BA810 Team Assignment

Rochel Chan, Vaibhav Garimella, Arpit Jain, Zijing Wang, Cordell Williams

10/13/2021

Predicting the Success of Rocket Launches

Background: There is a saying that "Space Is Hard" and that is certainly not an understatement. In the following notebook we will be exploring the dataset "All Space Mission from 1957-2020" as well as some variables that go into a rocket launch. We will then determine the importance of each variable and then predict whether a space mission will be a success or a failure. The first rocket was launched into space just over 64 years ago, and since then there have been over 4,319 launches with an increasing rate every year. Close to 500 launches have resulted in failure. It is imperative that governments and corporations can mitigate as many risks as possible when deciding when, where, and how to launch a rocket. In this work we will be using machine learning to create a model that predicts if a rocket mission will be a success or a failure based on attributes such as the country that launched the rocket, the location of the launch, when the rocket launched, as well as the budget of the mission. Within the dataset our target feature is the Status Mission, where we will set 0 as a mission failure and 1 as a mission success. We anticipate to find that most failed rocket launches will have the following characteristics: * They will launch outside of the traditional summer months (June-August) * They will be launched by smaller government agencies with smaller budgets * That there will be specific rocket engines that consistently result in failure.

Dataset: Next Space Flights, All Space Missions from 1957 - 2020 (https://nextspaceflight.com/launches/past/?page=1 (https://nextspaceflight.com/launches/past/?page=1))

Objective: Tune a machien learning model that can accurately predict the classification labels "Mission Success" or "Mission Failure" for rocket launches.

Import Libraries

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
library(tidyselect)
library(stringr)
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(ggplot2)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-2
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
library(readxl)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(ggplot2)
library(ggthemes)
library(mltools)
```

```
##
## Attaching package: 'mltools'
```

```
## The following object is masked from 'package:tidyr':
##
##     replace_na
```

```
library(janitor)
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
#Calling the dataframe with name df
df <- fread("C:/Users/corde/OneDrive/Documents/Space_Corrected.csv")
```

Dataset Overview

The dataset contains 7 columns and 4319 observation counts. It has various data types including strings, datetime stamps, and floats. The 7 columns are company, location, datum, detail, status rocket, rocket (budget), and status mission. Status mission, the predictive column of our dataset indicates whether a rocket was launched either as a “success” or “failure” (prelaunch failure and partial failure were grouped with ‘failure’). Out of the 4319 observation counts, failure accounts for approximately 11% of the rocket launches.

After loading the dataset in R and plotting the datasets, we identified significant differences between the attributes. For example, certain companies and countries have more launches than others. To further explore how different attributes could affect the success rate of a rocket launch, dummy variables will be created for the categorical columns such as location, rocket name, and datum. These dummy variables will then be incorporated into the regression models.

```
df[,"Unnamed: 0":=NULL]
df[,"V1":=NULL]
summary(df)
```

```
##  Company Name         Location            Datum              Detail
##  Length:4324        Length:4324        Length:4324        Length:4324
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##  Status Rocket         Rocket          Status Mission
##  Length:4324        Length:4324        Length:4324
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
```

```
unique_vals_count <- lapply(df, function(x) length(unique(x)))
unique_vals_count
```

```
## $`Company Name`
## [1] 56
##
## $Location
## [1] 137
##
## $Datum
## [1] 4319
##
## $Detail
## [1] 4278
##
## $`Status Rocket`
## [1] 2
##
## $Rocket
## [1] 57
##
## $`Status Mission`
## [1] 4
```

Data Cleaning and Preprocessing

The rocket launch data contains mostly categorical variables that need to be split, cleaned, and encoded. We used different methods to process the data depending upon the type of variable and it's number of levels.

The "Detail" column is a string that includes the type of rocket and mission name. We separated these into two separate columns, and then cleaned any extraneous punctuation marks. The "Company Name" column contained nearly 60 unique companies. We cleaned extraneous punctuation from this variable to clean it and prepare for encoding. The "Location" column contained information including the name of the launch site, state/province, and country. This column was split and cleaned of punctuation. Given that the text sometimes had inconsistent patterns, this field took additional work and some brute force to complete processing. The "Datum" column includes date and time information that was split into individual parts for modeling.

```
#Clean up the Detail column that contains the rocket name and mission name separated by a '|'
df$Rocketname_clean <- sapply(strsplit(as.character(df$Detail),'|',2), "[",1)
df$Missionname_clean <- sapply(strsplit(as.character(df$Detail),'|',2), "[", 2)
#df$Detail <- NULL
df$Rocketname_clean <- tolower(df$Rocketname)
df$Missionname_clean <- tolower(df$Missionname)
df$Missionname_clean <- gsub("\\??", "", df$Missionname)
df$Missionname_clean <- gsub("\\†","",df$Missionname)
df$Rocketname_clean <- gsub("\\??", "", df$Rocketname)
df$Rocketname_clean <- gsub("\\†","",df$Rocketname)
df$Rocketname_clean <- str_squish(df$Rocketname_clean)
df$Missionname_clean <- str_squish(df$Missionname_clean)
```

```
#Clean up the Company Name column
unique_companies <- unique(df$'Company Name')
len_company_name <- length(unique(df$'Company Name'))

#aggregate_company_name <- aggregate(df$'Company Name',by=list(df$`Company Name`),FUN=length)
#aggregate_company_name <- aggregate_company_name[order(-aggregate_company_name$x),]

df$CompanyName_clean <- df$'Company Name'
df$CompanyName_clean <-gsub("\\??", "", df$CompanyName_clean)
df$CompanyName_clean <-gsub("\\†","",df$CompanyName_clean)
df$CompanyName_clean <- tolower(df$CompanyName_clean)
```

```
#Clean up the Location column

df1 <- df

#Separate Location columns
df1 <- separate(df, 'Location', paste('Location', 1:3, sep = ' '), sep=',', extra = 'merge')
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 23 rows [27, 36,
## 76, 134, 144, 219, 274, 378, 588, 925, 1030, 1080, 1152, 1162, 1341, 1400, 1801,
## 1929, 3490, 3530, ...].
```

```
colnames(df1)[2] <- 'Site.Code'
colnames(df1)[3] <- 'Site'
colnames(df1)[4] <- 'Location'

#change the strings to lowercase
df1$Site <- tolower(df1$Site)
df1$Location <- tolower(df1$Location)
df1$Site.Code <- tolower(df1$Site.Code)

df1$Site <-gsub("?â€º", "a", df1$Site)

df1$Site <- str_squish(df1$Site)
df1$Site.Code <- str_squish(df1$Site.Code)
df1$Location <- str_squish(df1$Location)

