ASSIGNMENT 4

TITLE: REGULARIZATION

NAME: NAM HO PHAN

COURSE: 6015

PROFESSOR: VLADIMIR SHAPIRO

INTRODUCTION:

One of the major aspects of training your machine learning model is avoiding overfitting. The model will have a low accuracy if it is overfitting. This happens because your model is trying too hard to capture the noise in your training dataset. By noise we mean the data points that don't really represent the true properties of your data, but random chance. Learning such data points, makes your model more flexible, at the risk of overfitting (Gupsta, 2017). Conversely, if we take so few features, it could lead to the underfitting because the model is too simpler to predict the outcome accurately.

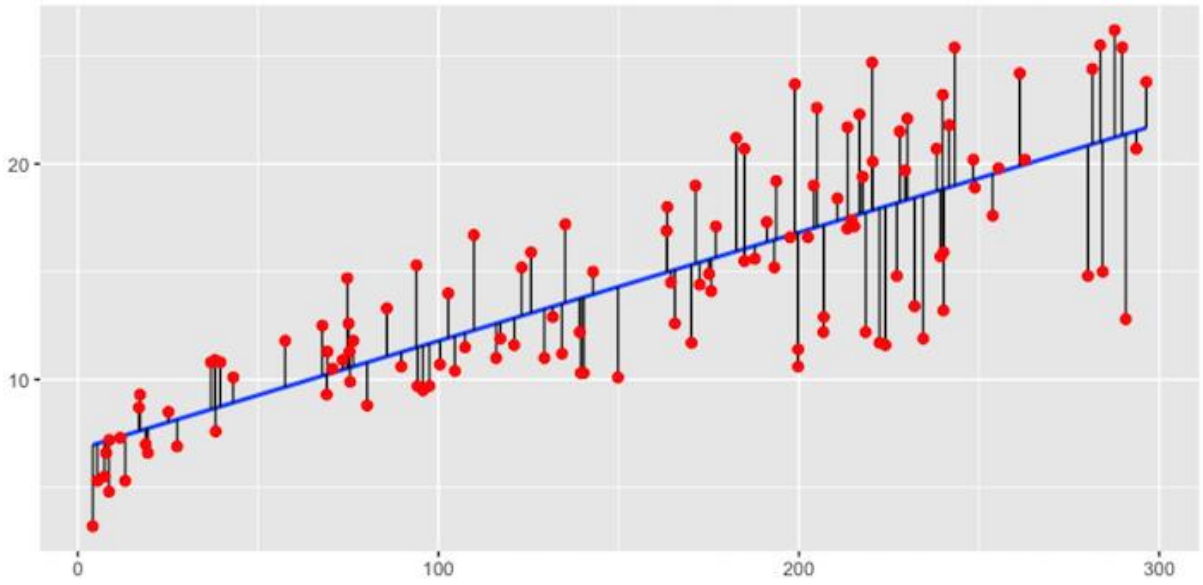
Thus, here are some techniques to prevent overfitting:

- Cross-validation, Cross-validation is a powerful preventative measure against overfitting. Cross-validation allows you to tune hyperparameters with only your original training set. This allows you to keep your test set as a truly unseen dataset for selecting your final model.
- Removing the data. Because we need to collect the appropriate features to avoid the high coefficient, leading to overfitting of the model, we can remove some features causing that overfitting. For example, the income and the yearly expense are very highly variant so we cannot put it together.
- However, if you want to keep your variables in the model but still preventing its risk of overfitting, you can use regularization, which will constrain the coefficient estimate toward zero (Gupsta, 2017).

For this assignment, we are going to try Ridge regression and LASSO regression to see how these methods impact on the predicted model.

ANALYSIS:

1. Ridge Regression



From the plot, we can see how the overfitting when the distance from predicted outcome to actual outcome is too far. Thus, to make the distance short, we should minimize the sum of squared errors {SSE = n∑i=1(yi−^yi)2}.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

The formula of ridge regression shows us how it minimizes the coefficient because the flexibility of our model will increase the

