ML-Based Phishing Website Detection

Sarviin Hari

2024-04-28

```
setwd("/Users/Sarviin Hari/Desktop/Data")
```

1. Data Exploration

```
rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32885741) # Your Student ID is the random seed
L <- as.data.frame(c(1:50))
L <- L[sample(nrow(L), 10, replace = FALSE),]
Phish <- Phish[(Phish$A01 %in% L),]
PD <- Phish[sample(nrow(Phish), 2000, replace = FALSE),] # sample of 2000 rows
```

```
# Import libraries
library(tree)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```
library(adabag)
```

```
## Warning: package 'adabag' was built under R version 4.3.3
```

```
## Loading required package: rpart
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

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

```
## Loading required package: foreach
```

```
## Loading required package: doParallel
## Warning: package 'doParallel' was built under R version 4.3.3
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(rpart)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
## randomForest 4.7-1.1
```

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

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

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

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.3.3
```

```
library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

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

```
library(randomForest)
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.3.3
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:ROCR':
##
##     prediction
```

```
library(car)
```

```
## Warning: package 'car' was built under R version 4.3.3
```

```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 4.3.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

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

```
## The following object is masked from 'package:car':
##
##     recode
```

```
## The following object is masked from 'package:neuralnet':
##
##     compute
```

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

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

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

```
library(caret)
```

```
dim(PD)
```

```
## [1] 2000   26
```

```
str(PD)
```

```
## 'data.frame':    2000 obs. of  26 variables:
##  $ A01  : int  2 40 9 47 42 9 4 47 4 42 ...
##  $ A02  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A03  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A04  : int  3 2 2 2 2 3 2 3 3 3 ...
##  $ A05  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A06  : int  0 NA 0 0 1 0 1 0 1 0 ...
##  $ A07  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A08  : num  0.486 0.667 0.75 0.636 1 ...
##  $ A09  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A10  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A11  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A12  : int  232 474 369 255 569 232 255 232 232 232 ...
##  $ A13  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A14  : int  0 1 0 1 1 0 0 0 0 0 ...
##  $ A15  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A16  : int  0 1 0 0 0 0 1 1 1 0 ...
##  $ A17  : int  1 2 1 1 1 1 2 1 1 1 ...
##  $ A18  : int  8 165 6 15 124 29 22 19 87 104 ...
##  $ A19  : int  0 0 0 0 0 0 1 0 1 0 ...
##  $ A20  : int  0 1 0 0 1 0 0 0 1 0 ...
##  $ A21  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A22  : num  0.0412 0.0707 0.0493 0.0522 0.0505 ...
##  $ A23  : int  100 85 100 3 17 105 103 27 116 23 ...
##  $ A24  : num  0.52291 0.00075 0.00508 0.03265 0.00402 ...
##  $ A25  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Class: int  0 0 0 1 1 0 0 1 0 0 ...
```

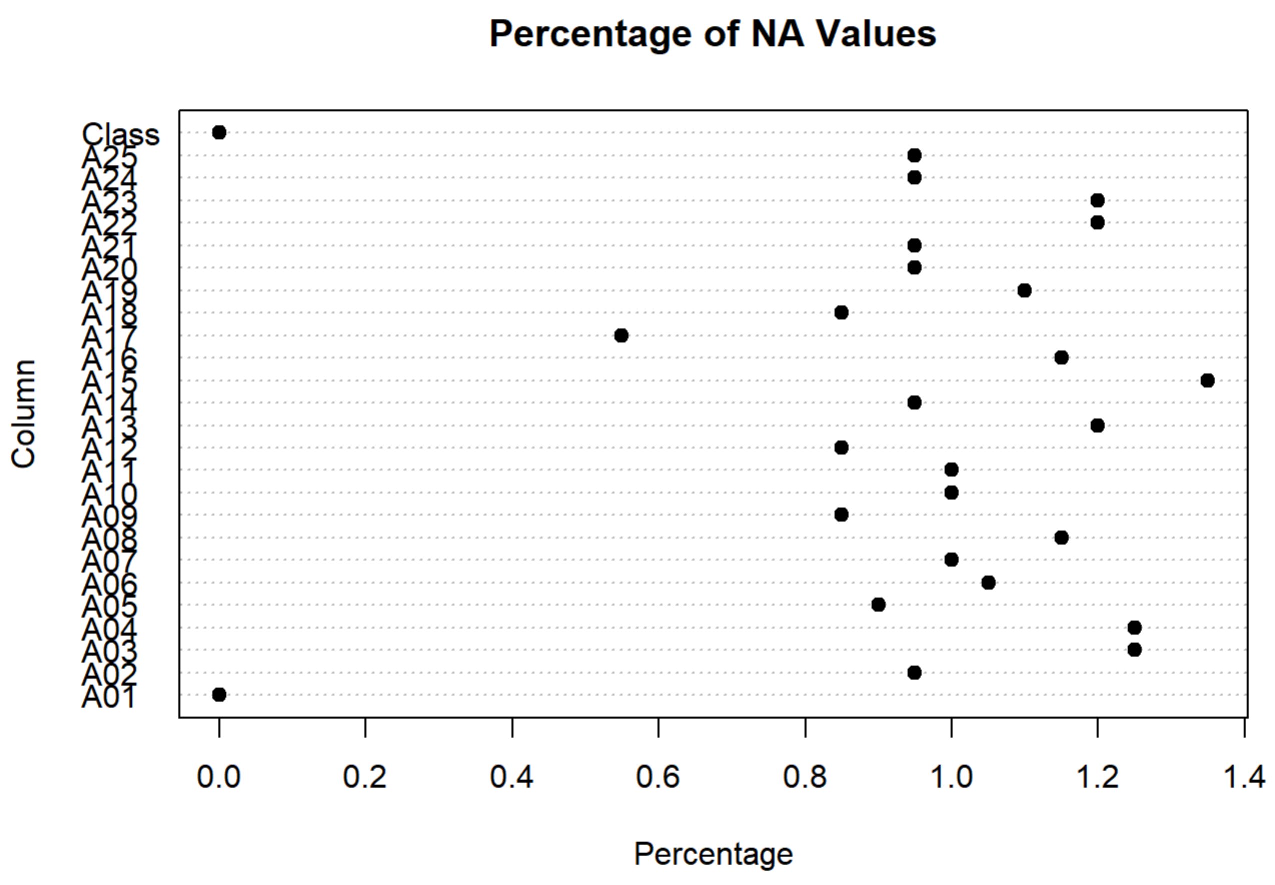
```
# Get the number of unique values in each column
unique_counts <- sapply(PD, function(x) length(unique(x)))
print(unique_counts)
```

```
##   A01   A02   A03   A04   A05   A06   A07   A08   A09   A10   A11   A12   A13 
##    10    17     3     7     6     3     3   216     3     3     9   122     5 
##   A14   A15   A16   A17   A18   A19   A20   A21   A22   A23   A24   A25 Class 
##     3     3     3     6   215     3     3     5  1977   188   120     7     2
```

```
# NA's in plot
p <- function(x) {
  sum(is.na(x)) / length(x) * 100
}
apply(PD, 2, p)
```

```
##   A01   A02   A03   A04   A05   A06   A07   A08   A09   A10   A11   A12   A13 
##  0.00  0.95  1.25  1.25  0.90  1.05  1.00  1.15  0.85  1.00  1.00  0.85  1.20 
##   A14   A15   A16   A17   A18   A19   A20   A21   A22   A23   A24   A25 Class 
##  0.95  1.35  1.15  0.55  0.85  1.10  0.95  0.95  1.20  1.20  0.95  0.95  0.00
```

```
p <- function(x) {
  sum(is.na(x)) / length(x) * 100
}

# Apply the function to each column of the data frame
na_percentages <- apply(PD, 2, p)

na_percentages_df <- data.frame(Column = names(na_percentages), Percentage = na_percentages)

# Create a dot plot
dotchart(na_percentages_df$Percentage, labels = na_percentages_df$Column, main = "Percentage
of NA Values", ylab = "Column", xlab = "Percentage", pch = 19)
```



```
aggregate(PD[26], PD[26], sum)
```

