ASSIGNMENT 4

ENGIN 242 - Applications in Data Analytics

Aditya Peshin
Enrollment Number: 3035280249

Q1. Problem 1: Predicting Useful Questions on Stack Overflow (100 points)
Part (a). Start by cleaning up the dataset to get it into a form where we can apply statistical learning methods to predict our dependent variable of interest. Please briefly describe each step that you took to clean and process the data.

Ans. I started the data cleaning process by looking at the dataset to see what kind of data I was working with, and saw that there 3 columns of data, “Title” - which was a character variable without any HTML formatting, “Body” - a character variable which had some html tags that would pose a problem, and “Score” - an integer variable that indicated the usefulness of the question.

Then, I used conditional statements to convert the “Score” variable into a binary categorical variable, and converted into a factor type variable. I then dropped the original feature in order to ensure that it doesn’t affect the decision making of the models that I test.

After that, I converted both the body and title into Corpora. Following this, I removed the html characters from the body data, and found that it had failed to remove ‘\n’ (new line) from the body. Hence, I changed the letters of the words to lowercase and dropped all instances of “\n” from the body text, to remove noise, and followed that up with removing the punctuation.

I removed the instances of \n first, as if I did it after removing the punctuation, it would have been very difficult to remove that from the body(as the ‘\’ would have disappeared). Following that, I removed the stopwords, and stemmed the document, converting it into a document term matrix.

Part (b) Now split your processed data into a training set and a test set. Use 70% of the data as training data and be sure to split the data in such a way that keeps the relative amount of useful questions roughly the same in each set (we have used two functions in R for splitting data, only one of them is appropriate here). Use your analytics skills to build the best model that you can for predicting useful questions.

Report on the details of your training procedures and nal comparisons on the test set. Use your best judgment to choose a nal model and explain your choice. Use the bootstrap to assess the performance of your nal model in a way that properly reports on the variability of the relevant performance metrics (accuracy, TPR, and FPR).

Use the bootstrap to assess the performance of your nal model in a way that properly reports on the variability of the relevant performance metrics (accuracy, TPR, and FPR).

Ans. In order to find out the usefulness of the features extracted, and to avoid a lengthy computation, I placed the sparsity threshold as 0.93 for the title, and 0.70 for the body text, giving me 9 variables for the title, and 13 variables for the body. I selected more variables for the body because it had more words. I added a 'Title_' and 'Body_' to each column name, in order to indicate where the word came from.
The initial idea was to run logistic regression, LDA, random forests and boosting **quickly to get some base estimates** for the values of the accuracy possible with those methods. I split the data into training and test sets, and ended up with values of accuracy as:

1) Logistic Regression:
   - Accuracy = 0.5348
   - TPR = 0.4406
   - FPR = 0.3737
2) LDA:
   - Accuracy = 0.5352
   - TPR = 0.4415
   - FPR = 0.3737
3) Random Forests (without cross-validation):
   - Accuracy = 0.5205
   - TPR = 0.4832
   - FPR = 0.4432
4) Boosting :
   - Accuracy = 0.5258
   - TPR = 0.4179
   - FPR = 0.3693
5) Stepwise Logistic Regression :
   - Accuracy = 0.5343
   - TPR = 0.4415
   - FPR = 0.3755

These values were interesting, as they told me that the models are not good with these kinds of features, and that I need to select better features in order to improve predictions. In particular, when I looked at the Variable Importance Measure in Random forests, I found something peculiar:

```
> as.data.frame(mod_basicRF$importance) %>%
+   mutate(Words = rownames(mod_basicRF$importance)) %>%
+   arrange(desc(MeanDecreaseGini))
   MeanDecreaseGini         Words
1         225.21648     Body_plot
2         176.88990     Body_data
3         176.88285      Body_use
4         148.29314     Body_valu
5         138.85239      Body_tri
6         134.08803     Body_want
7         132.26684     Body_like
8         128.70055      Body_can
9         127.61999   Body_ggplot
10        126.34964     Body_code
11        115.34566      Body_get
12         67.63958 Title_ggplot2
13         62.64925    Title_plot
14         61.03789  Title_ggplot
15         52.72292     Title_use
16         46.25506    Title_line
17         46.09043     Title_bar
18         38.87081    Title_data
19         38.78477  Title_legend
20         35.67929   Title_label
> 
```

The interesting thing to note here was that all the Body variables had much higher 'MeanDecreaseGini' values than the Title variables, and this finding helped guide the next stage of my analysis. **I was also able to confirm that not removing the word "ggplot" from my document term matrix was the right decision**, as it gives a great contribution to the 'MeanDecreaseGini' in both the title and body.