coefficient. λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.

```
#load package
library(ISLR)
library(plyr)
library(readr)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(ggplot2)
library(repr)
library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.0-2

library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
college_data = ISLR::College
#explore data
college_data$Private = as.numeric(college_data$Private)
college_data$Private = as.factor(college_data$Private)
glimpse(college_data)
```

```
## Rows: 777
## Columns: 18
## $ Private     <fct> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2...
## $ Apps        <dbl> 1660, 2186, 1428, 417, 193, 587, 353, 1899, 1038, 582,
...
## $ Accept      <dbl> 1232, 1924, 1097, 349, 146, 479, 340, 1720, 839, 498,
1...
## $ Enroll      <dbl> 721, 512, 336, 137, 55, 158, 103, 489, 227, 172, 472,
4...
## $ Top10perc   <dbl> 23, 16, 22, 60, 16, 38, 17, 37, 30, 21, 37, 44, 38, 44
,...
## $ Top25perc   <dbl> 52, 29, 50, 89, 44, 62, 45, 68, 63, 44, 75, 77, 64, 73
,...
## $ F.Undergrad <dbl> 2885, 2683, 1036, 510, 249, 678, 416, 1594, 973, 799,
1...
## $ P.Undergrad <dbl> 537, 1227, 99, 63, 869, 41, 230, 32, 306, 78, 110, 44,
...
## $ Outstate    <dbl> 7440, 12280, 11250, 12960, 7560, 13500, 13290, 13868,
1...
## $ Room.Board  <dbl> 3300, 6450, 3750, 5450, 4120, 3335, 5720, 4826, 4400,
3...
## $ Books       <dbl> 450, 750, 400, 450, 800, 500, 500, 450, 300, 660, 500,
...
## $ Personal    <dbl> 2200, 1500, 1165, 875, 1500, 675, 1500, 850, 500, 1800
,...
## $ PhD         <dbl> 70, 29, 53, 92, 76, 67, 90, 89, 79, 40, 82, 73, 60, 79
,...
## $ Terminal    <dbl> 78, 30, 66, 97, 72, 73, 93, 100, 84, 41, 88, 91, 84, 8
7...
## $ S.F.Ratio   <dbl> 18.1, 12.2, 12.9, 7.7, 11.9, 9.4, 11.5, 13.7, 11.3, 11
....
## $ perc.alumni <dbl> 12, 16, 30, 37, 2, 11, 26, 37, 23, 15, 31, 41, 21, 32,
...
## $ Expend      <dbl> 7041, 10527, 8735, 19016, 10922, 9727, 8861, 11487, 11
6...
## $ Grad.Rate   <dbl> 60, 56, 54, 59, 15, 55, 63, 73, 80, 52, 73, 76, 74, 68
,...
```

```r
sample_rows <- sample(nrow(college_data), nrow(college_data)*75/100)

#training set
college_train = college_data[sample_rows,]
```

```r
#check the overfitting
college_model = lm(Grad.Rate ~., data = college_train)

summary(college_model)
```

```
##
## Call:
## lm(formula = Grad.Rate ~ ., data = college_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -36.343  -6.708  -0.424   6.908  48.775
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.5947565  5.1783584   6.488 1.91e-10 ***
## Private2     2.2132515  1.8588502   1.191 0.234289
## Apps         0.0014020  0.0004457   3.146 0.001744 **
## Accept      -0.0016125  0.0009209  -1.751 0.080504 .
## Enroll       0.0028446  0.0026808   1.061 0.289082
## Top10perc   -0.0347332  0.0792873  -0.438 0.661505
## Top25perc    0.2224262  0.0618819   3.594 0.000354 ***
## F.Undergrad -0.0002095  0.0004788  -0.438 0.661908
## P.Undergrad -0.0014931  0.0003960  -3.771 0.000180 ***
## Outstate     0.0010430  0.0002547   4.095 4.84e-05 ***
## Room.Board   0.0018699  0.0006304   2.966 0.003143 **
## Books       -0.0009394  0.0030376  -0.309 0.757229
## Personal    -0.0014727  0.0008156  -1.806 0.071491 .
## PhD          0.1491271  0.0598824   2.490 0.013049 *
## Terminal    -0.1656891  0.0638893  -2.593 0.009751 **
## S.F.Ratio   -0.0386732  0.1721526  -0.225 0.822337
## perc.alumni  0.3395588  0.0504544   6.730 4.18e-11 ***
## Expend      -0.0004384  0.0001774  -2.471 0.013772 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.77 on 564 degrees of freedom
## Multiple R-squared:  0.5259, Adjusted R-squared:  0.5116
## F-statistic:  36.8 on 17 and 564 DF,  p-value: < 2.2e-16
```

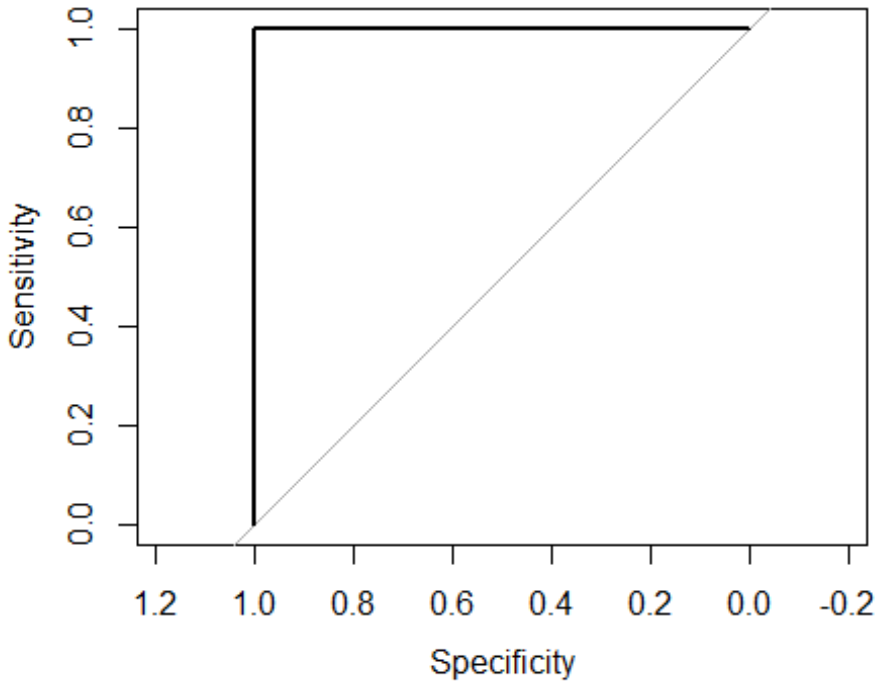
```r
college_train$pred = predict(college_model,type="response")

auc = auc(college_train$Grad.Rate,college_train$pred)
```

```
## Warning in roc.default(response, predictor, auc = TRUE, ...): 'response'
## has more than two levels. Consider setting 'levels' explicitly or using
## 'multiclass.roc' instead

## Setting levels: control = 15, case = 18

## Setting direction: controls > cases
```

```
plot(roc(college_train$Grad.Rate,college_train$pred))
```

```
## Warning in roc.default(college_train$Grad.Rate, college_train$pred): 'resp
onse'
## has more than two levels. Consider setting 'levels' explicitly or using
## 'multiclass.roc' instead
```

```
## Setting levels: control = 15, case = 18
## Setting direction: controls > cases
```