	Class <int>	Class <int>
	0	0
	1	584

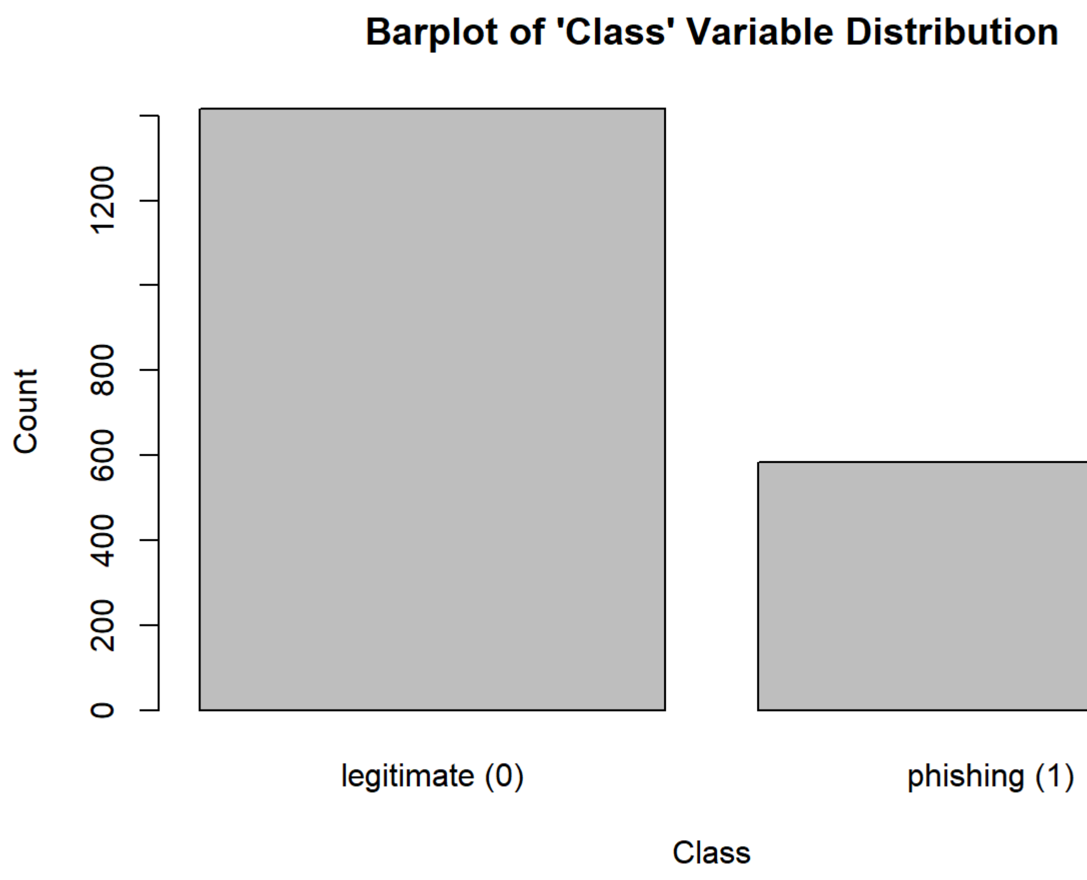
2 rows

```
Class.value.count = PD %>% group_by(Class) %>% summarise(Count = n()) %>% as.data.frame()
Class.value.count
```

Class <int>	Count <int>
0	1416
1	584

2 rows

```
barplot(Class.value.count$Count, names.arg = c('legitimate (0)', 'phishing (1)'), xlab = "Cla
ss", ylab = "Count", main = "Barplot of 'Class' Variable Distribution")
```



The dataset provided for the analysis has a dimension of 2000 rows and 26 columns. Using the str() method to view the datatypes of the columns, I observed that all the columns are of integer or numerical datatypes. From the summary of the columns of the dataset, we can also see that all the variables have NA values ranging from 11 to 27 except for the predictor A01 and the target variable, Class. By plotting the graph of Percentage of NA's vs the columns in the dataset, we can also observe that the highest percentage of NA's for the dataset for each column is < 1.4%

We can also deduce that the target variable that consists of 2 values “0” (legitimate) and “1” (phishing) has an uneven distribution of values through the mean of the values. Since the mean is < 0.5 (in between 0 and 1), we can observe that there is more data for “0”s than “1”s labels.

This is further proven by finding the value counts for the Count column, which shows that 1416 of the labels refers to the “0” (legitimate) and 584 labels refers to “1” (phishing). Since the proportion of “0”s is more than “1”s, we can conclude that the data provided has more “0”s than “1”s for the target variable, Class. Thus, our data is not a balanced data as it doesn't have a 50-50 proportion of “0”s and “1”s.

Following that, I printed all the value counts for every predictor for the dataset as well. From here I observed that some predictors have a relatively high proportion of data for a single category compared to other predictors. These data's might not be useful for analysis as they provide very minimal information about the predictor to distinguish between groups. They might act like constant values with no variation towards the data analysis and might induce overfitting as the model might take specific patterns of the data without generalizing it. So, during the pre-processing of the data I am considering removing the predictors that have 99% of the values belonging to a single category.