#Clean up NAs in the Location column due to missing site codes
missing_location_code <- unique(df1[is.na(df1$Location)]$Site)
missing_location_code <- missing_location_code[missing_location_code != "shahrud missile test site"]
df1$Location[is.na(df1$Location)] <- df1$Site[is.na(df1$Location)]
df1$Location[df1$Location == "shahrud missile test site"] <- "iran"
df1$Site[df1$Site %in% missing_location_code] <- df1$Site.Code[df1$Site %in% missing_location_code]
```

```
#Clean up the Date and Rocket Cost columns
space <- df1

space$DateTime <- as.POSIXct(space$Datum, format="%a %b %d, %Y %H:%M", tz="UTC")
space$Launch_Year <- format(space$DateTime, format="%Y") #year(space$DateTime)
space$Launch_Month <- format(space$DateTime, format="%m") #month(space$DateTime)
space$Launch_DayOfMonth <- format(space$DateTime, format="%d")#mday(space$DateTime)
space$Launch_WeekDay <- format(space$DateTime, format="%a") #wday(space$DateTime)
space$Launch_Hour <- format(space$DateTime, format="%H") #hour(space$DateTime)
space$Launch_Minutes <- format(space$DateTime, format="%M") #minute(space$DateTime)
space$Rocket_Cost <- as.integer(trimws(space$Rocket))
```

```
## Warning: NAs introduced by coercion
```

```
space$Rocket_Cost[is.na(space$Rocket_Cost)] <- mean(space$Rocket_Cost, na.rm=TRUE)

#Clean up the Mission Status column
space$MissionStatus <- ifelse(space[["Status Mission"]] == "Success", 1, 0)
```

```
#Clean up the rocket status column
Space_Corrected <- space

Space_Corrected$ActiveRocket <- ifelse(Space_Corrected$'Status Rocket' == 'StatusActive', 1, 0)

active_rocket<- aggregate(Space_Corrected$'Status Rocket',by=list(Space_Corrected$'Status Rocket'),FUN=length)
```

Finalize our cleaned data set

```
Space_Final <- Space_Corrected %>% select(MissionStatus, ActiveRocket,
                                      Launch_Year, Launch_Month,
                                      Launch_DayOfMonth, Launch_Hour,
                                      Launch_Minutes, Rocketname_clean,
                                      CompanyName_clean, Location,
                                      Site, Rocket_Cost)
```

One-Hot Encoding

Majority of the variables in our dataset are categorical variables which generally must be encoded before running machine-learning models. The typical method for encoding categorical variables is one-hot encoding where you apply a binary 1:0 to indicate true or false for a specific category. Due to the high cardinality in some of our variables such as Company Name (which has over 300 unique values), the one-hot encoding process will create an addition 600 variables in our dataset.

```
#Categorical variables that need to be mean encoded
categorical_variables <- c('Launch_Year', 'Launch_Month', 'Launch_DayOfMonth', 'Launch_Hour', 'Launch_Minutes',
'Rocketname_clean', 'CompanyName_clean', 'Location', 'Site')

Space_Final$Rocketname_clean <- as.factor(Space_Final$Rocketname_clean)
Space_Final$CompanyName_clean <- as.factor(Space_Final$CompanyName_clean)
Space_Final$Location <- as.factor(Space_Final$Location)
Space_Final$Launch_DayOfMonth <- as.factor(Space_Final$Launch_DayOfMonth)
Space_Final$Launch_Year <- as.factor(Space_Final$Launch_Year)
Space_Final$Launch_Month <- as.factor(Space_Final$Launch_Month)
Space_Final$Launch_Hour <- as.factor(Space_Final$Launch_Hour)
Space_Final$Site <- as.factor(Space_Final$Site)

Space_Final_encoded <- one_hot(Space_Final, cols = categorical_variables, sparsifyNAs = TRUE)

Space_Final_encoded <- Space_Final_encoded %>%
                          clean_names()
```

Train-Test Split

We will split our dataset into a train and test set, using 50% for training data and 50% for testing data.The reason for our high data set split is due to the high variability in the special outliers of our complete population dataset. Due to the low failures, and the high number of variables, 50% train-test split will allow our training and test data set to have equal chances of having spcecial outliers.

```
library(caret)
```

```
## Loading required package: lattice
```

```
set.seed(12231988)

y_data <- Space_Final_encoded$mission_status
trainIndex <- createDataPartition(y_data, times = 1, list = FALSE, p = 0.5)

Space_Final_training <- Space_Final_encoded[trainIndex, ]
Space_Final_testing <- Space_Final_encoded[-trainIndex, ]

X_train <- data.matrix(select(Space_Final_training,
                          -c('mission_status','launch_year_1957','launch_month_01',
                             'launch_day_of_month_01',
                             'launch_hour_00','rocketname_clean_angara_1_2',
                             'company_name_clean_aeb','location_alaska_usa',
                             'site_baikonur_cosmodrome')))
y_train <- y_data[trainIndex]

X_test <- data.matrix(select(Space_Final_testing,
                          -c('mission_status','launch_year_1957','launch_month_01',
                             'launch_day_of_month_01',
                             'launch_hour_00','rocketname_clean_angara_1_2',
                             'company_name_clean_aeb','location_alaska_usa',
                             'site_baikonur_cosmodrome')))
y_test <- y_data[-trainIndex]
```

Modeling

We first started with linear regression to check for correlation between the factors which would affect the success rate of the rocket. In this process, we learned that there is a correlation between the factors and the success rate of the rockets. However, the correlation may either be positive or negative, the positive would be the launch day of the month and launch minutes and even the rocket name while other factors such as site, location, and company name are negative. The model also showed the intercept to be -0.357532 and MSE to be 0.2503 To learn more about our dataset, we decided to use lasso regression and got MSE train as 0.06593034, MSE test as 0.07178347, alpha was set to 1 and nlambda was 250, Since the MSE train is lower than the test, the model is overfitted. This was something we expected as the number of successful rocket launches was over 90% of the data we had. In order to learn more about the dataset, we had also decided to do ridge regression and in this process, we found that the MSE minimum was 0.06691877, MSE maximum was 0.07225838, when alpha is 0 and nlambda was set to 250 and Lambda was 162.716 We had also decided to work on a Random Tree Model, we found that the OOB estimate error rate was 8.42% and that the model failed to predict 67% of failed launches while it was great at predicting successful launches. We have also attached the confusion matrix to show the results

Confusion matrix: 0 1 class.error 0 105 221 0.67791411 1 52 2865 0.01782653

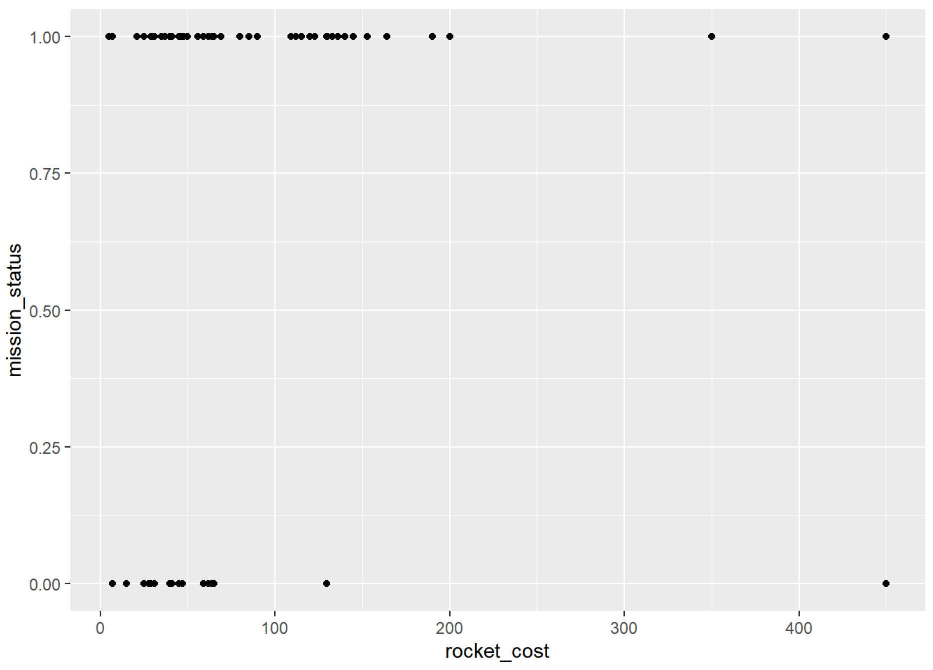
Model Preparation

