094
rm	2.258113
ptratio	1.984490
black	1.369741
chas	1.094326

b

Iteratively remove the predictor with the highest VIF until no predictors have a VIF greater than 5. Call this predictor set `VIF.5`.

Solution.

The goal here is to apply VIF in a sequential method removing the highest VIF predictor until we have no predictors above 5. We start with the output above indicating we should remove `tax` first. I remove `tax` from the predictor set and run VIF again. I use `Boston.ws` to represent my working set.

```
Boston.ws <- Boston.ws %>% select(-tax)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
nox	4.542222
dis	4.286740
medv	3.695636
lstat	3.539632
indus	3.226111
age	3.098035
rad	2.361408
rm	2.257707
zn	2.193080
ptratio	1.982986
black	1.369741
chas	1.083922

After removing of `tax` we have all predictors with a VIF less that 5. We save our VIF.5 set as from the working set for further analysis below.

```
VIF.5 <- Boston.ws
```

c

Prepare a predictor set for VIF less than 3 (`VIF.3`) and 2 (`VIF.2`)

Solution.

We are asked to keep iterating until we achieve a `VIF.3` and `VIF.2` predictor set. We continue our iterations by removing `nox`, the next highest VIF in our current working set.

```
Boston.ws <- Boston.ws %>% select(-nox)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
dis	3.848970
medv	3.541833
lstat	3.534978
indus	2.900572
age	2.884712
rm	2.257480
zn	2.193066
rad	2.082554
ptratio	1.717002
black	1.368255
chas	1.080046

We have not achieved all predictors with VIF less than 3, so we remove `dis` next.

```
Boston.ws <- Boston.ws %>% select(-dis)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
lstat	3.494699
medv	3.314382
indus	2.516595
age	2.426706
rm	2.257473
rad	2.035221
zn	1.754760
ptratio	1.689190
black	1.363823
chas	1.079678

Removing `lstat` and continuing...

```
Boston.ws <- Boston.ws %>% select(-lstat)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
medv	2.714202
indus	2.501214
rm	2.070512
age	2.060694
rad	1.984586
zn	1.747424
ptratio	1.669100
black	1.353535
chas	1.079624

The removal of `lstat` achieves all remaining predictors with VIF less than 3, so we retain our VIF.3 set.

```
VIF.3 <- Boston.ws
```

Then continuing iterating by removing `medv`.

```
Boston.ws <- Boston.ws %>% select(-medv)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
indus	2.488210
age	2.019256

Predictor	VIF
rad	1.984585
zn	1.745347
ptratio	1.529119
rm	1.291832
black	1.279052
chas	1.049946

Starting to achieve a pretty restricted predictor set, but need to remove `indus`.

```
Boston.ws <- Boston.ws %>% select(-indus)
VIF.single(Boston.ws) %>% kable()
```

Predictor	VIF
age	1.742165
rad	1.739217
zn	1.674486
ptratio	1.528779
black	1.265661
rm	1.207545
chas	1.043482

Removal of `indus` achieves all predictors with VIF less than 2.

```
VIF.2 <- Boston.ws
```

We now have our three predictors sets, saved to `VIF.5`, `VIF.3`, and `VIF.2`.

d

Run 10-repeats of 10-fold cross-validation on the three different VIF predictor sets. Estimate logistic regression models for above/below median crime rate. Report the mean test set accuracy/error within a table. How well did your model perform compared to your results from Assignment 5?

Solution.

I set up all the necessary models for running the cross-validation. There are several ways this could be produced. I will use our VIF work above and the known HW5 model `crim01 ~ indus + nox + rad + tax + black + medv` to save formula objects for each model we would like to test. I will then use the full Boston data set (with appropriate response changes) of which I can then make the same cross-validation splits, and only model the proper predictors needed. Overall we compare 4 models on the same 10x10 KCV.