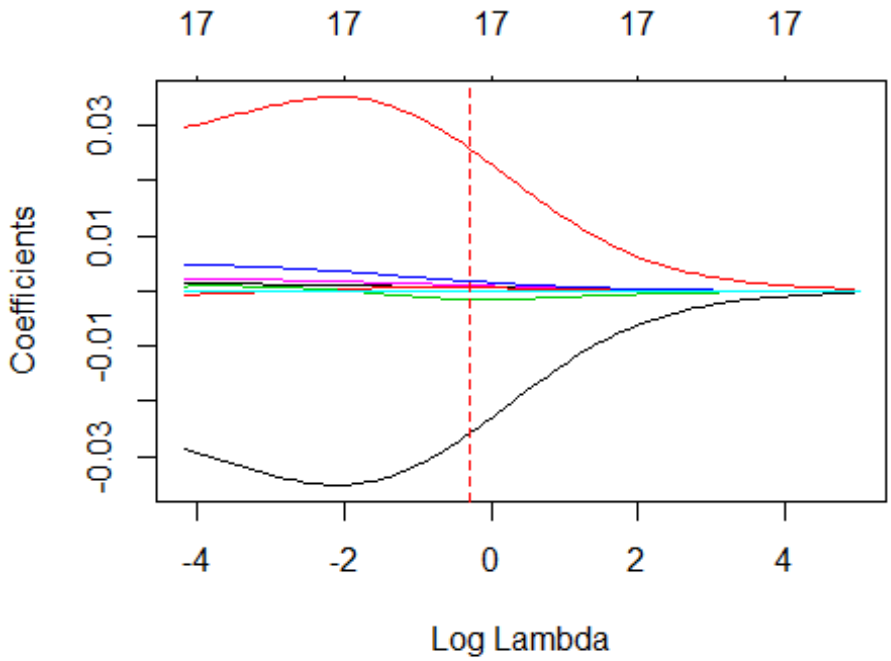



```
auc
```

```
## Area under the curve: 1
```

This model is overfitting because its AUC is 1. I assume it is too high compared to a good model (usually, 0.7-0.8). There is another way to check if we need to constrain the coefficient due to the plot of the function glmnet.

```
ridge_reg = glmnet(x,y_train,alpha=0)
plot(ridge_reg,xvar="lambda")
abline(v = log(cv_ridge$lambda.1se), col = "red", lty = "dashed")
```



As we can see that the larger value lambda increases, the smaller value the coefficient approach. The constraint lambda is where we can predict the model while maintaining all the features in the model. To perform how to apply ridge regression to minimize the coefficient. I will split data into train and test set, which is used to evaluate the accuracy of the prediction model and the model for the future value.

```
#split data into train and test set
N = nrow(college_data)
target = round(N*0.75)
vector = runif(N)

sample_rows <- sample(nrow(college_data), nrow(college_data)*75/100)
```

```r
#training set
college_train = college_data[sample_rows,]
dim(college_train)
```

```
## [1] 582  18
```

```r
#test set
college_test= college_data[-sample_rows,]
dim(college_test)
```

```
## [1] 195  18
```

```r
#select all explanatory variables
cols_reg = c("Private","Apps","Accept","Enroll","Top10perc","F.Undergrad","P.
Undergrad","Outstate","Room.Board","Books","Personal","PhD","Terminal","S.F.R
atio","perc.alumni","Expend","Grad.Rate")

#convert it into dummy variables
dummies = dummyVars(Grad.Rate~.,college_data[,cols_reg])

#predict the outcome for both sets
train_dummies = predict(dummies,newdata=college_train[, cols_reg])
test_dummies = predict(dummies,newdata=college_test[, cols_reg])

dim(train_dummies)
```

```
## [1] 582  17
```

```r
dim(test_dummies)
```

```
## [1] 195  17
```