```
f1 <- as.formula(mission_status ~ .)

x1.train.sample <- model.matrix(f1, Space_Final_training)
```

Linear Regression Model

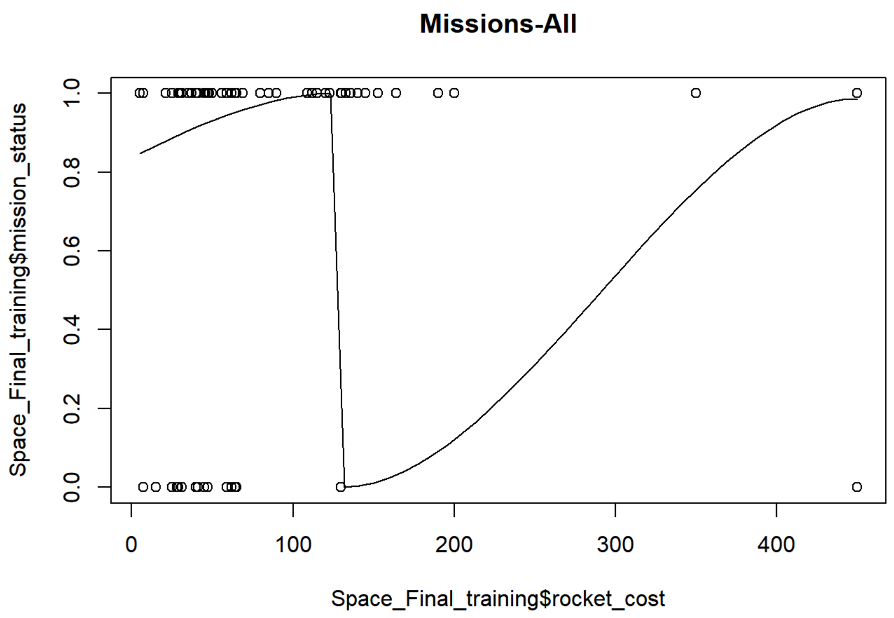
Model 1

```
linearMod_v <- lm(mission_status ~., data=Space_Final_training)

#summary(linearMod_v)

ggplot(Space_Final_training, aes(rocket_cost, mission_status)) +
  geom_point()
```



```
scatter.smooth(x=Space_Final_training$rocket_cost,y=Space_Final_training$mission_status,main="Missions-All")
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger
```



Model 2

```
linearMod_z <- lm( f1, data=Space_Final_training)

distPred <- predict(linearMod_z, Space_Final_testing)
```

```
## Warning in predict.lm(linearMod_z, Space_Final_testing): prediction from a rank-
## deficient fit may be misleading
```

```
#summary (linearMod_z)
#AIC (linearMod_z)

actuals_preds <- data.frame(cbind(actuals=Space_Final_testing$mission_status, predicteds=distPred))
correlation_accuracy <- cor(actuals_preds)

#head(actuals_preds)

min_max_accuracy <- mean(apply(actuals_preds, 1, min) / apply(actuals_preds, 1, max))
mape <- mean(abs((actuals_preds$predicteds - actuals_preds$actuals))/actuals_preds$actuals)
```

Model 3

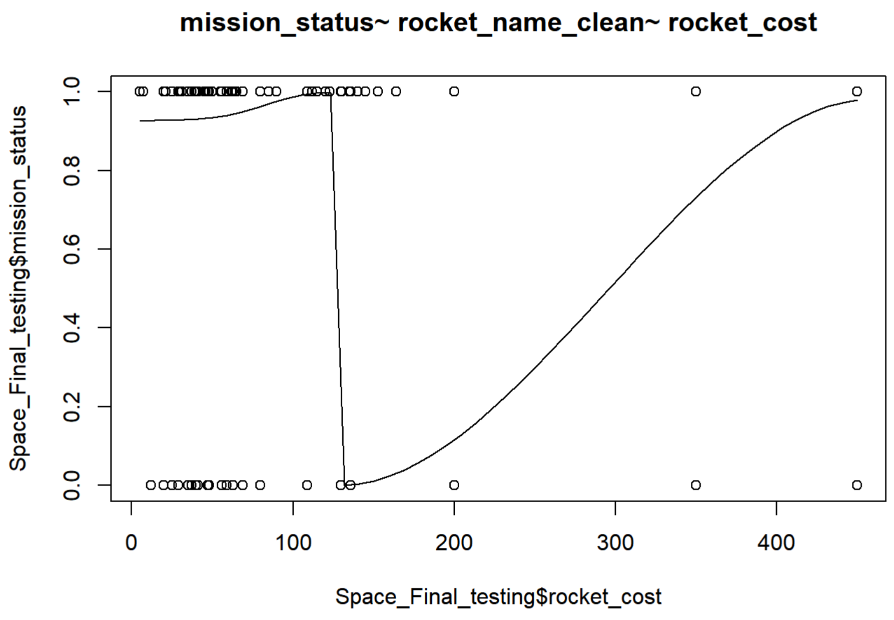
```
linearMod_z1 <- lm( f1,
                    data = Space_Final_training)

distPred1 <- predict(linearMod_z1, Space_Final_testing)
```

```
## Warning in predict.lm(linearMod_z1, Space_Final_testing): prediction from a
## rank-deficient fit may be misleading
```