```
### setup crim01 response variable.
Boston <- Boston %>% mutate(crim01 = factor(ifelse(crim>median(crim),1,0))) %>%
  select(-crim)
```

```
### formula for VIF.5
VIF.5.model <- formula(paste('crim01~', paste(colnames(VIF.5), collapse='+')))

## By doing this I can now fit models like this.
# glm(VIF.5.model, data=Boston, family='binomial')

### create a VIF.3 set by grabbing crim01 and columns of VIF.3
VIF.3.model <- formula(paste('crim01~', paste(colnames(VIF.3), collapse='+')))

### create a VIF.2 set by grabbing crim01 and columns of VIF.2
VIF.2.model <- formula(paste('crim01~', paste(colnames(VIF.2), collapse='+')))

### HW5 model using BSS analysis of LRs
HW5.model <- formula('crim01 ~ indus + nox + rad + tax + black + medv')
```

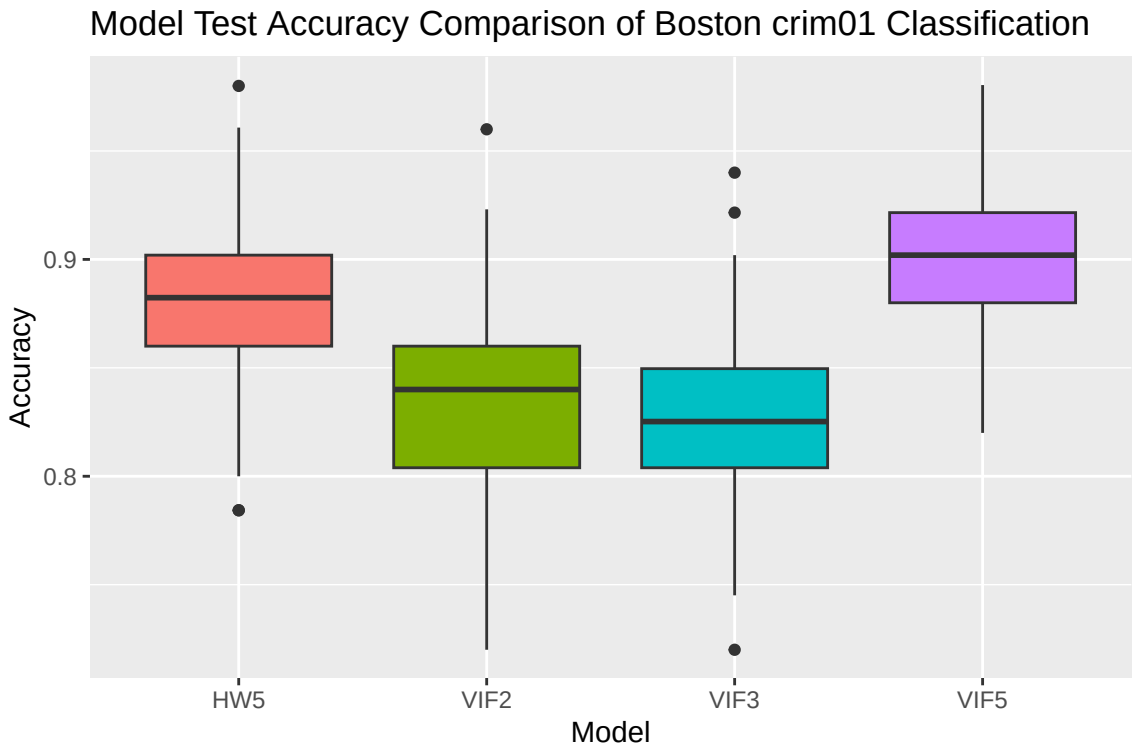
With all of our models defined, I set up my 10x10 KCV for proper caching and to compare all models at one time. I setup a matrix for the accuracy outputs that has 4 columns and 100 rows.

```
set.seed(10)
acc <- data.frame()
for(i in 1:10)
{
   # create folds - stratifying on crim01
   folds.temp <- caret::createFolds(Boston$crim01, k=10)
   for(k in 1:10)
   {
      # iterate over folds and test models
      Train <- Boston %>% slice(-folds.temp[[k]])
      Test <- Boston %>% slice(folds.temp[[k]])

      # VIF5
      glm.fit <- glm(VIF.5.model, data=Train, family='binomial')
      glm.pred <- predict(glm.fit, Test, type='response')
      glm.class <- ifelse(glm.pred >0.5, 1, 0)
      acc.5 <- mean(Test$crim01==glm.class)

      # VIF3
      glm.fit <- glm(VIF.3.model, data=Train, family='binomial')
      glm.pred <- predict(glm.fit, Test, type='response')
      glm.class <- ifelse(glm.pred >0.5, 1, 0)
      acc.3 <- mean(Test$crim01==glm.class)