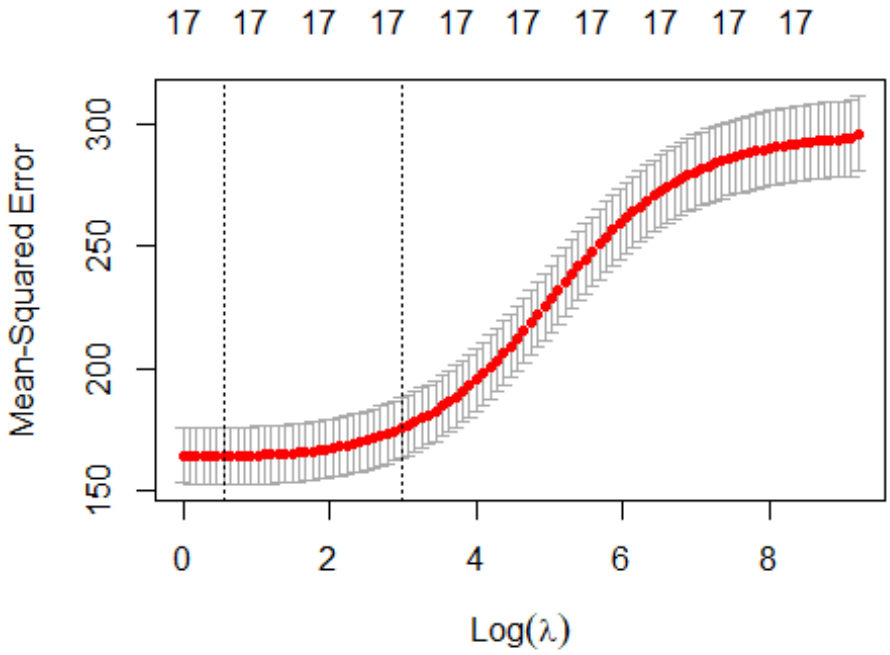
```r
#convert into matrix
x = as.matrix(train_dummies)
y_train = log(college_train$Grad.Rate)

x_test = as.matrix(test_dummies)
y_test = log(college_test$Grad.Rate)

#apply cv.glmnet to find the lambda
cv_ridge = cv.glmnet(x,y_train,alpha=0)
cv_ridge
```

```
##
## Call:  cv.glmnet(x = x, y = y_train, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure      SE Nonzero
## min 0.0557 0.05056 0.00556      17
## 1se 0.7537 0.05595 0.00642      17
```

```
min(cv_ridge$cvm)
```

```
## [1] 0.05055689
```

```
plot(cv_ridge)
```



Applying cv.glmnet to know the optimal lambda for the model. So, I get the min lambda (0.05) and largest lambda (0.75). To know which lambda is the best value for using, I apply it into finding coefficient for model. Then we got the coefficient of min lambda which is so close to 0.

```
#lambdamin and largest
optimal_lambda = cv_ridge$lambda.min
cat(paste("min_lambda",optimal_lambda))
```

```
## min_lambda 0.0557027888471503
```

```
cat(paste("largest_lambda",cv_ridge$lambda.1se))
```

```
## largest_lambda 0.753685343046781
```

```r
#lambda_min

y_train_unlog = college_train$Grad.Rate

ridge_reg1 = glmnet(x,y_train_unlog,alpha=0,family = "gaussian",lambda = cv_r
idge$lambda.min)

coef(ridge_reg1,cv_ridge$lambda.min)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) 39.7335073553
## Private.1   -2.6479240040
## Private.2    0.5134558188
## Apps         0.0009188540
## Accept      -0.0002987555
## Enroll       0.0021863931
## Top10perc    0.1408636191
## F.Undergrad -0.0002765624
## P.Undergrad -0.0015548532
## Outstate     0.0009106581
## Room.Board   0.0021385752
## Books       -0.0005676499
## Personal    -0.0021647474
## PhD          0.1239138282
## Terminal    -0.1054421330
## S.F.Ratio    0.0796076390
## perc.alumni  0.3199175302
## Expend      -0.0003785309
```

```r
sum(coef(ridge_reg1,cv_ridge$lambda.min))
```

```
## [1] 38.15881
```

```r
#lambda_largest

ridge_reg1 = glmnet(x,y_train_unlog,alpha=0,family = "gaussian",lambda = cv_r
idge$lambda.1se)

coef(ridge_reg1,cv_ridge$lambda.1se)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) 38.7219967773
## Private.1   -1.8735317158
## Private.2    1.6672580673
## Apps         0.0006104181
## Accept       0.0002439180
## Enroll       0.0009757714
## Top10perc    0.1426148049
## F.Undergrad -0.0001088583
```

```
## P.Undergrad -0.0014946075
## Outstate     0.0008137409
## Room.Board   0.0020311456
## Books       -0.0008340455
## Personal    -0.0021744589
## PhD          0.0984476155
## Terminal    -0.0737569496
## S.F.Ratio    0.0903841589
## perc.alumni  0.3093641479
## Expend      -0.0002701373
```

```
sum(coef(ridge_reg1,cv_ridge$lambda.1se))
```