2. Data Preprocessing

```
## REMOVE DUPLICATED ROWS
unique_rows <- PD[!duplicated(PD), ]
# Display unique rows (original rows without duplicates)
print(dim(unique_rows))
```

```
## [1] 2000   26
```

```
### No Duplicated Rows
```

During the pre-processing of data, I decided to delete the duplicated rows in the data as they don't provide any valuable analysis. Unfortunately, there is no duplicated rows in my data, thus, the dimension of the data remains as (2000, 26)

```
old_PD = PD
clean_PD = PD %>% na.omit()
dim(clean_PD)
```

```
## [1] 1556   26
```

```
print(nrow(clean_PD ) / nrow(PD) *100)
```

```
## [1] 77.8
```

Next, I remove all the rows with NA values from the data. After removing the NA values, the data consists of 1556 rows and 26 columns where we retain 77.8% of the initial data. While some data might be lost, since our model is a classification of a binary class, 1556 rows of data would be sufficient to train a classification model to identify the pattern with the remaining predictors to classify between the two classes of the target variable. I removed NA values instead of imputation to preserve the integrity of data as imputed data for NA values may incur bias and potentially affect the original distribution of data.

```
# Get the number of unique values in each column
unique_counts <- sapply(clean_PD, function(x) length(unique(x)))
print(unique_counts)
```

```
##   A01   A02   A03   A04   A05   A06   A07   A08   A09   A10   A11   A12   A13 
##    10    15     2     6     5     2     2   187     2     2     8   114     4 
##   A14   A15   A16   A17   A18   A19   A20   A21   A22   A23   A24   A25 Class 
##     2     2     2     5   200     2     2     4  1556   165   112     6     2
```

```
columns_unique_lt_10 <- names(which(sapply(clean_PD, function(x) length(unique(x))) < 10))
print(columns_unique_lt_10)
```

```
##  [1] "A03"   "A04"   "A05"   "A06"   "A07"   "A09"   "A10"   "A11"   "A13"
## [10] "A14"   "A15"   "A16"   "A17"   "A19"   "A20"   "A21"   "A25"   "Class"
```

```
clean_PD[,c(3:7, 9:11, 13:17, 19:21, 25)]=lapply(clean_PD[,c(3:7, 9:11, 13:17, 19:21, 25)] ,
factor)
```

Next, I will find the predictors (excluding the target variable) that has < 10 unique values and convert these predictors to a factorial column. This measure is taken to improve the interpretability of the model for these predictors.

```
clean_PD = clean_PD %>% mutate(Class = factor(Class, levels=c(0,1), labels=c('legitimate
(0)', 'phishing (1)')))
```

Next, I converted the target variable, Class or a factor of 2 levels, where the integer “0” refers to ‘legitimate (0)’ while the integer “1” refers to ‘phishing (1)’.

```
# Number of datas in a single label
for (col in names(clean_PD)) {
  # Get the frequency table
  value_counts <- table(clean_PD[[col]])

  # Find the maximum frequency
  max_freq <- max(value_counts)

  cat("Column:", col, "\n")
  print(max_freq)
  }
```

```
## Column: A01
## [1] 167
## Column: A02
## [1] 1456
## Column: A03
## [1] 1554
## Column: A04
## [1] 1032
## Column: A05
## [1] 1548
## Column: A06
## [1] 1355
## Column: A07
## [1] 1551
## Column: A08
## [1] 944
## Column: A09
## [1] 1520
## Column: A10
## [1] 1506
## Column: A11
## [1] 1535
## Column: A12
## [1] 748
## Column: A13
## [1] 1551
## Column: A14
## [1] 1357
## Column: A15
## [1] 1354
## Column: A16
## [1] 1499
## Column: A17
## [1] 1171
## Column: A18
## [1] 69
## Column: A19
## [1] 1405
## Column: A20
## [1] 1166
## Column: A21
## [1] 1532
## Column: A22
## [1] 1
## Column: A23
## [1] 341
## Column: A24
## [1] 748
## Column: A25
## [1] 1551
## Column: Class
## [1] 1091
```

```
# remove variables with 99% of values in one single label
threshold = 0.99*nrow(clean_PD)
clean_PD$A03 = NULL
clean_PD$A05 = NULL
clean_PD$A07 = NULL
clean_PD$A13 = NULL
clean_PD$A25 = NULL
```

Next, I decide to remove the predictors which has 99% of the values belonging to a single class (> 1540). This results in 5 predictors to be dropped which are A03, A05, A07, A13 and A25. I decided to not remove any other predictors as there might be valuable information in the remaining predictors (even if 98%/97% are within the same category) due to the limited number of remaining predictors (20 predictors) and due to the remaining predictors having at least 15 data values being in other categories that might still hold specific interpretation for the model that can provide useful information.

3. Data Train-Test Split

```
data_to_split = clean_PD

set.seed(32885741) #Student ID as random seed
train.row = sample(1:nrow(data_to_split), 0.7*nrow(data_to_split))
data_to_split.train = data_to_split[train.row,]
data_to_split.test = data_to_split[-train.row,]
```

```
dim(data_to_split)
```

```
## [1] 1556   21
```

```
dim(data_to_split.train)
```

```
## [1] 1089   21
```

```
dim(data_to_split.test)
```

```
## [1] 467  21
```

Next, I divided my data into training and testing dataset from the remaining data with a dimension of (1556, 21) to 70% training data, yielding a dimension of (1089, 21) and to 30% testing data, yielding a dimension of (467, 21)