```
#summary (linearMod_z1)
#AIC (linearMod_z1)

actuals_preds1 <- data.frame(cbind(actuals=Space_Final_testing$mission_status,
                                   predicteds=distPred1))
correlation_accuracy1 <- cor(actuals_preds1)

#head(actuals_preds1)

min_max_accuracy <- mean(apply(actuals_preds, 1, min) / apply(actuals_preds, 1, max))
mape <- mean(abs((actuals_preds1$predicteds - actuals_preds1$actuals))/actuals_preds1$actuals)

scatter.smooth(x=Space_Final_testing$rocket_cost, y=Space_Final_testing$mission_status,
               main="mission_status~ rocket_name_clean~ rocket_cost")
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = FALSE, :
## zero-width neighborhood. make span bigger
```



Ridge Regression Model

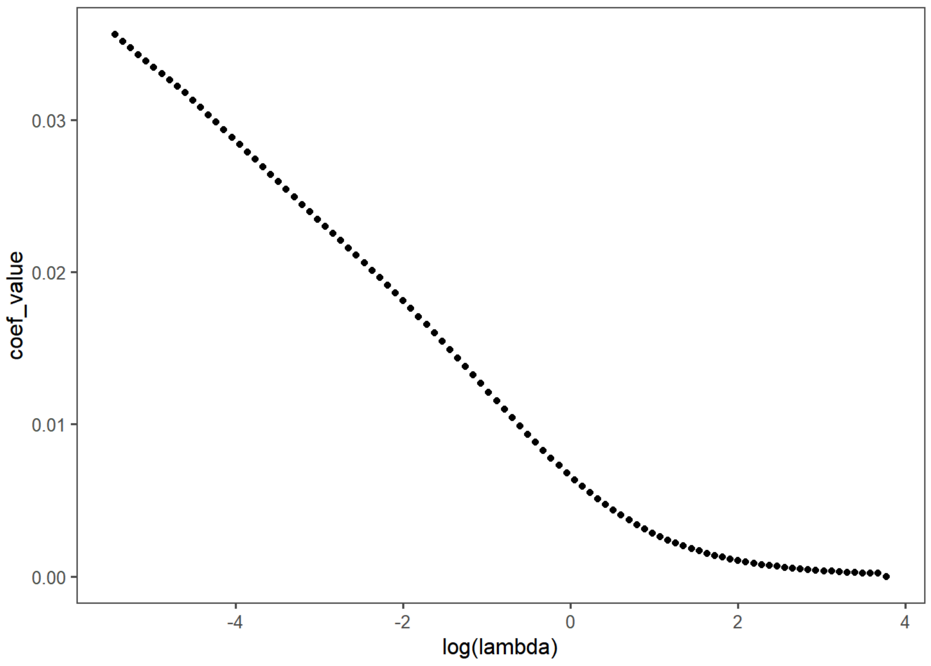
Model 1

Next, we'll try a simple ridge regression model where we test 100 lambdas across 5 folds to minimize test MSE.

```
ridge_fit_v <- cv.glmnet(X_train, y_train, alpha = 0, nlambda = 100, nfolds = 10)
ridge_fit_v
```

```
##
## Call:  cv.glmnet(x = X_train, y = y_train, nfolds = 10, alpha = 0, nlambda = 100)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure       SE Nonzero
## min  0.724    45 0.08071 0.004703     531
## 1se 11.797    15 0.08537 0.004469     531
```

```
ridge_coef_v <- predict(ridge_fit_v,
                     type = "coefficients",
                     s = ridge_fit_v$lambda)

to_plot_v <- data.table(
lambda = ridge_fit_v$lambda,
coef_value = ridge_coef_v[2, ]
)

ggplot(to_plot_v, aes(log(lambda), coef_value)) +
geom_point() +
theme_few()
```

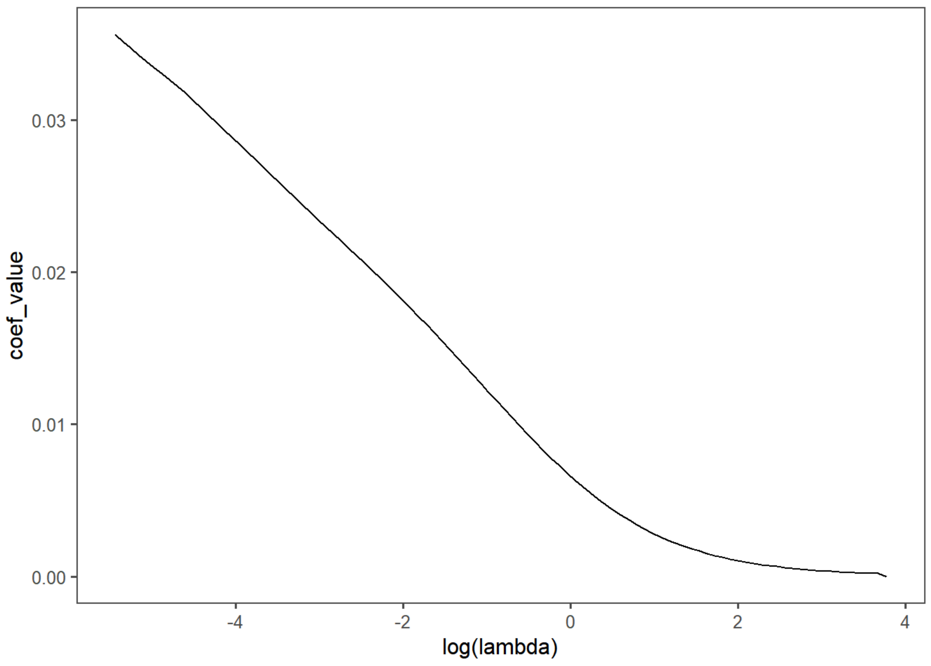


Model 2

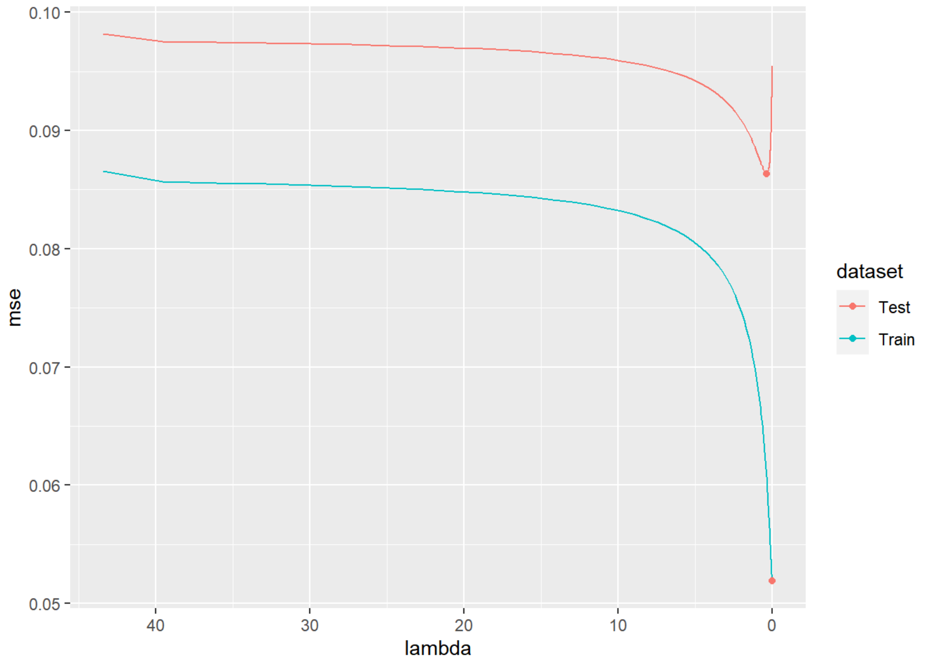
```
ridge_fit_r <- cv.glmnet(X_train, y_train, alpha = 0, nlambda = 100, nfolds = 5)
ridge_fit_r
```