      # VIF2
      glm.fit <- glm(VIF.2.model, data=Train, family='binomial')
      glm.pred <- predict(glm.fit, Test, type='response')
      glm.class <- ifelse(glm.pred >0.5, 1, 0)
      acc.2 <- mean(Test$crim01==glm.class)

      # HW5
```

```
      glm.fit <- glm(HW5.model, data=Train, family='binomial')
      glm.pred <- predict(glm.fit, Test, type='response')
      glm.class <- ifelse(glm.pred >0.5, 1, 0)
      acc.hw5 <- mean(Test$crim01==glm.class)

      #storage
      acc <- acc %>% rbind(c(acc.5, acc.3, acc.2, acc.hw5))
   }
   colnames(acc) <- c("VIF5", "VIF3", "VIF2" , "HW5")
}
```

With our cross-validation run we can make model comparisons.

```
acc.long <- acc %>%
  pivot_longer(everything(), names_to="Model", values_to="Accuracy")
ggplot(acc.long, aes(x = Model, y = Accuracy)) +
  geom_boxplot(aes(fill=Model)) +
  theme(legend.position = "none") +
  labs(title = "Model Test Accuracy Comparison of Boston crim01 Classification")
```



We observe that `VIF5` model works extremely well on unseen data! Although my best subset analysis unveiled a pretty outstanding model, the VIF5 model does have improved accuracy for prediction of above/below median crime rate! The secondary learning objective here, besides the power of variance inflation factors, is that over reduction of VIF typically can reduce predictor sets too far. Notice that `VIF2` and `VIF3` seem to be doing slightly worse than `VIF5`, likely because we have removed too much information from the data frame! We might want to compare these via some statistics.

```
acc.long %>% group_by(Model) %>%
  summarize(Mean = mean(Accuracy), SD = sd(Accuracy),
            Median = median(Accuracy), IQR = IQR(Accuracy)) %>%
  kable()
```

Model	Mean	SD	Median	IQR
HW5	0.8834615	0.0407469	0.8823529	0.0419608
VIF2	0.8325195	0.0435044	0.8400000	0.0560784
VIF3	0.8273250	0.0398712	0.8252262	0.0456938
VIF5	0.9024149	0.0383652	0.9019608	0.0415686

Competitive modeling, but it seems VIF5 is a clear winner. Surprised that HW5 didn't do a bit better, but I built the best subset using 66/33 subset percent splits instead of 10x10 KCV. NOW you are ready to investigate further. You have done about all that is possible with Boston on the main effects with NO transformations. The next steps to improve items further might be to try some variable transforms, seeing how these might affect the relationship between the predictor and the classes. You could also start introducing interaction terms, higher order terms, and other elements of non-linearity.

Exercise 3

Review the `glmnet` vignette found at https://glmnet.stanford.edu/articles/glmnet.html. This was recently updated with the current form of the `glmnet` package and is full of useful information! Review through `Guassian` linear regression at a minimum, most should review through `Logistic Regression`. Reviewing the `glmnet` vignette is required to help you run the analysis below.

Solution.

Read as much as you desired here. They are constantly updating the webpage and the package. I hope you are feeling more confident that you can read new material and apply complex models without aid from an instructor!

Exercise 4

We will predict the number of applications received using all other variables in the `College` data set.

a Load the data and give a brief evaluation of the predictors (we should never blindly apply models without an understanding of the data). Remove `Accept`, `Enroll`, `Top10perc`, and `Top25perc` from the data frame.

Solution.

```
data('College', package='ISLR')
#str(College)
College.2 <- College %>% select(-Accept, -Enroll, -Top10perc, -Top25perc)
```

b

Produce a two-thirds/one-third split of the `College` data for training and testing.

Solution.

```
index <- sample(1:nrow(College.2), ceiling(nrow(College.2)*0.66))
Train <- College.2[index,]
Test <- College.2[-index,]
```

c

Estimate a backward hybrid stepwise linear model using the training set. Report the test set root MSE (RMSE).

Solution.

I did not specify a metric in the question. I happen to like and typically use BIC. If you defaulted the options you may have used AIC.