4. Model Fitting

DECISION TREE

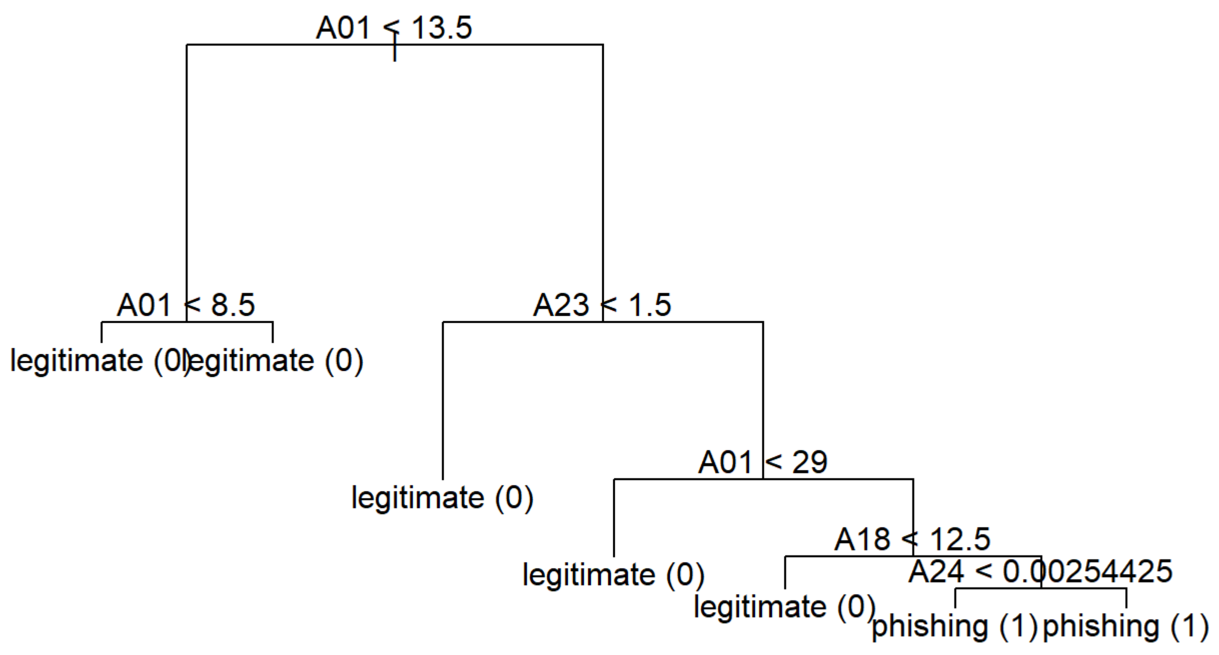
```
ptfit = tree(Class ~ ., data = data_to_split.train)
summary(ptfit)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train)
## Variables actually used in tree construction:
## [1] "A01" "A23" "A18" "A24"
## Number of terminal nodes:  7
## Residual mean deviance:  0.6854 = 741.6 / 1082
## Misclassification error rate: 0.1258 = 137 / 1089
```

```
ptfit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1089 1339.00 legitimate (0) ( 0.69513 0.30487 )
##    2) A01 < 13.5 536  289.60 legitimate (0) ( 0.92351 0.07649 )
##      4) A01 < 8.5 433  168.10 legitimate (0) ( 0.95150 0.04850 ) *
##      5) A01 > 8.5 103  101.40 legitimate (0) ( 0.80583 0.19417 ) *
##    3) A01 > 13.5 553  765.10 phishing (1) ( 0.47378 0.52622 )
##      6) A23 < 1.5 167  130.20 legitimate (0) ( 0.86826 0.13174 ) *
##      7) A23 > 1.5 386  473.60 phishing (1) ( 0.30311 0.69689 )
##       14) A01 < 29 99  126.00 legitimate (0) ( 0.66667 0.33333 ) *
##       15) A01 > 29 287  268.60 phishing (1) ( 0.17770 0.82230 )
##         30) A18 < 12.5 14   11.48 legitimate (0) ( 0.85714 0.14286 ) *
##         31) A18 > 12.5 273  223.90 phishing (1) ( 0.14286 0.85714 )
##           62) A24 < 0.00254425 28   38.67 phishing (1) ( 0.46429 0.53571 ) *
##           63) A24 > 0.00254425 245  165.80 phishing (1) ( 0.10612 0.89388 ) *
```

```
plot(ptfit)
text(ptfit, pretty = 0)
```



A decision tree is a supervised machine learning algorithm to classify an instance to each decision modes by comparison of the predictors to a specific value or a range. I created a basic decision tree model with the “tree” function from the library, “tree” using the training data. From the decision tree plotted we can see that out of the 20 predictors, the decision tree model classified using 4 predictors only which are A01, A23, A18 and A24

```
tpredict = predict(ptfit, data_to_split.test, type="class")

cat("Decision Tree" , "\n")
```

```
## Decision Tree
```

```
confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            313           21
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
cat("\n")
```

```
cat("Accuracy: ", accuracy)
```

```
## Accuracy:  0.8415418
```

NAIVE BAYES

```
naiveBayesFit = naiveBayes(Class ~. , data = data_to_split.train)

tbpredict = predict(naiveBayesFit, data_to_split.test)
cat("Naive Bayes" , "\n")
```

```
## Naive Bayes
```

```
confusion_matrix = table(actual = data_to_split.test$Class, predicted = tbpredict)
confusion_matrix
```

```
##                  predicted
## actual            legitimate (0) phishing (1)
##   legitimate (0)            277           57
##   phishing (1)               43           90
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
cat("\n")
```

```
cat("Accuracy: ", accuracy)
```

```
## Accuracy:  0.7858672
```

Naïve Bayes is a probabilistic classifier that is created based upon the Bayes theorem with a strong assumption that the predictors are independent to each other. The model is built upon the fact that it is consistent against attributes that are no relevant and the adaptability to a new set of data. I created a naïve bayes model using the naiveBayes function from the “e1071” library on the training data

BAGGING

```
set.seed(32885741)
ibag <- bagging(Class ~ ., data=data_to_split.train, mfinal=10)

ibpred <- predict.bagging(ibag, newdata=data_to_split.test)
cat("Bagging" , "\n")
```

```
## Bagging
```

```
confusion_matrix = table(observed = data_to_split.test$Class, predicted = ibpred$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            317           17
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
cat("\n")
```

```
cat("Accuracy: ", accuracy)
```

```
## Accuracy:  0.8501071
```

Bagging is a ensemble method for classification that makes multiple clones of initial data by row sampling with replacement on the training dataset, for each clones, a single classifier is created and all classifiers are combined based on majority voting to produce a final decision. Before creating a bagging model, I set the seed to my student id first to reproduce the same result for each iteration and to control the randomness in analysis. Next, I fit my bagging model using the “bagging” function on the training data set. I have set the maximum number of iterations for the training, mfinal to 10 to control the number of decision trees built and combined.

BOOSTING

```
set.seed(32885741)
iboost <- boosting(Class ~ ., data=data_to_split.train, mfinal=10)

ibpred <- predict.boosting(iboost, newdata=data_to_split.test)
cat("Boosting" , "\n")
```

```
## Boosting
```

```
confusion_matrix = table(observed = data_to_split.test$Class, predicted =ibpred$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            299           35
##   phishing (1)               50           83
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
cat("\n")
```

```
cat("Accuracy: ", accuracy)
```

```
## Accuracy:  0.8179872
```

Boosting is a ensemble method for classification that creates multiple trees using the original dataset (where predictors that are hard to classify are given higher weights) by classifying based on the weighted sum of each classifier by assigning a higher weights to accurate trees. Before creating a boosting model, I set the seed to my student id. Next, I fit my boosting model using the “boosting” function on the training data. I set the max number of iterations for the training, mfinal to 10 to control the number of decision trees built and combined.