```
##
## Call:  cv.glmnet(x = X_train, y = y_train, nfolds = 5, alpha = 0, nlambda = 100)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure       SE Nonzero
## min   0.72    45 0.08152 0.005438     531
## 1se  43.39     1 0.08652 0.004914     531
```

```
ridge_coef_r <- predict(ridge_fit_r,
                      type = "coefficients",
                      s = ridge_fit_r$lambda)

to_plot <- data.table(
lambda = ridge_fit_r$lambda,
coef_value = ridge_coef_r[2, ]
)

ggplot(to_plot, aes(log(lambda), coef_value)) +
geom_line() +
theme_few()
```



```
yhat_train_ridge_r <- predict(ridge_fit_r, X_train, s = ridge_fit_r$lambda)
mse_train_ridge_r <- colMeans((y_train - yhat_train_ridge_r)**2)

yhat_test_ridge_r <- predict(ridge_fit_r, X_test, s = ridge_fit_r$lambda)
mse_test_ridge_r <- colMeans((y_test - yhat_test_ridge_r)**2)

ridge_lambda_min_mse_train_r <- ridge_fit_r$lambda[which.min(mse_train_ridge_r)]
ridge_lambda_min_mse_test_r <- ridge_fit_r$lambda[which.min(mse_test_ridge_r)]

ridge_mse_min_test_r <- mse_test_ridge_r[which.min(mse_test_ridge_r)]
ridge_mse_min_train_r <- mse_train_ridge_r[which.min(mse_train_ridge_r)]

r_dd_mse <- data.table(lambda = ridge_fit_r$lambda,mse = mse_train_ridge_r, dataset = 'Train')
#print(head(r_dd_mse,5))

r_dd_mse <- rbind(r_dd_mse, data.table(lambda = ridge_fit_r$lambda,mse = mse_test_ridge_r,
dataset ="Test"))
#print(tail(r_dd_mse,5))

ridge_plot_mse_r <- ggplot(r_dd_mse, aes( x = lambda, y = mse, color= dataset)) +
  geom_line() +
  geom_point(aes(x=ridge_lambda_min_mse_train_r, y = ridge_mse_min_train_r)) +
  geom_point(aes(x=ridge_lambda_min_mse_test_r, y= ridge_mse_min_test_r)) +
  scale_x_reverse()
#Print plot
print(ridge_plot_mse_r)
```

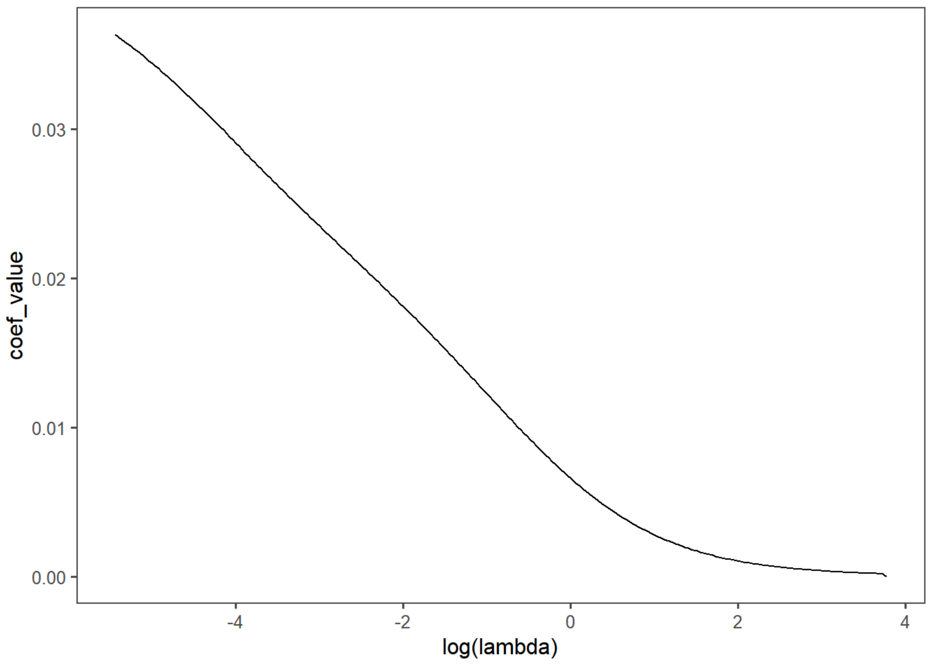


Model 3

```
ridge_fit_c <- cv.glmnet(X_train, y_train, alpha = 0, nlambda = 250, nfolds = 10)
ridge_fit_c
```

```
##
## Call:  cv.glmnet(x = X_train, y = y_train, nfolds = 10, alpha = 0, nlambda = 250)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure       SE Nonzero
## min   0.71   112 0.08076 0.005797     531
## 1se  43.39     1 0.08654 0.006145     531
```

```
ridge_coef_c <- predict(ridge_fit_c,
                      type = "coefficients",
                      s = ridge_fit_c$lambda)

to_plot_c <- data.table(
lambda = ridge_fit_c$lambda,
coef_value = ridge_coef_c[2, ]
)
ggplot(to_plot_c, aes(log(lambda), coef_value)) +
geom_line() +
theme_few()
```