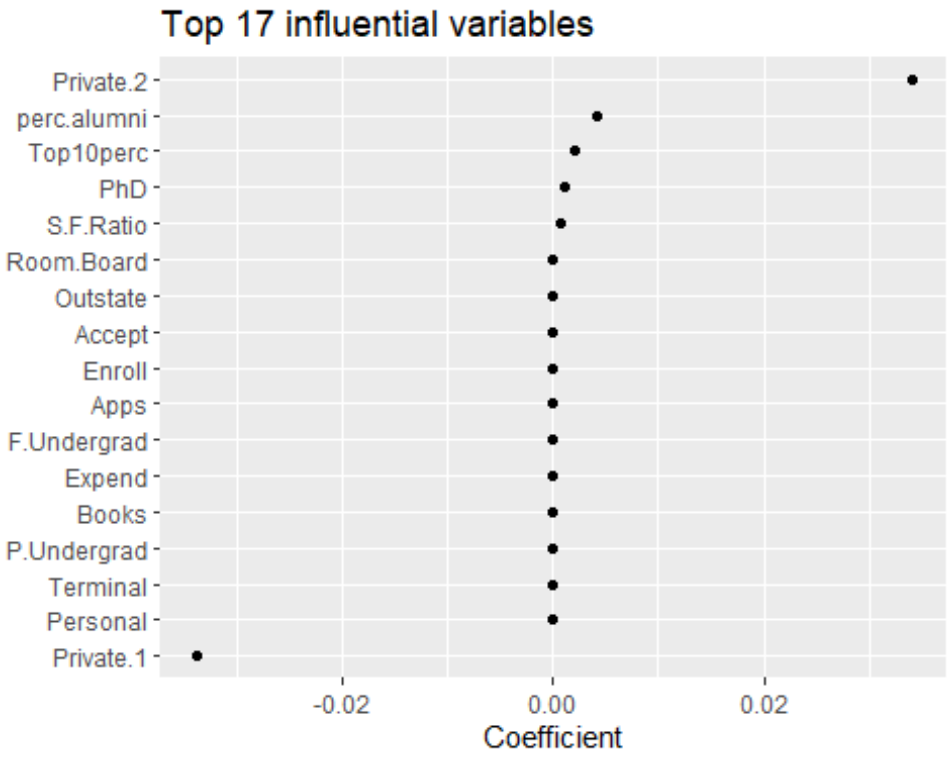
```
## [1] 39.08257
```

It is helpful to know how the coefficients change after I apply min lambda.

```
coef(cv_ridge, s = "lambda.min") %>%
  tidy() %>%
  filter(row != "(Intercept)") %>%
  top_n(17, wt = abs(value)) %>%
  ggplot(aes(value, reorder(row, value))) +
  geom_point() +
  ggtitle("Top 17 influential variables") +
  xlab("Coefficient") +
  ylab(NULL)
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```




```
y_train = log(college_train$Grad.Rate)

#ridge_model
ridge_reg = glmnet(x,y_train,alpha=0,family = "gaussian",lambda = cv_ridge$la
mbda.min)


pred = predict(ridge_reg, s = cv_ridge$lambda.min, x)

# Compute R^2 from true and predicted values
eval_results <- function(true, predicted) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(mean((predicted-true)^2))


  # Model performance metrics
data.frame(
  RMSE = RMSE,
  Rsquare = R_square
)}

eval_results(y_train,pred)
```

```
##        RMSE   Rsquare
## 1 0.2182734 0.3980042

y_test_unlog = college_test$Grad.Rate

#ridge_model
ridge_reg2 = glmnet(x_test,y_test,alpha=0,family = "gaussian",lambda = cv_rid
ge$lambda.min)

#predicted outcome
pred = predict(ridge_reg2,s = cv_ridge$lambda.min,x_test)

# Compute R^2 from true and predicted values
eval_results <- function(true, predicted) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(mean((predicted-true)^2))


  # Model performance metrics
data.frame(
  RMSE = RMSE,
  Rsquare = R_square
)}

eval_results(y_test,pred)