RANDOM FOREST

```
set.seed(32885741)
fit.rf <- randomForest(Class ~ ., data=data_to_split.train)

rf.pred <- predict(fit.rf, data_to_split.test)

cat("Random Forest" , "\n")
```

```
## Random Forest
```

```
confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            318           16
##   phishing (1)               53           80
```

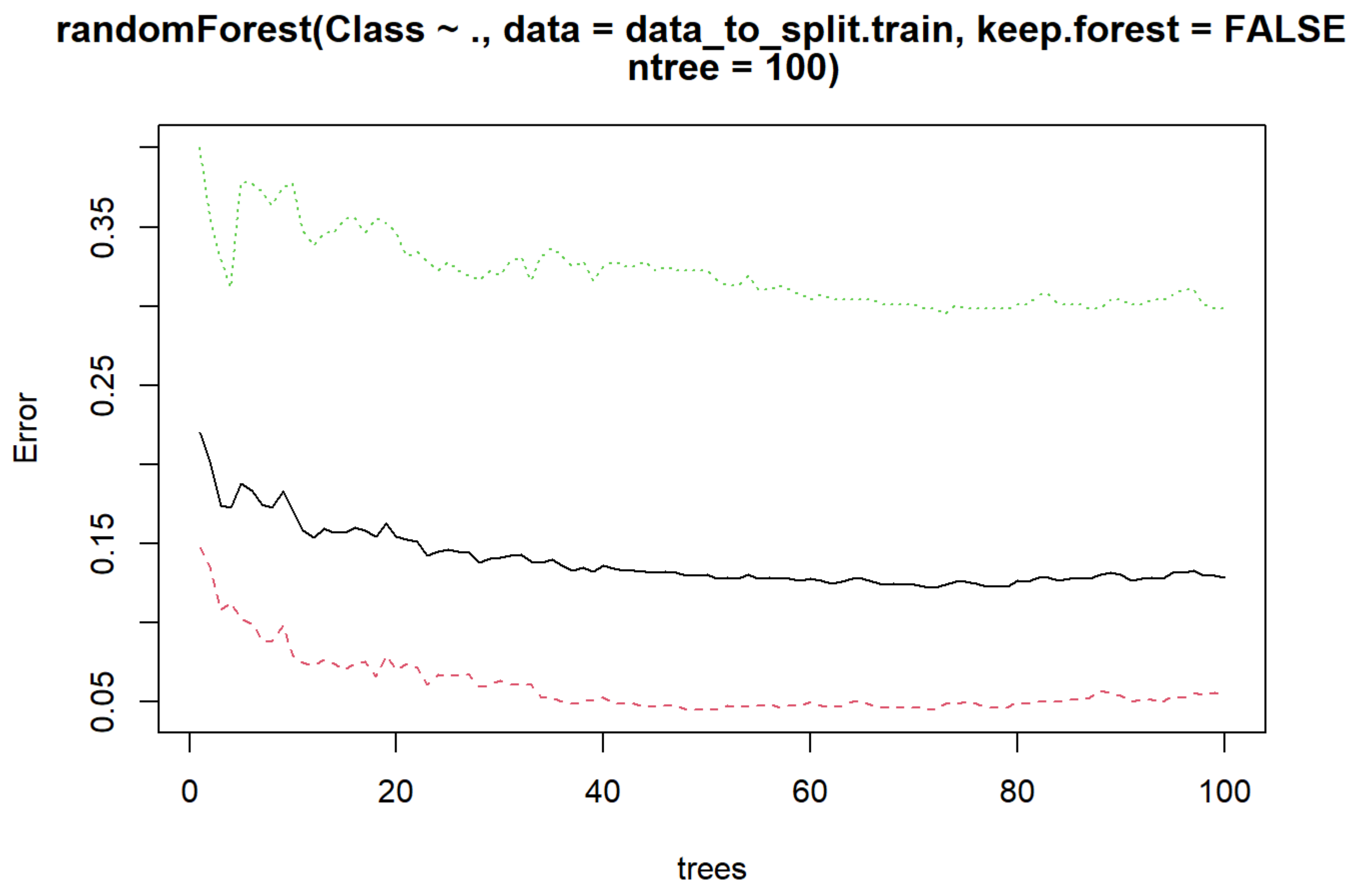
```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
cat("\n")
```

```
cat("Accuracy: ", accuracy)
```

```
## Accuracy:  0.8522484
```

Random Forest is a ensemble method for classification that makes multiple datasets from the training data with row sampling and subset of predictors, for each dataset, a single decision tree is created and all classifiers are combined based on majority voting to produce a final decision. Before creating a random forest model, I set the seed to my student id. Next, I fit my random forest model using the “randomForest” function on the training data set.

```
plot(randomForest(Class ~ ., data=data_to_split.train, keep.forest=FALSE, ntree=100))
```



Summary of Accuracy and AUC for Decision Tree, Naïve Bayes, Bagging, Boosting and Random Forest