I decided to **include more variables from the Body (around 35~40) and reduce the Title Variables to (around 5~7).** By reducing the sparsity to 0.92 for "Title", and increasing it to 0.835 for "Body", I found that this resulted in **5 "Title" and 40 "Body" Variables**. Combining the two document term matrices together, splitting the training and testing set, and training the models led to:

1) Logistic Regression:

Accuracy	= 0.5817
TPR	= 0.5013
FPR	= 0.3403

The logistic regression model shows an improvement of 0.047, and the TPR and FPR also show significantly improved values.

2) LDA:

Accuracy	= 0.5817
TPR	= 0.5013
FPR	= 0.3403

Here, the new LDA model behaved similar to the new Logistic Regression model, showing a significant improvement over its earlier counterpart.

3) Cross Validated Random Forests:

For this model, I chose to retain the original values of k = 5 folds, and ntrees = 500, as I believe that selecting mtry was more important. I decided to vary mtry between 1 and 20, as I know that it was highly unlikely that mtry would be a large value. Running this model, I found that my suspicions were right and I obtained the maximum value of accuracy at **mtry = 2**.

Accuracy	= 0.5718
TPR	= 0.4161
FPR	= 0.2770

This is one of the lowest values of FPR that we have observed during the entire training process.

4) Cross Validated Boosting Model:

In my original boosting model, the final value of n.trees selected by the model was 100, and the value of interaction depth was 1 and with learning rate as “0.1”. Keeping this in mind, I decided to keep the range of my values for these variables as n.trees = (1:2500, jumps of 100), interaction.depth = (1,2,4,8) and learning rate as “0.1”. Running the Boosting algorithm on these parameters, I got the cross validated selection of parameters as:

N.trees = 300, interaction.depth = 1.

Now, I knew the general ballpark in which the values of N.Trees and interaction depth lie, so I decided to get a better model by fine tuning my hyperparameter selection. In order to fine tune the search, I decided to run a second set of boosting, with the tuning parameters of n.trees as

(50,100,150, … 950,1000), interaction depth as (1,2,3) and **learning rate as (0.01)**. Doing this second set of modelling led to:

N.Trees = 650, interaction.depth = 2.

Doing this turned out to be beneficial, as the smaller learning rate lead to a better model at those parameter values.

Using this final model to make predictions on the test set, we get:

Accuracy = 0.5830
TPR = 0.4986
FPR = 0.3350

5) Stepwise Logistic Regression:

Running this model resulted in 20 coefficients being selected out of the original 45 variables. The results of this model are:

Accuracy = 0.5763
TPR = 0.4913
FPR = 0.3412

For this particular application, what is most important is **a mix of the accuracy and the FPR**. Ideally, if 2 models have similar values of accuracy, we would want the model with a lower FPR to be selected, so that **consistently high-quality questions are displayed**.

Model	Accuracy	TPR	FPR
Logistic Regression	0.5817	**0.5013**	0.3403
LDA	0.5825	0.5004	0.3377
CV Random Forest	**0.5718**	**0.4161**	**0.2770**
CV Boosting	**0.5830**	0.4986	0.3350
Stepwise Logistic Regression	0.5763	0.4913	**0.3412**

The best performers have been marked in green, and the worst performers in red. If we try to go for the lowest FPR, we get the worst accuracy and substantial drops in TPR. On the other hand, if we opt for the **highest accuracy model**, we get the **second best TPR and FPR**, which means that it is a good compromise, and should be our final model.

Thus, I will select the **CV Boosting model** as my final model. Note that I could have run the analysis with more variables, but could not due to computing constraints in the given time frame.