```
yhat_train_ridge_c <- predict(ridge_fit_c, X_train)
mse_train_ridge_c <- colMeans((y_train - yhat_train_ridge_c)**2)

yhat_test_ridge_c <- predict(ridge_fit_c, X_train)
mse_test_ridge_c <- colMeans((y_test - yhat_test_ridge_c)**2)

lambda_min_mse_train_c <- ridge_fit_c$lambda[which.min(mse_train_ridge_c)]
lambda_min_mse_test_c <- ridge_fit_c$lambda[which.min(mse_test_ridge_c)]
```

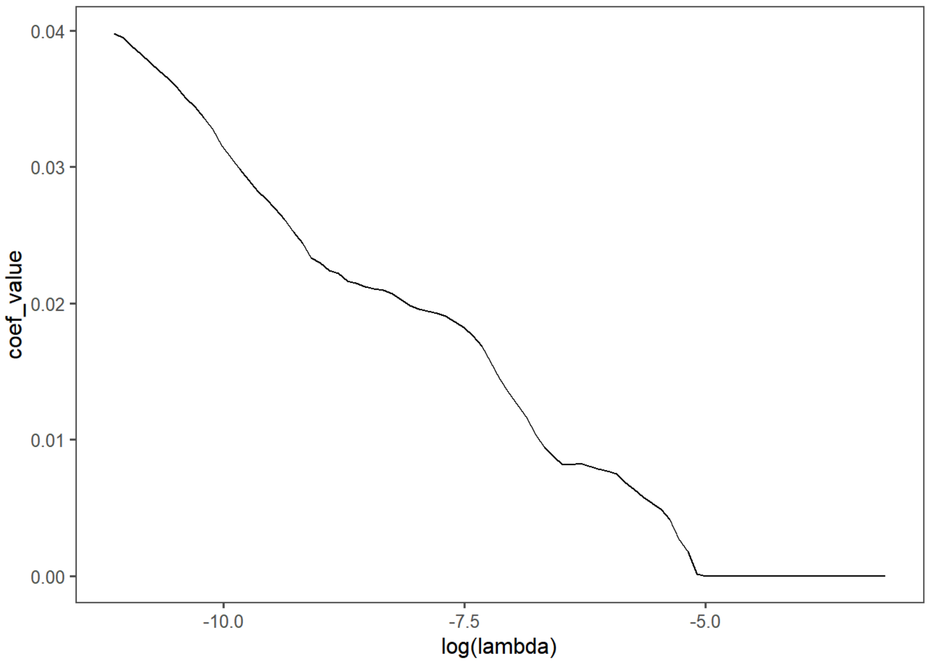
Lasso Regression Model

Model 1

```
#lambdas <-
#
lasso_fit_r <- cv.glmnet(X_train, y_train, alpha = 1, nlambda = 100, nfolds = 10)
lasso_fit_r
```

```
##
## Call:  cv.glmnet(x = X_train, y = y_train, nfolds = 10, alpha = 1, nlambda = 100)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure       SE Nonzero
## min 0.01075    16 0.08062 0.005567      86
## 1se 0.03283     4 0.08597 0.005563       4
```

```
lasso_coef_r <- predict(lasso_fit_r,
                      type = "coefficients",
                      s = lasso_fit_r$lambda)

yhat_train_lasso_r <- predict(lasso_fit_r, X_train, s = lasso_fit_r$lambda)
yhat_test_lasso_r <- predict(lasso_fit_r, X_test, s = lasso_fit_r$lambda)

mse_train_lasso_r <- colMeans((y_train - yhat_train_lasso_r)^2)
mse_test_lasso_r <- colMeans((y_test - yhat_test_lasso_r)^2)

lasso_lambda_min_mse_train_r <- lasso_fit_r$lambda[which.min(mse_train_lasso_r)]
lasso_lambda_min_mse_test_r <- lasso_fit_r$lambda[which.min(mse_test_lasso_r)]

lasso_mse_min_train_r <- mse_train_lasso_r[which.min(mse_train_lasso_r)]
lasso_mse_min_test_r <- mse_test_lasso_r[which.min(mse_test_lasso_r)]

to_plot_r <- data.table(
lambda = lasso_fit_r$lambda,
coef_value = lasso_coef_r[2, ]
)
ggplot(to_plot_r, aes(log(lambda), coef_value)) +
geom_line() +
theme_few()
```



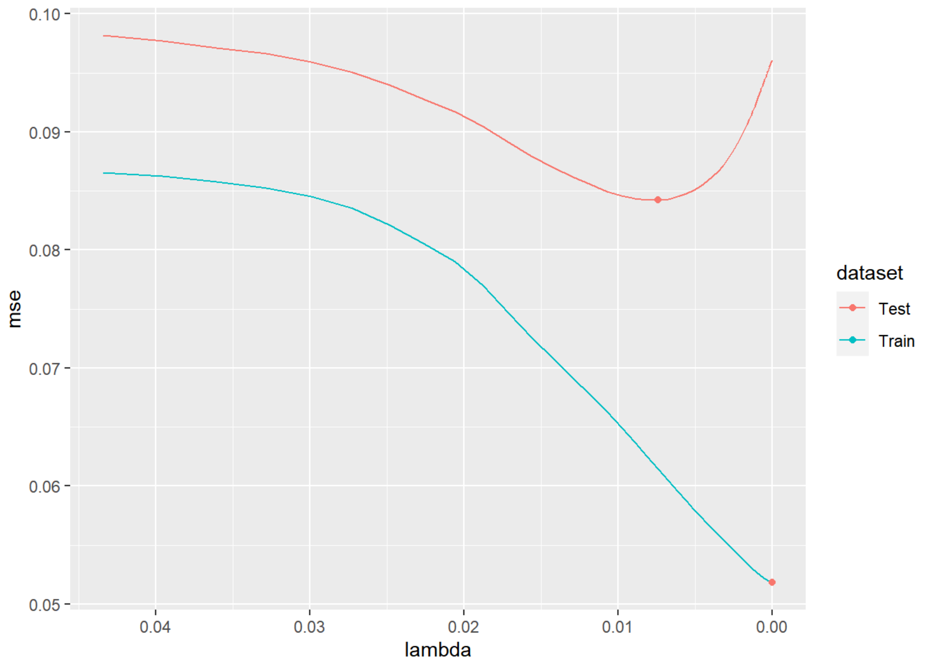
```
dd_mse_r <- data.table(lambda = lasso_fit_r$lambda,mse = mse_train_lasso_r, dataset = 'Train')
print(head(dd_mse_r,5))
```