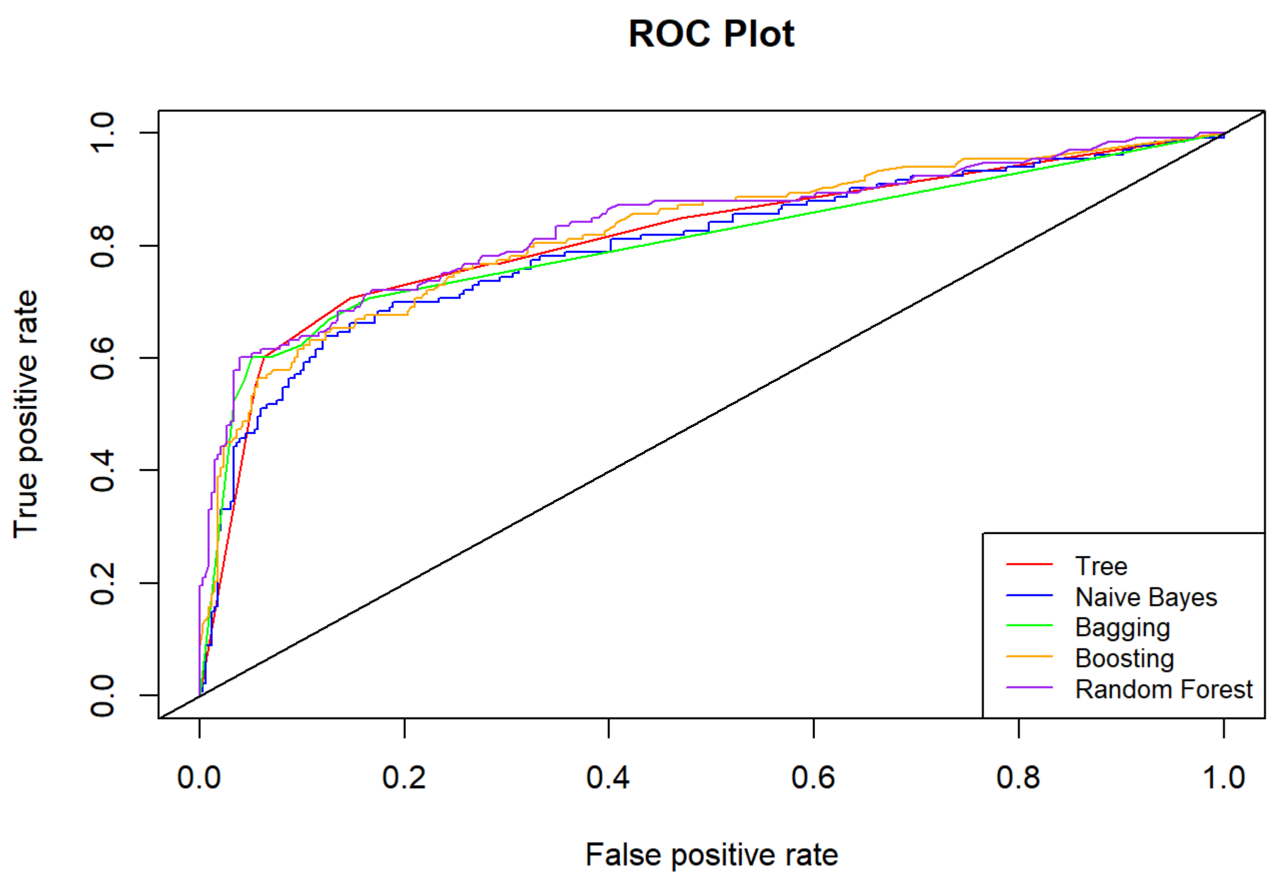
From these models, we can see that the model with the highest accuracy is the Random Forest model. This is because Random Forest is an ensemble learning model that is built by combining multiple decision trees. Since each tree in Random Forest is built upon row sampling and random subset of predictors, a diverse decision models are built for the random forest that captures the complex relationships between the predictors and the target variable, Class. The averaging method by majority voting to get the final best decision also ensures that the model created has a lesser chance of overfitting and generalize the data well, thus the high accuracy of the model. The model with the lowest accuracy is the Naïve Bayes model. This is mainly due to the assumption that we had to make while fitting the model that the predictors are independent to each other. Since this might not have been the case and some predictors might have influenced the decision of other predictors, the accuracy of the Naïve Bayes model decreases significantly. Naïve Bayes model also might not have been able to capture the complex relationship between the predictors and the target variable which results in underfitting of the model, hence the decrease in the accuracy.

5. ROC curve & AUC metrics for classifiers

```
# do predictions as probabilities and draw ROC
PD.pred1 = predict(ptfit, data_to_split.test, type = "vector")
PD.pred2 = predict(naiveBayesFit, data_to_split.test, type = "raw")
PD.pred3 = predict.bagging(ibag, data_to_split.test)
PD.pred4 = predict.boosting(iboost, newdata=data_to_split.test)
PD.pred5 = predict(fit.rf, data_to_split.test, type = "prob")

?prediction
```

```
## starting httpd help server ... done
```

```
# # computing a simple ROC curve (x-axis: fpr, y-axis: tpr)
# # labels are actual values, predictors are probability of class low
pred1 <- ROCR :: prediction(PD.pred1[,2], data_to_split.test$Class)
pred2<- ROCR :: prediction(PD.pred2[,2], data_to_split.test$Class)
pred3 <- ROCR :: prediction(PD.pred3$prob[,2], data_to_split.test$Class)
pred4 <- ROCR :: prediction(PD.pred4$prob[,2], data_to_split.test$Class)
pred5 <- ROCR :: prediction(PD.pred5[,2], data_to_split.test$Class)

# Plot ROC
perf1 <- performance(pred1,"tpr","fpr")
perf2 <- performance(pred2,"tpr","fpr")
perf3 <- performance(pred3,"tpr","fpr")
perf4 <- performance(pred4,"tpr","fpr")
perf5 <- performance(pred5,"tpr","fpr")

plot(perf1, col = "red", main="ROC Plot")
plot(perf2, add = TRUE, col = "blue")
plot(perf3, add = TRUE, col = "green")
plot(perf4, add = TRUE, col = "orange")
plot(perf5, add = TRUE, col = "purple")

legend("bottomright",  # Specify the position of the legend
       legend = c("Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),  # Labels f
or the legend
       col = c("red", "blue", "green", "orange", "purple"),  # Colors corresponding to each l
ine
       lty = 1, cex = 0.8)  # Line type for each line (solid line)



abline(0,1)
```



The ROC curve plotted for all 5 classifiers can be seen in the plot above. ROC is a measure of performance of a classifier given as a single point for a graph of TPR vs FPR. It gives an all-inclusive comparison of multiple classifiers by visualizing the goodness of these classifiers on a plot. Based on the plot, we can see that all 5 classifiers are better than a 'random' classifier as the plotted curves are above the threshold of the diagonal line. The plot also suggests that the pattern of the TPR vs FPR for all 5 classifiers are almost similar to each other where we can see the distribution of the ROC Curve are very close to each other. So, to understand the performance of the model from the ROC better we calculated the AUC for each of the classifier. The statistical property of the AUC suggests that the AUC of a classifier is equals to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance (phishing (1) or legitimate (0)). Based on the AUC, we can see that the Random Forest has the best performance with an AUC of 0.8357683. This suggests that the Random Forest model is able to well differentiate the positive and negative classes (legitimate or phishing) due to the ability of Random Forest to understand the complex pattern of the predictors by row sampling and feature subset selection. The model with the lowest AUC is Naïve Bayes, which might be due to the Naïve Bayes model unable to generalize the positive and negative classes well due to the complex characteristics of the predictors. Although Random Forest has the highest AUC, the AUC between each of the classifiers are still close to each other (proving the closeness of the pattern of the curve in the ROC plot), where the AUC are > 0.8 for all the models suggesting that all the classifiers have a reasonable performance in distinguishing the positive and negative classes.

```
auc_value1 <- auc(data_to_split.test$Class, PD.pred1[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value2 <- auc(data_to_split.test$Class, PD.pred2[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value3 <- auc(data_to_split.test$Class, PD.pred3$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value4 <- auc(data_to_split.test$Class, PD.pred4$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value5 <- auc(data_to_split.test$Class, PD.pred5[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
cat("Area Under the Curve:-", "\n")
```

```
## Area Under the Curve:-
```

```
cat("  Decision Tree: ", auc_value1, "\n")
```

```
##   Decision Tree:  0.816251
```

```
cat("  Naive Bayes:   ", auc_value2, "\n")
```

```
##   Naive Bayes:    0.8000991
```

```
cat("  Bagging:       ", auc_value3, "\n")
```

```
##   Bagging:        0.8048602
```

```
cat("  Boosting:      ", auc_value4, "\n")
```

```
##   Boosting:       0.8238711
```

```
cat("  Random Forest: ", auc_value5, "\n")
```

```
##   Random Forest:  0.8357683
```

```
             Accuracy
```