```
lm.1 <- step(lm(Apps ~ ., data=Train), k=log(nrow(Train)), trace=FALSE)
coef(lm.1)
```

```
##   (Intercept)    PrivateYes   F.Undergrad    Room.Board        Expend
## -3268.7697430  -761.9548440     0.5999508     0.3939613     0.1125015
##     Grad.Rate
##    25.9789413
```

```
preds.lm <- predict(lm.1, Test)
mean((Test$Apps - preds.lm)^2) %>% sqrt()
```

```
## [1] 1674.912
```

The resulting model using backward stepwise with BIC gives a 5 predictor model with a test set MSE of 1674.9. This is to be used as a reference point for the below models. This is SELECTION as you have seen it before - now we can contrast that to SELECTION via an L1-penalty!

d

The `glmnet` functions will require you to produce design matrices. Produce the design matrix for the training and test set. You may use `model.matrix()` from base R, or the built in `makeX()` function from the glmnet package. It does not use the `y ~ x` form we commonly use.

Solution.

The new vignette has some additional approaches, here is how I setup my Elastic-Net work. We want to keep all predictors we have in the model and allow the penalization to make decisions for us.

```
x.train<-model.matrix(Apps~., data=Train)
x.test<-model.matrix(Apps~., data=Test)
y.train <- Train$Apps
```

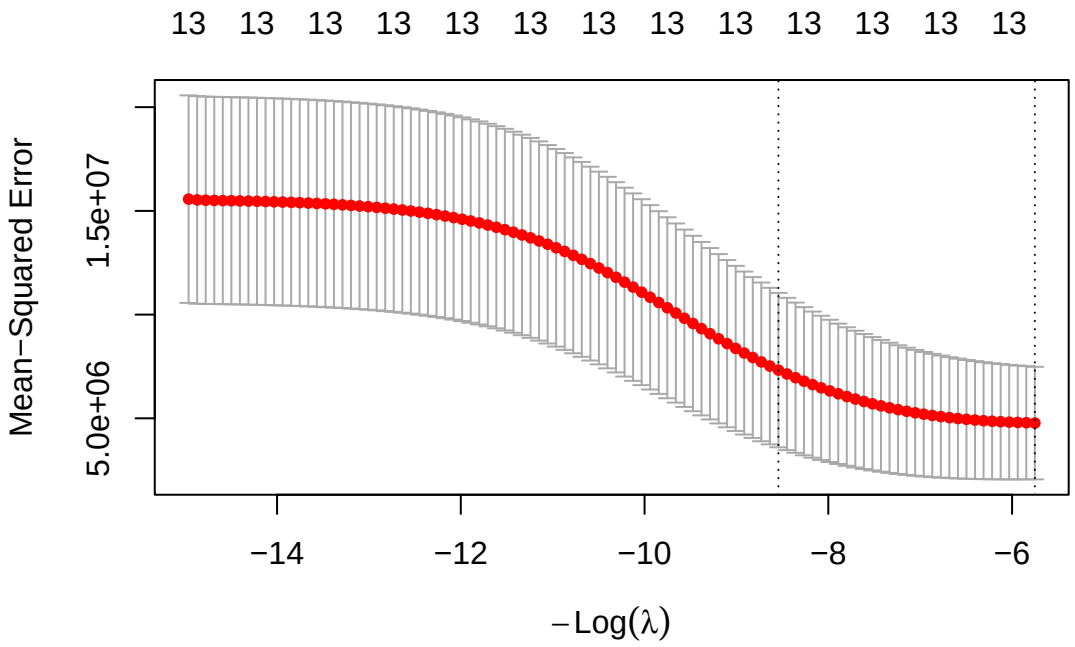
e

Estimate a regularized model using ridge regression. Use the design matrix you produce in **d.** and let `y` be the response vector `Apps`. Validate the penalty coefficient, λ, using `cv.glmnet()`. Plot the `cv.glmnet` object to visualize the cross-validation result.

Solution.

We are asked to start with ridge regression, which requires we set $\alpha = 0$. We then validate the penalty and output the model visualizations showing its cross-validation performance. Report the λ and test set RMSE for the one-third left out set using the penalty coefficient λ taken from `lambda.min`. Export the coefficients for the model at `lambda.min`.

```
cv.test.ridge <- cv.glmnet(x=x.train, y=y.train, alpha=0)
plot(cv.test.ridge)
```



We get the cross-validation output in $-\log(\lambda)$, so the larger penalty is to the left, with decreasing penalty to the right. We observe that ridge cannot perform selection, so the number of predictors in the model stays constant at 13, but we do see some minimization of error occurring as the shrinkage is applied.