Performing the Bootstrap analysis, I calculated the performance metrics on 10,000 bootstrapped datasets. The results are shown below:

	Original	Bias	Standard Dev
Accuracy	0.5830	0.000121	0.010454
TPR	0.4986401	0.0001092529	0.01504682
FPR	0.3350923	-0.0001159890	0.0139658

The results of the analysis state that there is almost certainly no bias in the estimation of Accuracy. The 95% confidence intervals were also provided by the analysis, and they are as follows:

Accuracy		TPR		FPR	
Lower	Upper	Lower	Upper	Lower	Upper
0.5625	0.6036	0.4690	0.5278	0.3074	0.3629

This means we can say with 95% confidence that the values of Accuracy, TPR and FPR will stay within these ranges.

Part (c) Now, let us consider how Stack Overflow might use your model. In particular, consider the following scenario. When a user navigates to the page showing ggplot2 questions, they are automatically presented with the 15 most recently submitted questions, in order of most recent first (this is not necessarily true in reality, but let's pretend that it is).
Suppose further that Stack Overflow believes that most users are extremely impatient and will only pay attention to the single question at the very top of the page, and therefore they would like to **maximize the probability that the top question is useful**.

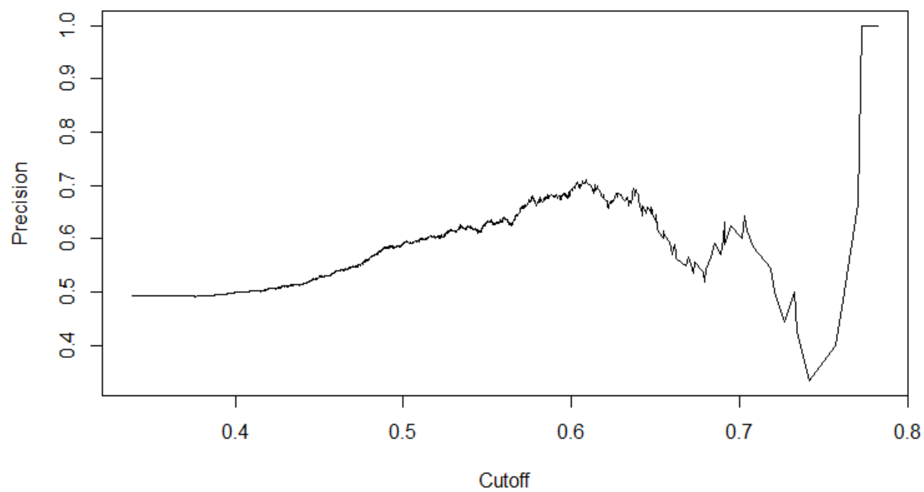
i) Think about how to select a model, among many different models, to best accomplish the goal of maximizing the probability that the top question is useful. Comment on the precise criteria that you would use (e.g., “I would select a model with the highest accuracy" or “I would select a model with the highest TPR", etc.) and note that your answer may involve multiple performance metrics if you wish. Explain your response.

Ans. Of the 15 most recent questions, we have to make sure that the one on the top is useful. The candidate questions for this spot will obviously be only the ones that the model identifies as useful, and hence a logical idea might be to maximize the ratio of True Positives to False Positives. In this scenario, the idea is that we don’t need to worry as much about the TPR, as there are 15 observations to choose from, and we will identify at least one TP. We are more concerned about the False Positives, as if the user sees a question that is not useful on the top, they won't stay on the webpage. However, the FPR is not the right metric as we need to minimize the FP while maximizing the TP.Thus, I will choose a threshold that maximises the value of precision, which is defined as:

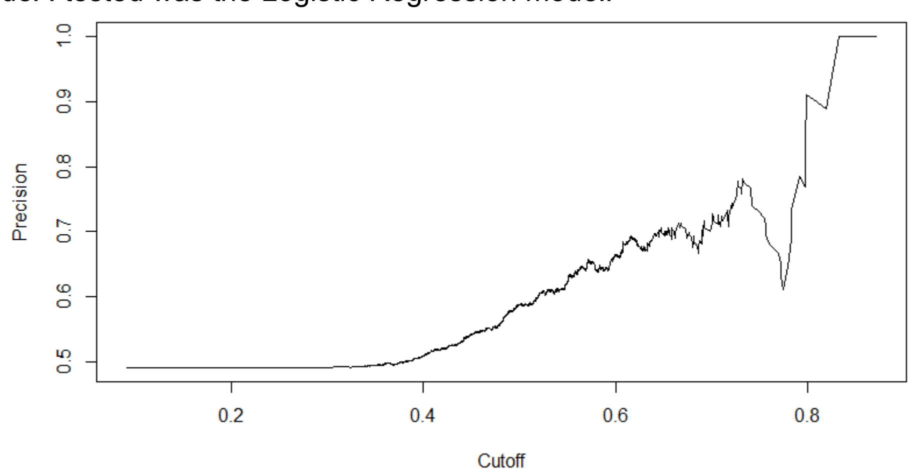
Precision = TP / (TP + FP)