Tree 0.8415418 Naive Bayes 0.7858672 Bagging 0.8501071 Boosting 0.8179872 Random Forest 0.8522484

In terms of Accuracy and AUC, the classifier Random Forest has a higher value compared to the other classifiers. This suggests that Random Forest is the single “best” classifier among all 5 classifier models. The Random Forest model dominates both the accuracy and the AUC metrics is due to the ability of the model to identify the complex pattern of the predictors due to the random sampling and feature selection property of random forest when fitting multiple decision tree that allows the model to capture multiple different patterns of the predictors and the concurrent effect on the target variable, Class different set of features. Random Forest’s properties such as ability to capture non-linear interaction of the predictors, ensemble average by majority voting in minimizing overfitting of model, lesser sensitivity to noisy data and capturing the predictor importance better suggests that Random Forest is definitely the ‘best’ classifier compared to other classifiers.

6. Most important variables in predicting phishing or legitimate websites

Decision Tree

```
ptfit2 <- tree(Class ~ ., data = data_to_split.train[, c("A01", "A23", "A18", "A24", "Clas
s")])

summary(ptfit2)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train[, c("A01",
##     "A23", "A18", "A24", "Class")])
## Number of terminal nodes:  7
## Residual mean deviance:  0.6854 = 741.6 / 1082
## Misclassification error rate: 0.1258 = 137 / 1089
```

```
tpredict = predict(ptfit2, data_to_split.test, type="class")

confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            313           21
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8415418
```

```
PD.pred1 = predict(ptfit2, data_to_split.test, type = "vector")
pred1 <- ROCR :: prediction(PD.pred1[,2], data_to_split.test$Class)
auc_value1 <- auc(data_to_split.test$Class, PD.pred1[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value1
```

```
## Area under the curve: 0.8163
```

```
ptfit3 <- tree(Class ~ ., data = data_to_split.train[, c("A01", "A23", "Class")])

summary(ptfit3)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train[, c("A01",
##     "A23", "Class")])
## Number of terminal nodes:  7
## Residual mean deviance:  0.688 = 744.5 / 1082
## Misclassification error rate: 0.1221 = 133 / 1089
```

```
tpredict = predict(ptfit3, data_to_split.test, type="class")

confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            313           21
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8415418
```

```
PD.pred1 = predict(ptfit3, data_to_split.test, type = "vector")
pred1 <- ROCR :: prediction(PD.pred1[,2], data_to_split.test$Class)
auc_value1 <- auc(data_to_split.test$Class, PD.pred1[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value1
```

```
## Area under the curve: 0.8139
```

```
ptfit4 <- tree(Class ~ ., data = data_to_split.train[, c("A23", "Class")])

summary(ptfit4)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train[, c("A23",
##     "Class")])
## Number of terminal nodes:  4
## Residual mean deviance:  0.8818 = 956.8 / 1085
## Misclassification error rate: 0.1699 = 185 / 1089
```

```
tpredict = predict(ptfit4, data_to_split.test, type="class")

confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            301           33
##   phishing (1)               55           78
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8115632
```

```
PD.pred1 = predict(ptfit4, data_to_split.test, type = "vector")
pred1 <- ROCR :: prediction(PD.pred1[,2], data_to_split.test$Class)
auc_value1 <- auc(data_to_split.test$Class, PD.pred1[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value1
```

```
## Area under the curve: 0.7436
```

```
ptfit5 <- tree(Class ~ ., data = data_to_split.train[, c("A01", "Class")])

summary(ptfit5)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train[, c("A01",
##     "Class")])
## Number of terminal nodes:  4
## Residual mean deviance:  0.9319 = 1011 / 1085
## Misclassification error rate: 0.2415 = 263 / 1089
```

```
tpredict = predict(ptfit5, data_to_split.test, type="class")

confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            256           78
##   phishing (1)               44           89
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.738758
```

```
PD.pred1 = predict(ptfit5, data_to_split.test, type = "vector")
pred1 <- ROCR :: prediction(PD.pred1[,2], data_to_split.test$Class)
auc_value1 <- auc(data_to_split.test$Class, PD.pred1[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value1
```

```
## Area under the curve: 0.7529
```

For a decision tree model, the most important predictors in predicting whether the website is legitimate, or phishing are A01, A23, A18 and A24. This is because for the decision tree model, all the other predictors are omitted as they do not influence the performance of the decision tree, thus we have a model of 4 predictors with 7 leaves. Based on the original Decision Tree Model, to find the variables that can be omitted, I tried to remove the variables “A18” and “A24”, the nodes at the right most of the root which gave me an accuracy similar to the original model which is 0.8415418 with a slight decrease in AUC from 0.8163 to 0.8139. So next I again attempted to remove “A01” and “A23” individually and got an accuracy of 0.8115632 and 0.739758 respectively with a significant decrease in AUC for both models from 0.8163 to 0.7439 and 0.7529 respectively.

This suggests that the to get a very little negative effect on the performance we can remove the predictors “A18” and “A24” leaving the model with the variables “A23” and “A01” with the same accuracy of the previous model and a slight decrease in AUC.