```
coef(cv.test.ridge, s='lambda.min')
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                 lambda.min
## (Intercept) -3.418973e+03
## (Intercept)  .
## PrivateYes  -1.078111e+03
```

```
## F.Undergrad  5.294744e-01
## P.Undergrad -4.430337e-03
## Outstate     6.743545e-02
## Room.Board   2.792098e-01
## Books        4.007270e-01
## Personal    -2.893008e-03
## PhD          5.177242e+00
## Terminal    -1.586956e+00
## S.F.Ratio    2.939234e+01
## perc.alumni -1.659391e+01
## Expend       1.107896e-01
## Grad.Rate    2.537882e+01
```

```
preds.out.ridge <- predict(cv.test.ridge, x.test, s='lambda.min')
mean((Test$Apps - preds.out.ridge)^2) %>% sqrt()
```

```
## [1] 1728.69
```

The final model has all predictors retained, but notice that several predictors shrink greatly, with the coefficients of `P.Udergrad` and `Outstate` become as small as 10^{-3}. More important predictors, such as `PrivateYes` retain a high coefficient value. This is the effect of ridge diffusion any multicollinearity and the key drivers of `Apps` retain a larger coefficient value. We obtain a single mean test set MSE of 1728.7, a little worse than our step model. So even though no selection has not occurred, the L2-penalty resolves the multicollinearity a bit!

Note. Amazing how much these change with one iteration. My entire first run had ridge giving a very small MSE, but when compiled to RMD a lot changed. This is why we should ALWAYS look at multiple test/train splits to see the real behavior.

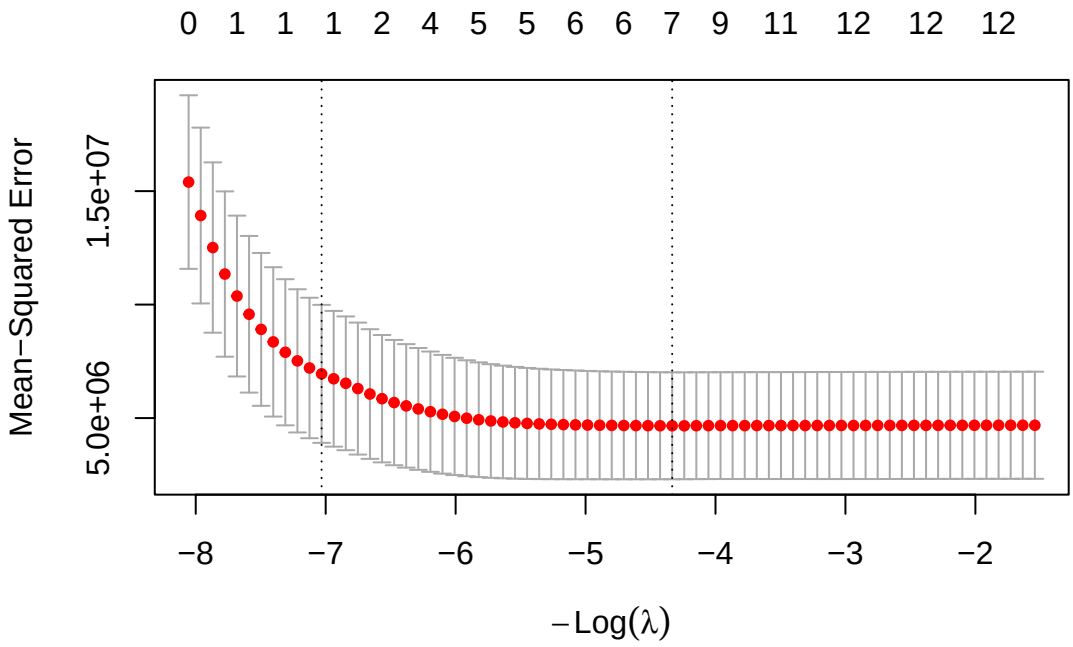
f

Repeat (e) using LASSO regression.

Solution.

We now use LASSO and set $\alpha = 1$. The same code is provided, with only this change.