ii) Revisiting the models that you have built in part (b), can you identify a specific model that best accomplishes the goal? How much does the model you selected improve upon Stack Overflow's current approach of showing the most recent posts first (described above)? In particular, use the results of your model on the test set to give a precise numerical estimate of the increase in the probability that the top question is useful.

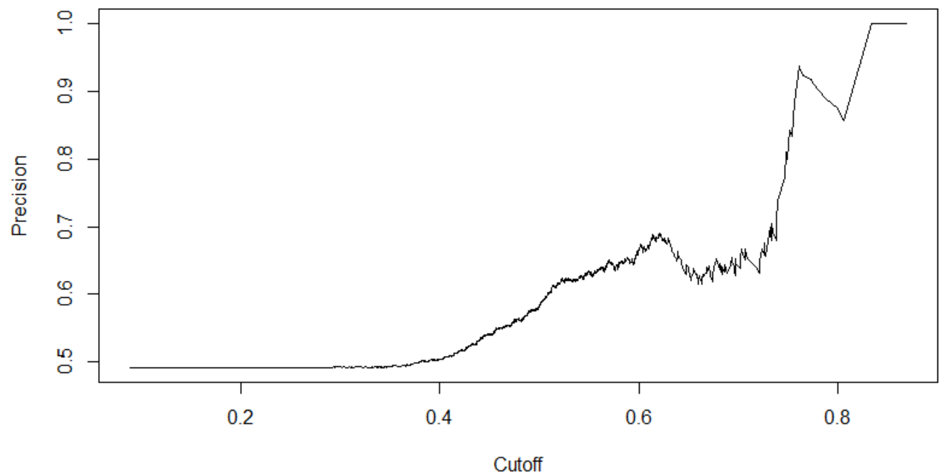
Ans. After calculating the precision of the Cross Validated Boosting Model over different values of cutoff, we get the following graph:



From the graph, we see that the Cross Validated Boosting has a maxima of performance ~0.7 at cutoff value ~ 0.6. There was a curious dip and rise in performance after that, and then a steep rise in performance when the number of positive predictions were small (which skews the ratio). The next model I tested was the Logistic Regression model.



From this graph, we can see that the logistic regression performs much better than the Cross Validated Boosting model, and has a precision of around 0.8 when the threshold is around 0.72. After this, I tried the stepwise logistic regression model. Trying the Stepwise Logistic Regression model, we find:



The stepwise logistic regression model shows some very interesting results. There is a local maxima for the precision value at p ~ 0.6, and another maxima just before p~0.8 . This is an interesting trend captured across all three graphs, where there is a maxima, then a dip, and another maxima before the performance value rises to reach 1.

In all three cases, I consider the first maxima, as it is a better representative of what will happen outside the sample. The second maxima may come when too few observations are being predicted as positive, and that's what is causing the erratic second maxima.

Due to time and resource constraints, I was unable to train my Cross Validated Random forests model on a modified version of the cost function, but since it was the worst performer, I do not expect much value from that model.

As it stands, the values of performance of the various models are (at optimal thresholds):

Model	Performance	Threshold
CV Boosting	0.69	0.4 (the direction of inequality is opposite)
Logistic Regression	**0.774**	0.735
Stepwise LR	0.6835	0.625
CV Random Forests	0.5930	Original (No changes to cost function)
LDA	0.5897	At default value

The range of the values for the thresholds were initially identified on the graph, and enumerated manually on R, in order to find the exact value of the threshold for best precision on the test set.

An interesting observation that I found while plotting and calculating values for the CVBoosting model is that since the direction of the inequality was the opposite, the plot of performance vs cutoff value was very irregular, until I flipped the inequality by supplying 1 minus the predicted probability, which gave the plot shown above.