##        RMSE   Rsquare
## 1 0.2579849 0.4487718
```

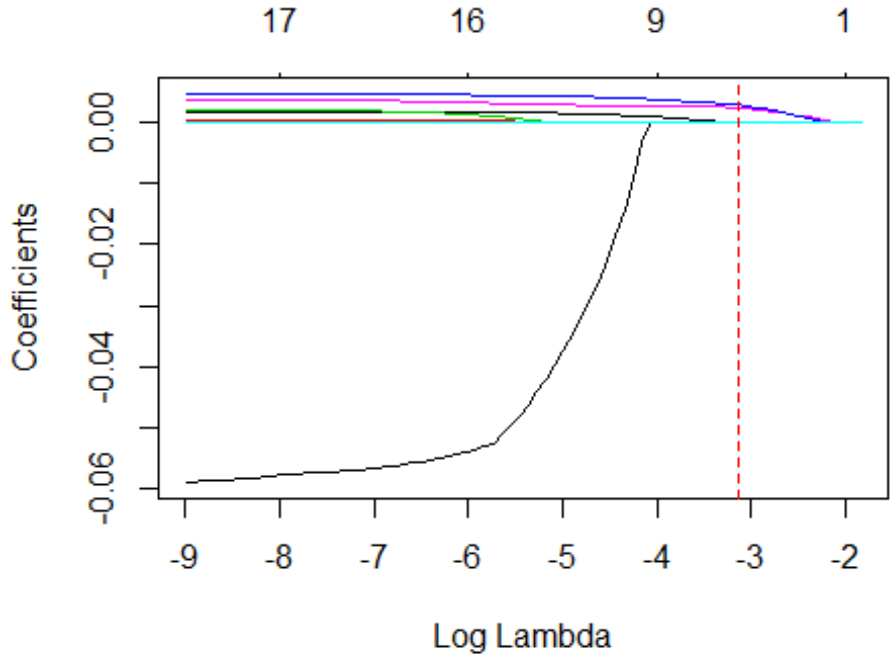
Although I don't get a good R-Squared for the model but I believe the regularization helps my model more accuracy as the RMSE is quite better. Additionally, we see that the model of train and test set is not too different, indicating the good model for prediction.

2. LASSO regression:

The difference of LASSO regression is that it will not try to force the coefficient of all values toward zero. Instead, it tries to reduce features to simplify the model.

$$L_{lasso}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m}|\hat{\beta}_j|.$$

```
lasso_reg = glmnet(x,y_train,alpha=1)
plot(lasso_reg,xvar="lambda")
abline(v = log(cv_lasso$lambda.1se), col = "red", lty = "dashed")
```

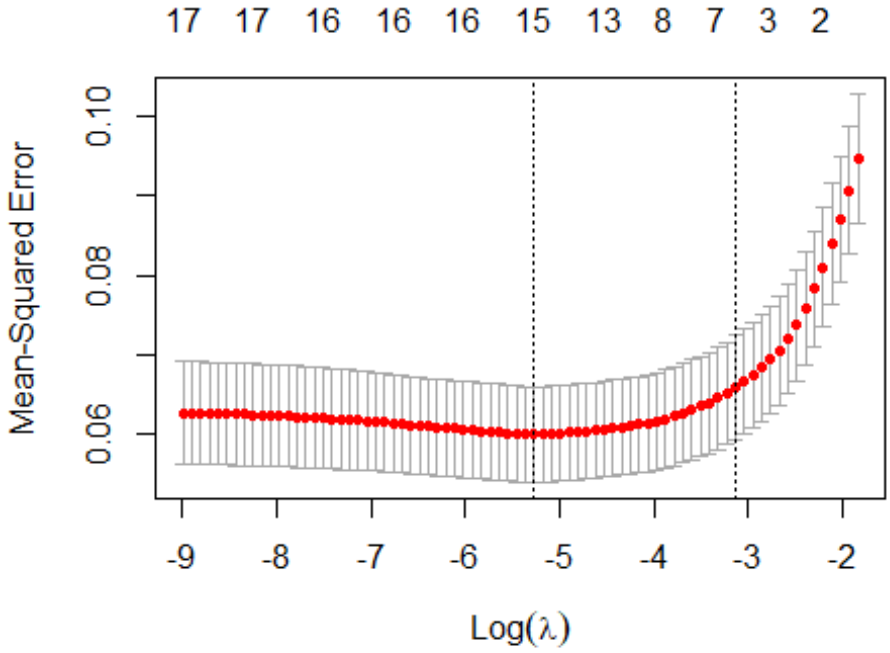


The plot shows where we should constraint the lambda to get the coefficient toward zero. However, this one indicates that if the lambda is larger, the coefficients will be close to 0.

```
#apply cv.glmnet to find the lambda
cv_lasso = cv.glmnet(x,y_train,alpha=1)
cv_lasso

##
## Call:  cv.glmnet(x = x, y = y_train, alpha = 1)
```

```
##
## Measure: Mean-Squared Error
##
##      Lambda Measure       SE Nonzero
## min 0.00513 0.06005 0.005999      15
## 1se 0.04355 0.06592 0.006504       6
```

```
plot(cv_lasso)
```



The first and second vertical dashed lines represent the λ value with the minimum MSE and the largest λ value within one standard error of the minimum MSE.