```
cv.test.lasso <- cv.glmnet(x=x.train, y=y.train, alpha=1)
plot(cv.test.lasso)
```



Same directionailty for penalty size. Notice with a very large penalty, LASSO removes all predictors from the model (leaving only an intercept, i.e. the null model). Reducing the penalty size we see the number of predictors in the model shifting (growing as we penalize less). There seems to be an interesting optimal at $p = 2$ predictors int he model, this would be the `lambda.1se`, which we are not currently using (although I'd explore it more with this type of output!). The minimum error model seems to have about 8 retained predictors.

```
coef(cv.test.lasso, s='lambda.min')
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                 lambda.min
## (Intercept) -2.765974e+03
## (Intercept)  .
## PrivateYes  -4.943471e+02
## F.Undergrad  5.980332e-01
## P.Undergrad  .
## Outstate     2.823908e-02
## Room.Board   2.912606e-01
## Books        .
## Personal     .
## PhD          .
## Terminal     .
## S.F.Ratio    .
## perc.alumni -4.435177e+00
## Expend       9.980265e-02
## Grad.Rate    2.113630e+01
```

```
preds.out.lasso <- predict(cv.test.lasso, x.test, s='lambda.min')
mean((Test$Apps - preds.out.lasso)^2) %>% sqrt()
```

```
## [1] 1694.647
```

With LASSO we begin to see the selection occurring. A . in the above output indicates selection out of the model. This is what a matrix looks like when using *sparse* conditions. Notice that many of the variables that were being highly shrunk by ridge are not completely selected out by LASSO - the behavior of an L1-penalty. However, there is multicollinearity present, so the coefficient estimates are a bit variable (some very low, some very high, a very large intercept term). Thus, the single test set MSE of 1694.6 on par with the backward step.

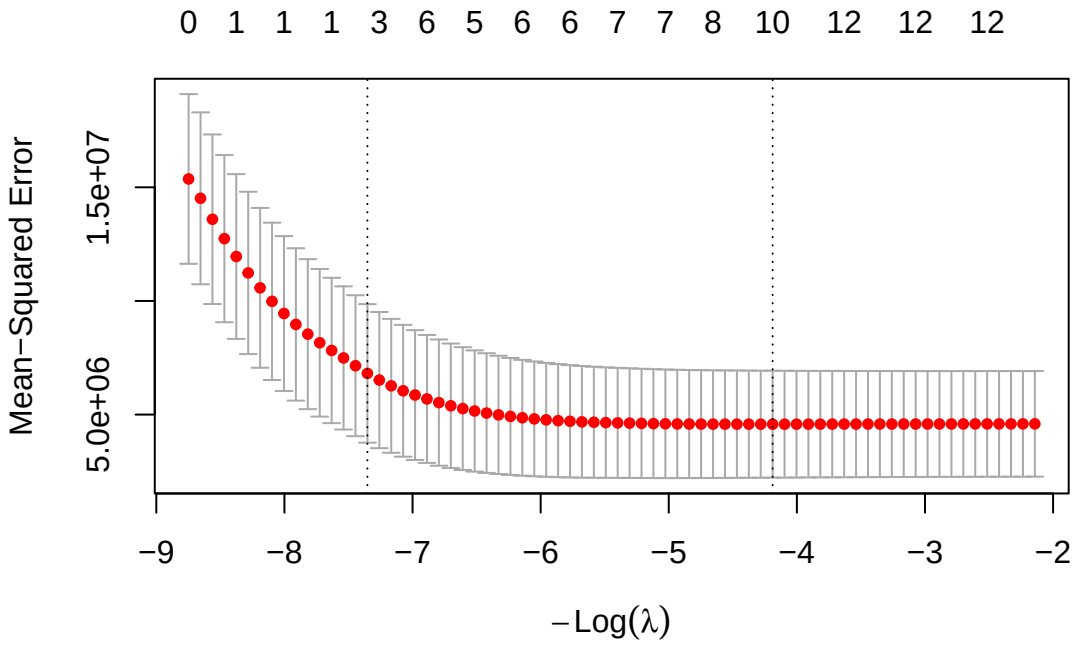
g

Repeat (e.) using the Elastic-Net. Set $\alpha = 0.5$.

Solution.

Same work with $\alpha = 0.5$

```
cv.test.enet <- cv.glmnet(x=x.train, y=y.train, alpha=0.5)
plot(cv.test.enet)
```



A similar behavior with LASSO in selection occurring with a mixture of L1- and L2-penalization. Now finding 10 predictors at `lambda.min`. I'm still interested in exploring this 2 predictor model at `lambda.1se`.