Now, looking at the table, we can easily decide that the best model for our application will be the **Logistic Regression Model** as it has the highest precision compared to the other models.

Also, an important thing to note is the baseline performance. Now, out of the 15 observations that are the most recent, we can calculate the expected precision of the current algorithm by looking at the confusion matrix of the entire test set.

Since the test set has 1137 bad questions, and 1104 good questions, if we arranged them randomly, there is a chance that each one of the observations can be the top question. Thus, we can make a confusion matrix that is equivalent to a model that predicts all the observations as useful. The confusion matrix that we get is:

		Prediction	
		Not useful	Useful
Reality	Not useful	0	1137
	Useful	0	1104

Thus here the baseline has a precision of (1104/(1104+1137)) = 0.4926
In other words, the probability that the top question shown is useful is **49.26%**

In our model **(Logistic Regression)** on the other hand, only the questions that the model considers useful will be shown on the top. Out of the questions that the model considers useful, the probability that the question is actually useful (the precision) is **77.41%**

Thus, the increase in the probability that the top question is useful is = 77.41 - 49.26 = **28.15%** improvement. Note that this is the expected difference in the usefulness of the questions predicted, not the actual difference.

Code:

```
# Homework 4

setwd("C:\\Users\\Aditya Peshin\\Desktop\\Studies\\UC Berkeley\\Fall 2019\\242 - Applications in Data Analysis\\Assignments\\hw 4")
getwd()
library(tm)
library(SnowballC)
library(wordcloud)
library(MASS)
library(caTools)
library(dplyr)
library(rpart)
library(rpart.plot)
library(randomForest)
library(caret)
library(tm.plugin.webmining)
library(corpus)
library(boot)
library(ROCR)
library(GGally)

# Defining the functions to calculate the metrics

tableAccuracy <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/length(test)
  return(a)
}

tableTPR <- function(label, pred) {
  t = table(label, pred)
  return(t[2,2]/(t[2,1] + t[2,2]))
}

tableFPR <- function(label, pred) {
  t = table(label, pred)
  return(t[1,2]/(t[1,1] + t[1,2]))
}

tablePrec <- function(label, pred) {
  t = table(label, pred)
```

```
  return(t[2,2]/(t[1,2] + t[2,2]))
}

boot_accuracy <- function(data, index) {
  labels <- data$response[index]
  predictions <- data$prediction[index]
  return(tableAccuracy(labels, predictions))
}

boot_tpr <- function(data, index) {
  labels <- data$response[index]
  predictions <- data$prediction[index]
  return(tableTPR(labels, predictions))
}

boot_fpr <- function(data, index) {
  labels <- data$response[index]
  predictions <- data$prediction[index]
  return(tableFPR(labels, predictions))
}

boot_all_metrics <- function(data, index) {
  acc = boot_accuracy(data, index)
  tpr = boot_tpr(data, index)
  fpr = boot_fpr(data, index)
  return(c(acc, tpr, fpr))
}
# Load file into R, create new column for predictor variable
stack_orig <- read.csv("sover.csv", stringsAsFactors = FALSE)
stack_orig$BinarySc <- ifelse(stack_orig$Score>=1, 1, 0)
str(stack_orig)

# Converting to factor variable
stack_orig$BinarySc <- as.factor(stack_orig$BinarySc)
st2 <- stack_orig
st2$Score <- NULL

v1 = Corpus(VectorSource(st2$Title))
v2 = Corpus(VectorSource(st2$Body))

j = 0
length(v2)
```

```
for (i in 1:length(v2)) {
  v2[[i]][["content"]] = extractHTMLStrip(v2[[i]][["content"]])
  print(j)
  j = j+1
}

# Changing text to lower case
modv1 = tm_map(v1, tolower)