```
#lambdamin and largest
optimal_lambda = cv_lasso$lambda.min
cat(paste("min_lambda",optimal_lambda))
```

```
## min_lambda 0.00482090886300743
```

```
cat(paste("largest_lambda",cv_lasso$lambda.1se))
```

```
## largest_lambda 0.0594344175900198
```

```
#lambda_min
```

```r
y_train_unlog = college_train$Grad.Rate

lasso_reg1 = glmnet(x,y_train,alpha=1,family = "gaussian",lambda = cv_lasso$l
ambda.min)

coef(lasso_reg1,cv_lasso$lambda.min)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  3.728849e+00
## Private.1   -4.174347e-02
## Private.2    5.883959e-15
## Apps         7.851058e-06
## Accept       5.859591e-06
## Enroll       .
## Top10perc    2.999185e-03
## F.Undergrad  .
## P.Undergrad -2.498406e-05
## Outstate     1.903302e-05
## Room.Board   2.438510e-05
## Books       -1.663115e-05
## Personal    -2.100024e-05
## PhD          2.262672e-04
## Terminal     .
## S.F.Ratio    .
## perc.alumni  3.908802e-03
## Expend      -4.946567e-06
```

```r
sum(coef(lasso_reg1,cv_lasso$lambda.min))
```

```
## [1] 3.694229
```

```r
#lambda_largest

lasso_reg1 = glmnet(x,y_train,alpha=1,family = "gaussian",lambda = cv_lasso$l
ambda.1se)

coef(lasso_reg1,cv_lasso$lambda.1se)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) 3.856501e+00
## Private.1   .
## Private.2   .
## Apps        .
## Accept      .
## Enroll      .
## Top10perc   1.440057e-03
## F.Undergrad .
## P.Undergrad .
## Outstate    2.017177e-05
```

```
## Room.Board  .
## Books       .
## Personal    .
## PhD         .
## Terminal    .
## S.F.Ratio   .
## perc.alumni 1.461183e-03
## Expend      .
```

```
sum(coef(lasso_reg1,cv_lasso$lambda.1se))
```

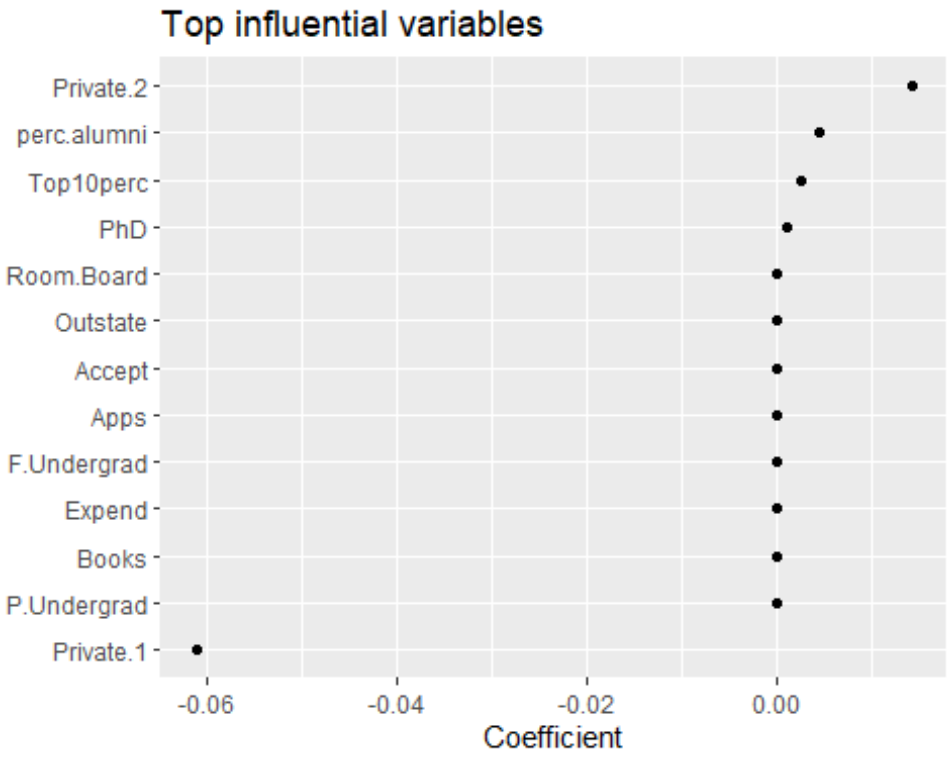
```
## [1] 3.859423
```

As I have mentioned about the feature of LASSO regression, the coefficient will not be forced toward exactly zero, otherwise, it will be eliminated. Additionally, the largest lambda helps us to eliminate unnecessary variables in model better.