```
coef(cv.test.enet, s='lambda.min')
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                 lambda.min
## (Intercept) -3.076467e+03
## (Intercept)  .
## PrivateYes  -7.391376e+02
## F.Undergrad  5.934919e-01
## P.Undergrad -3.566340e-02
## Outstate     5.661089e-02
## Room.Board   2.900133e-01
```

```
## Books         7.019646e-02
## Personal      .
## PhD           .
## Terminal      .
## S.F.Ratio     1.189967e+01
## perc.alumni -1.238950e+01
## Expend        1.063268e-01
## Grad.Rate     2.357499e+01
```

```
preds.out.enet <- predict(cv.test.enet, x.test, s='lambda.min')
mean((Test$Apps - preds.out.enet)^2) %>% sqrt()
```

```
## [1] 1675.252
```

We see shrinkage and selection occurring, some coefficients are shrinking nicely, others have been fully selected out. The model does not perform as well as ridge (on this one split), but does better than a pure L1-penalty due to the presence of multicollinearity. Single test set MSE of 1675.3.

h

Make a table comparing the coefficients from each model type. What are some differences between the estimated models?

Solution.

Just a request to smoosh the coefficient outputs together and line up some of the comments I've made above.

```
Coef.frame <- cbind(coef(cv.test.ridge, s='lambda.min'),
      coef(cv.test.enet, s='lambda.min'),
      coef(cv.test.lasso, s='lambda.min'))
colnames(Coef.frame) <- c("Ridge", "Elastic-net", "LASSO")
Coef.frame
```

```
## 15 x 3 sparse Matrix of class "dgCMatrix"
##                      Ridge   Elastic-net         LASSO
## (Intercept) -3.418973e+03 -3.076467e+03 -2.765974e+03
## (Intercept)  .             .             .
## PrivateYes  -1.078111e+03 -7.391376e+02 -4.943471e+02
## F.Undergrad  5.294744e-01  5.934919e-01  5.980332e-01
## P.Undergrad -4.430337e-03 -3.566340e-02  .
## Outstate     6.743545e-02  5.661089e-02  2.823908e-02
## Room.Board   2.792098e-01  2.900133e-01  2.912606e-01
## Books        4.007270e-01  7.019646e-02  .
## Personal    -2.893008e-03  .             .
## PhD          5.177242e+00  .             .
## Terminal    -1.586956e+00  .             .
## S.F.Ratio    2.939234e+01  1.189967e+01  .
## perc.alumni -1.659391e+01 -1.238950e+01 -4.435177e+00
## Expend       1.107896e-01  1.063268e-01  9.980265e-02
## Grad.Rate    2.537882e+01  2.357499e+01  2.113630e+01
```

Shrinking with ridge, selection with LASSO, and both effects occurring with elastic-net. It is not surprising to see the pure L1-penalty removes the most predictors from the model, and the elastic-net in between. Elastic-net can perform very well in helping resolve multicolliearity and helping to push our more predictors. This did not occur on this iteration, but could on a slightly different split.

i

Compare the single iteration test set MSE values? Is there much of a difference between the methods?

Solution.

```
testMSE.compare <- data.frame(step = mean((Test$Apps - preds.lm)^2) %>% sqrt(),
                              ridge = mean((Test$Apps - preds.out.ridge)^2) %>% sqrt(),
                              enet = mean((Test$Apps - preds.out.enet)^2) %>% sqrt(),
                              lasso = mean((Test$Apps - preds.out.lasso)^2) %>% sqrt())
testMSE.compare %>% kable(caption = "Single test set MSE comparison.")
```

Table 9: Single test set MSE comparison.

step	ridge	enet	lasso
1674.912	1728.69	1675.252	1694.647

People often under estimate the usefulness of ridge. Dealing with multicollinearity is often a bigger issue than selection! This iteration shows that ENET and LASSO are similar to the step result. My guess - if we were to cross-validate, ENET would be best, Ridge 2nd best, and LASSO worst (due to the presence of multicollinearity). This one draw is a bit misleading.

j

Discuss what is happening to the regularized models when we let $\lambda = 0$ and what changes when we increase the size of the penalty coefficient λ. What is the limiting behavior as $\lambda \to \infty$?

Solution.

I have done this throughout my document. See above.