modv2 = tm_map(v2, tolower)

strwrap(modv1[["1"]])
strwrap(modv2[["1"]])

# Removing \n from text
modv1 = tm_map(modv1, removeWords, "\n")
modv2 = tm_map(modv2, removeWords, "\n")

strwrap(modv1[["1"]])
strwrap(modv2[["1"]])

# Removing Punctuation
modv1 = tm_map(modv1, removePunctuation)
modv2 = tm_map(modv2, removePunctuation)

strwrap(modv1[["1"]])
strwrap(modv2[["1"]])

# Removing Stop Words

# stopwords("english")[1:10]
# length(stopwords("english"))
modv1 = tm_map(modv1, removeWords, stopwords("english"))
modv2 = tm_map(modv2, removeWords, stopwords("english"))

strwrap(modv1[["1"]])
strwrap(modv2[["1"]])

# Stemming the Document

modv1 = tm_map(modv1, stemDocument)
modv2 = tm_map(modv2, stemDocument)

strwrap(modv1[["1"]])
```

```
strwrap(modv2[["1"]])

# Frequencies of words appearing

f1 = DocumentTermMatrix(modv1)
f2 = DocumentTermMatrix(modv2)

# For me to see which teerms are the most common
findFreqTerms(f1, lowfreq=450)
findFreqTerms(f2, lowfreq=2800)

# Identifying number of sparse terms
sparse1 = removeSparseTerms(f1, 0.92)
sparse2 = removeSparseTerms(f2, 0.835)

# Creating Document Term Matrix

Title_TM <- as.data.frame(as.matrix(sparse1))
colnames(Title_TM)
Body_TM <- as.data.frame(as.matrix(sparse2))
colnames(Body_TM)

colnames(Title_TM) <- paste0("Title" , sep = '_' , colnames(Title_TM) )
colnames(Title_TM)
colnames(Body_TM) <- paste0("Body", sep = "_", colnames(Body_TM))
colnames(Body_TM)

Combined_TM <- cbind(Title_TM, Body_TM)
colnames(Combined_TM)
nrow(Combined_TM)
Combined_TM$BinarySc <-  st2$BinarySc

# Splitting data into training and test sets

set.seed(123)
spl = sample.split(Combined_TM$BinarySc, SplitRatio = 0.7)

Combined_TM.train <- filter(Combined_TM, spl == TRUE)
Combined_TM.test <- filter(Combined_TM, spl == FALSE)

# Logistic Regression

mod_glm <- glm(BinarySc~. , data = Combined_TM.train, family = "binomial")
```

```
summary(mod_glm)

PredictLog = predict(mod_glm, newdata = Combined_TM.test, type = "response")
table(Combined_TM.test$BinarySc, PredictLog > 0.5)
#   FALSE TRUE
# 0   750  387
# 1   550  553
tableAccuracy(Combined_TM.test$BinarySc, PredictLog > 0.5)
# 0.5816964

# LDA

mod_lda = lda(BinarySc ~ ., data = Combined_TM.train)

PredictLDA = predict(mod_lda, newdata = Combined_TM.test)$class
table(Combined_TM.test$BinarySc, PredictLDA)
#     0   1
# 0 753 384
# 1 551 552
tableAccuracy(Combined_TM.test$BinarySc, PredictLDA)
# 0.5825893

# Cross Validated Random Forests

mod_basicRF = train(BinarySc ~ .,
                method = "rf",
                data=Combined_TM.train,
                tuneGrid = data.frame(mtry = 1:20),
                trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE))
mod_basicRF$results
# Best value of mtry = 2
final_CVrf = mod_basicRF$finalModel

Predict_CVRF = predict(final_CVrf, newdata = Combined_TM.test)
table(Combined_TM.test$BinarySc, Predict_CVRF)
#     0   1
# 0 822 315
# 1 644 459
tableAccuracy(Combined_TM.test$BinarySc, Predict_CVRF)
# 0.571875

as.data.frame(final_CVrf$importance) %>%
  mutate(Words = rownames(final_CVrf$importance)) %>%
```

```
  arrange(desc(MeanDecreaseGini))

# Stepwise Logistic Regression

mod_StepLog = step(mod_glm, direction = "backward")
summary(mod_StepLog)
length(mod_StepLog$coefficients)  # 20 variables selected in the final model

PredictStepLog = predict(mod_StepLog, newdata = Combined_TM.test, type = "response")
table(Combined_TM.test$BinarySc, PredictStepLog > 0.5)
# FALSE TRUE
# 0   749  388
# 1   561  542
tableAccuracy(Combined_TM.test$BinarySc, PredictStepLog > 0.5)