```
##        lambda        mse dataset
## 1: 0.04339475 0.08657764   Train
## 2: 0.03953968 0.08625792   Train
## 3: 0.03602709 0.08577406   Train
## 4: 0.03282654 0.08521800   Train
## 5: 0.02991032 0.08453482   Train
```

```
dd_mse_r <- rbind(dd_mse_r, data.table(lambda = lasso_fit_r$lambda, mse = mse_test_lasso_r,
dataset ="Test"))
print(tail(dd_mse_r,5))
```

```
##          lambda        mse dataset
## 1: 2.110114e-05 0.09604418    Test
## 2: 1.922657e-05 0.09604918    Test
## 3: 1.751854e-05 0.09605356    Test
## 4: 1.596224e-05 0.09605723    Test
## 5: 1.454420e-05 0.09606121    Test
```

```
lasso_plot_mse_r <- ggplot(dd_mse_r, aes( x = lambda, y = mse, color= dataset)) +
  geom_line() +
  geom_point(aes(x=lasso_lambda_min_mse_train_r, y = lasso_mse_min_train_r)) +
  geom_point(aes(x=lasso_lambda_min_mse_test_r, y= lasso_mse_min_test_r)) +
  scale_x_reverse()
#Print plot
print(lasso_plot_mse_r)
```



```
print(lasso_mse_min_test_r)
```

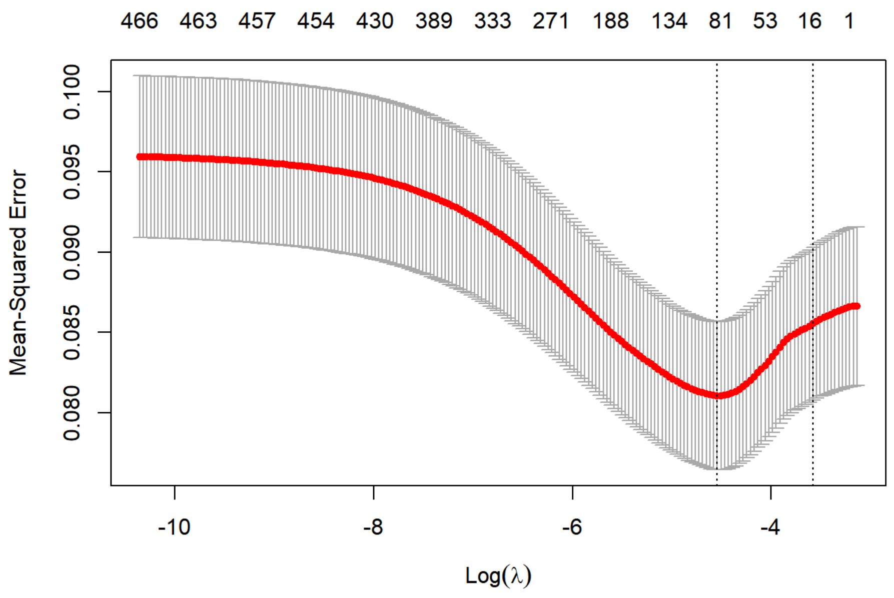
```
##        s20
## 0.08422565
```

Model 2

```
#Lasso Regression using train/test data from above
library(glmnet)
lasso_fit_b <- cv.glmnet(X_train, y_train, alpha = 1, nlambda = 250, nfolds = 10)
best_lam_lasso <- lasso_fit_b$lambda.min
print(best_lam_lasso)
```

```
## [1] 0.01064132
```

```
#Predicting the responses and assigning the train and test responses to variables
y_train_hat_b <- predict(lasso_fit_b ,newx = X_train)
y_test_hat_b <- predict(lasso_fit_b, newx = X_test)

#A matrix of 100 MSE trains and tests based in the prediction vs actual
mse_train_b <- colMeans((y_train_hat_b - y_train) ^ 2)
mse_test_b <- colMeans((y_test_hat_b - y_test) ^ 2)

#the lambda with the lowest mse_train and mse_test
min_mse_train_b <- mse_train_b[which.min(mse_train_b)]
min_mse_test_b <- mse_test_b[which.min(mse_test_b)]

#creating data tables which will be used to plot the Lasso regression
df_mse_b <- data.table(
lambda = lasso_fit_b$lambda,
mse = mse_train_b,
dataset = "Train"
)

df_mse_b <- rbind(df_mse_b, data.table(
lambda = lasso_fit_b$lambda,
mse = mse_test_b,
dataset = "Test"
))

plot(lasso_fit_b)
```



```
print(df_mse_b)
```

```
##            lambda        mse dataset
##   1: 4.339475e-02 0.08384336   Train
##   2: 4.181893e-02 0.08384336   Train
##   3: 4.030034e-02 0.08384336   Train
##   4: 3.883689e-02 0.08384336   Train
##   5: 3.742658e-02 0.08384336   Train
##  ---
## 388: 3.708209e-05 0.09528097    Test
## 389: 3.573551e-05 0.09528097    Test
## 390: 3.443783e-05 0.09528097    Test
## 391: 3.318727e-05 0.09528097    Test
## 392: 3.198212e-05 0.09528097    Test
```

```
print(min_mse_test_b)
```

```
## lambda.1se
## 0.09528097
```

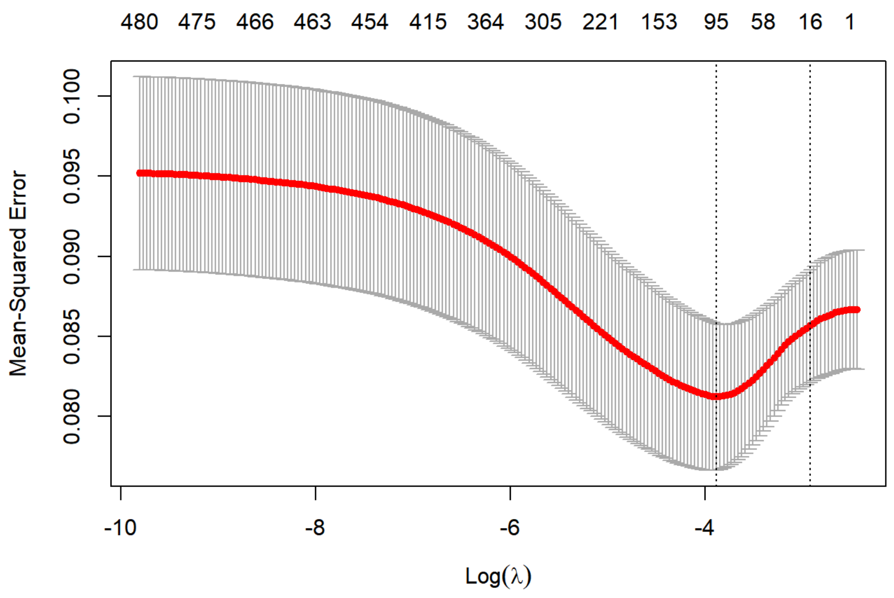
Elastic Net Model

The elastic net model combines many of the features of the ridge and lasso regression model. Let’s compare how this estimates the loss function.