```
coef(cv_lasso, s = "lambda.min") %>%
  tidy() %>%
  filter(row != "(Intercept)") %>%
  top_n(17, wt = abs(value)) %>%
  ggplot(aes(value, reorder(row, value))) +
  geom_point() +
  ggtitle("Top influential variables") +
  xlab("Coefficient") +
  ylab(NULL)
```

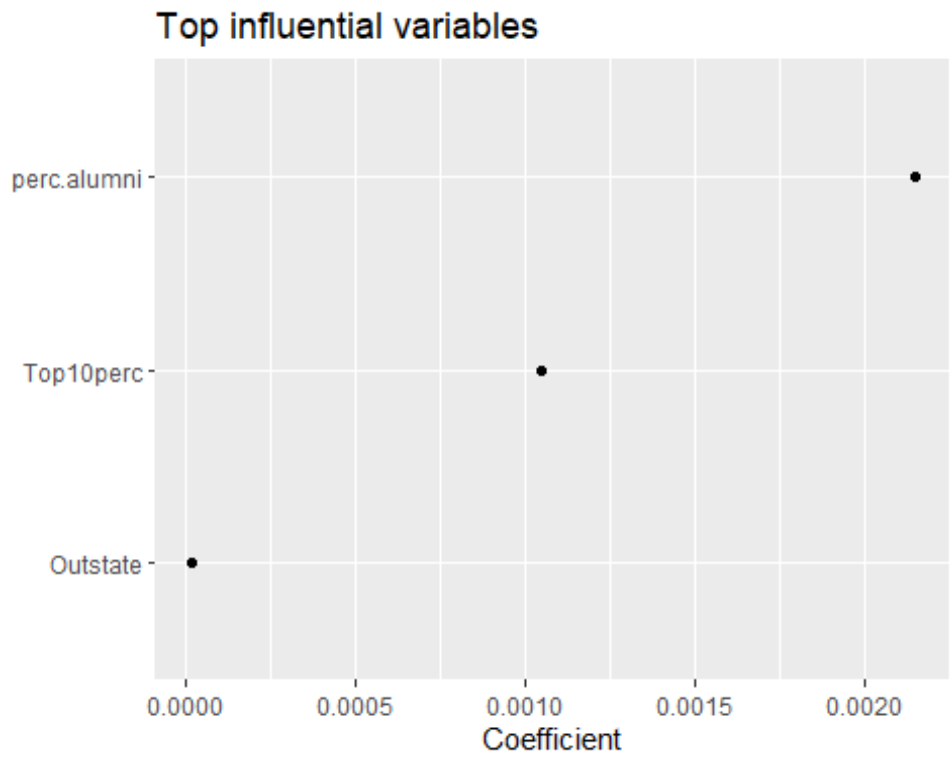
```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```




```
coef(cv_lasso, s = "lambda.1se") %>%
  tidy() %>%
  filter(row != "(Intercept)") %>%
  top_n(17, wt = abs(value)) %>%
  ggplot(aes(value, reorder(row, value))) +
  geom_point() +
  ggtitle("Top influential variables") +
  xlab("Coefficient") +
  ylab(NULL)
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```




```
y_train = log(college_train$Grad.Rate)

#lasso_model
lasso_reg = glmnet(x,y_train,alpha=1,family = "gaussian",lambda = cv_lasso$la
mbda.min)

pred = predict(lasso_reg,s = cv_lasso$lambda.1se,x)

# Compute R^2 from true and predicted values
eval_results <- function(true, predicted) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(mean((predicted-true)^2))


  # Model performance metrics
data.frame(
  RMSE = RMSE,
  Rsquare = R_square
)}

eval_results(y_train,pred)

##         RMSE   Rsquare
## 1 0.2324324 0.3970788
```

```r
y_test_unlog = college_test$Grad.Rate

#ridge_model
lasso_reg2 = glmnet(x_test,y_test,alpha=1,family = "gaussian",lambda = cv_las
so$lambda.min)

#predicted outcome
pred = predict(lasso_reg2,s = cv_lasso$lambda.1se,x_test)

# Compute R^2 from true and predicted values
eval_results <- function(true, predicted) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(mean((predicted-true)^2))


  # Model performance metrics
data.frame(
  RMSE = RMSE,
  Rsquare = R_square
)}

eval_results(y_test,pred)
```

```
##        RMSE   Rsquare
## 1 0.2115151 0.4998554
```

As a result, we have RMSE of training and test set are 0.23 and 0.21, which are very good for model.

COMPARISON

When comparing two types of regularization, I see that the model of Lasso does better a little bit because it gives the higher RMSE and R-Squared. Because Lasso tend to do well when only few predictors impact on the response. Therefore, as I have analyzed the data, the

graduation rate is just actually correlated with some independent variables, so I think my assumption is correct.

The stepwise selection and Ridge and Lasso regression both have advantages and disadvantages. LASSO uses a tuning parameter to penalize the number of parameters in the model. As the number of parameters are greater than the number of observations, Lasso works better, it can eliminate the variables to simplify the model but this can also cause the underfitting; meanwhile, the Ridge regression force all features toward zero but it potentially causes the complexity of the model. Stepwise selection avoids time-consuming to select a best fit model with a better understanding, but it does not give the absolute best combination.

CONCLUSION:

The Ridge and Lasso regularization are very. The advantage is that it can create the potential outcome, They still perform well in cases of a large multivariate data with the number of predictors (p) larger than the number of observations (n), the ridge estimator is good at improving the least-squares estimate when there is multicollinearity, it performs well in cases of a large multivariate data with the number of predictors larger than the number of observations. The disadvantage is that it includes all predictors in the final model, it is not a good choice for model selection, sometimes it will cause the biases as well. Therefore, it is necessary to examine if the regularization is appropriate for the model.

REFERENCES:

Regularized Regression. Retrieved from https://uc-r.github.io/regularized_regression

(Gupta. P., 2017). Regularization in Machine Learning. Retrieved from https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a