# 0.5763393

# Basic Boosting

mod_boost <- train( BinarySc ~ .,
                    data = Combined_TM.train,
                    method = "gbm",
                    metric = "Accuracy",
                    distribution = "bernoulli")
mod_boost

final_boost <- mod_boost$finalModel

PredictBoost = predict(mod_boost, newdata = Combined_TM.test)
table(Combined_TM.test$BinarySc, PredictBoost)
#    0   1
# 0 700 437
# 1 659 444
tableAccuracy(Combined_TM.test$BinarySc, PredictBoost)
# 0.5107143

# Cross Validated Boosting

tGrid = expand.grid(n.trees = (1:25)*100, interaction.depth = c(1,2,4,8),
                   shrinkage = 0.1, n.minobsinnode = 10)

mod_CVboost <- train(BinarySc ~ .,
                    data = Combined_TM.train,
                    method = "gbm",
```

```
                  tuneGrid = tGrid,
                  trControl = trainControl(method="cv", number=5, verboseIter = TRUE),
                  metric = "Accuracy",
                  distribution = "bernoulli")
mod_CVboost
mod_CVboost$results

# Boosting Cross Validation Round 2
tGrid2 = expand.grid(n.trees = (1:20)*50, interaction.depth = c(1,2,3),
                 shrinkage = 0.01, n.minobsinnode = 10)

mod_CVboost2 <- train(BinarySc ~ .,
                  data = Combined_TM.train,
                  method = "gbm",
                  tuneGrid = tGrid2,
                  trControl = trainControl(method="cv", number=5, verboseIter = TRUE),
                  metric = "Accuracy",
                  distribution = "bernoulli")
mod_CVboost2
mod_CVboost2$results
final_CVboost <- mod_CVboost2$finalModel
Combined_TM.test.mm = as.data.frame(model.matrix(BinarySc ~ . +0, data =
Combined_TM.test))
PredictCVBoost = predict(final_CVboost, newdata = Combined_TM.test.mm, n.trees = 650, type
= "response")
table(Combined_TM.test$BinarySc, PredictCVBoost < 0.5)
#   FALSE TRUE
# 0   756  381
# 1   553  550
tableAccuracy(Combined_TM.test$BinarySc, PredictCVBoost < 0.5)
# 0.5830357


# Bootstrapping the test set for identifying variability of performance metrics

Boost_test_set = data.frame(response = Combined_TM.test$BinarySc, predictions =
PredictCVBoost < 0.5)
set.seed(123)
Boost_boot = boot(Boost_test_set, boot_all_metrics, R = 10000)
Boost_boot
boot.ci(Boost_boot, index = 1, type = "basic")
boot.ci(Boost_boot, index = 2, type = "basic")
boot.ci(Boost_boot, index = 3, type = "basic")
```

```
# Calculating the performance of the model using precision

# For the Cross Validated Boosting MOdel
CVboostpred <- prediction(1 - PredictCVBoost, Combined_TM.test$BinarySc)
CVBoostPerf <- performance(CVboostpred, "prec")
plot(CVBoostPerf, colorize = FALSE)
as.numeric(performance(CVboostpred, "auc")@y.values)

# For the Logistic Regression Model
logisticpred <- prediction(PredictLog, Combined_TM.test$BinarySc)
logisticPerf <- performance(logisticpred, "prec")
plot(logisticPerf, colorize = FALSE)
as.numeric(performance(logisticpred, "auc")@y.values)

# For the Stepwise Logistic Regression Model
Steplogpred <- prediction(PredictStepLog, Combined_TM.test$BinarySc)
SteplogPerf <- performance(Steplogpred, "prec")
plot(SteplogPerf, colorize = FALSE)
as.numeric(performance(Steplogpred, "auc")@y.values)

# Calculating the value of precision for each of the models at their optima

# CV Boosting
tablePrec(Combined_TM.test$BinarySc, PredictCVBoost < 0.6)
#0.40  -> 0.690 prec

# Logistic Regression
tablePrec(Combined_TM.test$BinarySc, PredictLog > 0.735)
#0.735 -> 0.774 prec

# Stepwise Logistic Regression
tablePrec(Combined_TM.test$BinarySc, PredictStepLog > 0.625)
#0.625 -> 0.6835 prec

table(Combined_TM.test$BinarySc)
```