```
elastic1 <- cv.glmnet(X_train,y_train,alpha=0.5,nfold=10,nlambda=250)
best_lam <- elastic1$lambda.min
print(best_lam)
```

```
## [1] 0.02050979
```

```
#Predicting the responses and assigning the train and test responses to variables
y_train_hat_a <- predict(elastic1 ,newx = X_train)
y_test_hat_a <- predict(elastic1, newx = X_test)
#y_test_hat_a

#A matrix of 100 MSE trains and tests based in the prediction vs actual
mse_train_a <- colMeans((y_train_hat_a - y_train) ^ 2)
mse_test_a <- colMeans((y_test_hat_a - y_test) ^ 2)

#the lambda with the lowest mse_train and mse_test
min_mse_train_a <- mse_train_a[which.min(mse_train_a)]
min_mse_test_a <- mse_test_a[which.min(mse_test_a)]

#creating data tables which will be used to plot the Lasso regression
df_mse_a <- data.table(
lambda = elastic1$lambda,
mse = mse_train_a,
dataset = "Train"
)

df_mse_a <- rbind(df_mse_a, data.table(
lambda = elastic1$lambda,
mse = mse_test_a,
dataset = "Test"
))

plot(elastic1)
```



```
#print(df_mse_a)
print(min_mse_test_a)
```

```
## lambda.1se
##  0.0950144
```

Random Forest Tree Model

Model 1

How to interpret the RandomForest numbers:

-> We ran a classification RandomForest on the data using MissionStatus as the target variable -> Used 500 trees to run the regression -> 3 variables were considered at each internal node -> Classification trees have a default setting of sqrt(p) for setting m -> OOB error estimate says that 91.47% of OOB samples were correctly classified by random forest

-> Lastly is the confusion matrix: -> There were 139 launches that were correctly labeled failures -> There were 63 launches that were incorrectly labeled failures -> There were 308 launches that were incorrectly labeled successes (yikes) -> There were 3816 launches that were correctly labeled successes

In the following code trying to see if 500 trees is enough.

```
library(randomForest)
est <- randomForest(mission_status ~., data = Space_Final_training,
                    do.trace=F, proximity=TRUE)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
est
```

```
##
## Call:
##  randomForest(formula = mission_status ~ ., data = Space_Final_training,      do.trace = F, proximity = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 203
##
##           Mean of squared residuals: 0.08616993
##                     % Var explained: 0.47
```

Model 2

Next, we'll try to run our random forest using mtry = 2.

```
model <- randomForest(mission_status ~.,
                    data=Space_Final_training, do.trace=F, proximity=TRUE,mtry=2,)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
model
```

```
##
## Call:
##  randomForest(formula = mission_status ~ ., data = Space_Final_training,      do.trace = F, proximity = TRUE,
mtry = 2, )
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 0.08459331
##                     % Var explained: 2.29
```

-> New confusion matrix (old numbers in parentheses): -> There were 140 (139) launches that were correctly labeled failures -> There were 52 (63) launches that were incorrectly labeled failures -> There were 305 (308) launches that were incorrectly labeled successes -> There were 3827 (3816) launches that were correctly labeled successes

Model 3

How does the model perform with mtry = 4?

```
library(randomForest)
library(rpart)
library(rpart.plot)

rf_model_c <- randomForest(mission_status ~ ., Space_Final_training, do.trace = F, mtry = 4)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

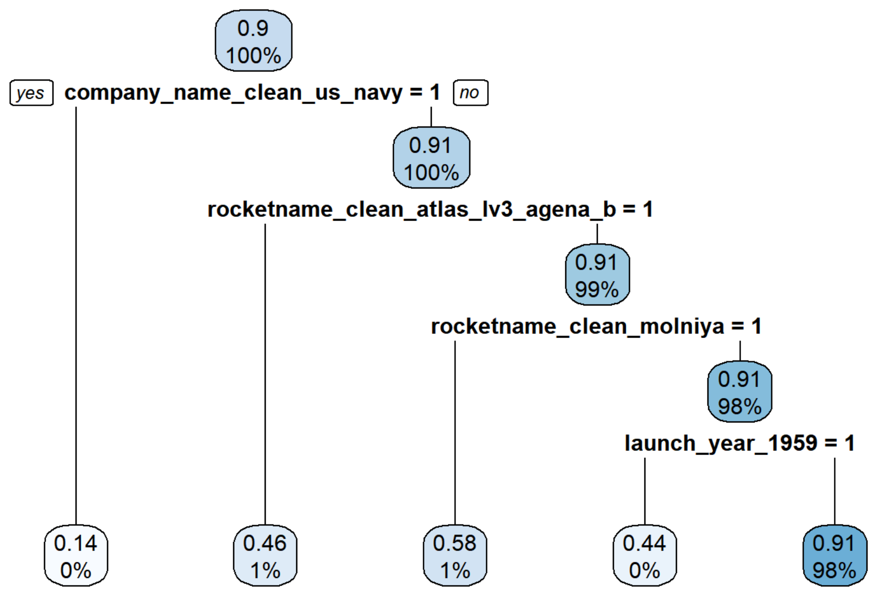
```
yhat_train_rand_c <- predict(rf_model_c, Space_Final_training)
mse_train_rand_c <- mean((y_train - yhat_train_rand_c)**2)
#confusionMatrix(y_train, yhat_train_rand)

yhat_test_rand_c <- predict(rf_model_c, Space_Final_testing)
mse_test_rand_c <- mean((y_test - yhat_test_rand_c)**2)

print(rf_model_c)
```

```
##
## Call:
##  randomForest(formula = mission_status ~ ., data = Space_Final_training,      do.trace = F, mtry = 4)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.08076548
##                     % Var explained: 6.71
```

```
print(paste("MSE_train: ", mse_train_rand_c, "MSE_test: ", mse_test_rand_c))
```

```
## [1] "MSE_train:  0.0717191646958217 MSE_test:  0.0899771684076209"
```

```
rpart_obj <- rpart(mission_status ~., Space_Final_training)
rpart.plot(rpart_obj)
```


0.9
100%
yes
company_name_clean_us_navy = 1
no
0.91
100%
rocketname_clean_atlas_lv3_agena_b = 1
0.91
99%
rocketname_clean_molniya = 1
0.91
98%
launch_year_1959 = 1
0.14
0%
0.46
1%
0.58
1%
0.44
0%
0.91
98%