Bagging

```
# Important variables of Decision Tree
# Assuming "ibag" is your data frame
# Filter column names where importance is greater than 0
selected_columns <- names(ibag$importance[ibag$importance > 0])

# Print the selected column names
print(selected_columns)
```

```
##  [1] "A01" "A02" "A04" "A08" "A12" "A14" "A17" "A18" "A22" "A23" "A24"
```

```
ibag$importance
```

```
##        A01        A02        A04        A06        A08        A09        A10
## 51.6420198  0.2005384  0.2465426  0.0000000  2.2351099  0.0000000  0.0000000
##        A11        A12        A14        A15        A16        A17        A18
##  0.0000000  0.6527752  0.5474656  0.0000000  0.0000000  0.5793072  3.9559241
##        A19        A20        A21        A22        A23        A24
##  0.0000000  0.0000000  0.0000000  2.1519122 35.6977117  2.0906931
```

```
set.seed(32885741) # Your Student ID is the random seed

ibag2 <- bagging(Class ~ ., data=data_to_split.train[,c("A01", "A04", "A02", "A08", "A12", "A
14", "A17", "A18", "A22", "A23", "A24", "Class")], mfinal=10)

ibpred2 <- predict.bagging(ibag2, newdata=data_to_split.test)


confusion_matrix = table(observed = data_to_split.test$Class, predicted = ibpred2$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            317           17
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8501071
```

```
PD.pred3 = predict.bagging(ibag2, data_to_split.test)
pred3 <- ROCR :: prediction(PD.pred3$prob[,2], data_to_split.test$Class)
auc_value3 <- auc(data_to_split.test$Class, PD.pred3$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value3
```

```
## Area under the curve: 0.8053
```

```
ibag2$importance
```

```
##        A01        A02        A04        A08        A12        A14        A17 
## 51.5041118  0.1996500  0.2454504  2.2252082  0.6498834  0.5450403  0.5767408 
##        A18        A22        A23        A24 
##  3.9383992  2.1423791 35.8917055  2.0814312
```

```
# set.seed
set.seed(32885741) # Your Student ID is the random seed

ibag3 <- bagging(Class ~ ., data=data_to_split.train[,c("A01", "A04", "A08", "A12", "A14", "A
17", "A18", "A22", "A23", "A24", "Class")], mfinal=10)

ibpred3 <- predict.bagging(ibag3, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted = ibpred3$class)
confusion_matrix
```

```
##                 predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            317           17
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8501071
```

```
PD.pred3 = predict.bagging(ibag3, data_to_split.test)
pred3 <- ROCR :: prediction(PD.pred3$prob[,2], data_to_split.test$Class)
auc_value3 <- auc(data_to_split.test$Class, PD.pred3$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value3
```

```
## Area under the curve: 0.8058
```

```
ibag3$importance
```

```
##        A01        A04        A08        A12        A14        A17        A18 
## 51.6071455  0.2459414  2.2296597  0.6511835  0.5461307  0.5778946  3.9462779 
##        A22        A23        A24 
##  2.1466649 35.9635067  2.0855951
```

```
set.seed(32885741) # Your Student ID is the random seed

ibag4 <- bagging(Class ~ ., data=data_to_split.train[,c("A01", "A08", "A12", "A14", "A17", "A18", "A22", "A23", "A24", "Class")], mfinal=10)

ibpred4 <- predict.bagging(ibag4, newdata=data_to_split.test)


confusion_matrix = table(observed = data_to_split.test$Class, predicted = ibpred4$class)
confusion_matrix
```

```
##                 predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            317           17
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8501071
```

```
PD.pred3 = predict.bagging(ibag4, data_to_split.test)
pred3 <- ROCR :: prediction(PD.pred3$prob[,2], data_to_split.test$Class)
auc_value3 <- auc(data_to_split.test$Class, PD.pred3$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value3
```

```
## Area under the curve: 0.8064
```

```
ibag4$importance
```

```
##        A01        A08        A12        A14        A17        A18        A22 
## 51.8262467  2.2391259  0.6539481  0.5484493  0.5803481  3.9630321  1.9782083 
##        A23        A24 
## 36.1161919  2.0944496
```

```
set.seed(32885741) # Your Student ID is the random seed

ibag5 <- bagging(Class ~ ., data=data_to_split.train[,c("A01", "A08", "A12", "A17", "A18", "A
22", "A23", "A24", "Class")], mfinal=10)

ibpred5 <- predict.bagging(ibag5, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted = ibpred5$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            316           18
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8479657
```

```
PD.pred3 = predict.bagging(ibag5, data_to_split.test)
pred3 <- ROCR :: prediction(PD.pred3$prob[,2], data_to_split.test$Class)
auc_value3 <- auc(data_to_split.test$Class, PD.pred3$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value3
```

```
## Area under the curve: 0.8079
```

```
ibag4$importance
```

```
##        A01        A08        A12        A14        A17        A18        A22 
## 51.8262467  2.2391259  0.6539481  0.5484493  0.5803481  3.9630321  1.9782083 
##        A23        A24 
## 36.1161919  2.0944496
```

For a bagging model, the important predictors are “A01”, “A02”, “A08”, “A12”, “A14”, “A17”, “A18”, “A22”, “A23” and “A24” for the base model with “A01” being the most important predictor with the highest importance score. The second model created with only the important predictors excluding the predictors that have zero importance to the base model gives an accuracy similar to the base model of 0.8501071 and a slight increase in AUC from 0.8049 to 0.8053 for the model with important predictors only and the base model. The slight increase in AUC is potentially due to the minimization of effect of non-important predictors on the model performance. The predictors with the highest importance to the model is “A01” with an accuracy of 51.6420198 and the predictors with a very low importance is “A02”. To find the influence of omitted predictors on the accuracy first I omitted the variable “A02” to create a third model based on the second model (with important predictors only) and obtained the same accuracy and a slight increase in AUC from 0.8049 (from base model) to 0.8058. This suggests that although there is no impact on accuracy, “A02” has been a noisy data that affects negatively the performance of the model. Next, I omit the predictors “A04 and “A14” from the third model (which has a very low importance from the third model) and get a fourth model with a slight decrease in the accuracy of the model to 0.8479657, where the difference is 0.0021414 and the AUC improved slightly to 0.8079. The slight increase in AUC suggests the models capability to discriminate between genuine and phishing classes but since we want our classification to perform better on accuracy to reduce the FP and FN rates, the decrease in accuracy has a significant impact on performance. Thus, the predictors that we can remove from the base model with only a slight effect on the performance of the model is predictors “A02”, “A14”, “A04” and predictors with 0 importance. These predictors either have no influence on the model or < 0.55 influence score for the model thus, these models only contribute to the noisiness on the data impacting the actual performance as we can see the improvement in the AUC and only a slight decrease in accuracy after removing these predictors. This leaves the model with the predictors “A01”, “A08”, “A12”, “A17”, “A18”, “A22”, “A23” and “A24”.

BOOSTING

```
# Important variables of Decision Tree
# Assuming "ibag" is your data frame
# Filter column names where importance is greater than 0
selected_columns <- names(iboost$importance[iboost$importance > 0])

# Print the selected column names
print(selected_columns)
```

```
##  [1] "A01" "A02" "A04" "A06" "A08" "A10" "A12" "A14" "A15" "A17" "A18" "A19"
## [13] "A20" "A22" "A23" "A24"
```

```
iboost$importance
```

```
##        A01        A02        A04        A06        A08        A09        A10
## 29.4455980  0.9689261  0.9803399  0.6518447  2.5761982  0.0000000  0.2587983
##        A11        A12        A14        A15        A16        A17        A18
##  0.0000000  4.5123923  1.0118641  1.3747826  0.0000000  2.6737612 13.3952814
##        A19        A20        A21        A22        A23        A24
##  0.5902995  0.8706185  0.0000000 16.6784972 16.8370511  7.1737469
```

```
set.seed(32885741) # Your Student ID is the random seed

iboost2 <- boosting(Class ~ ., data=data_to_split.train[,  c("A01", "A02", "A04", "A06", "A0
8","A10", "A12", "A14", "A15", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "Class")], mf
inal=10)

ibpred2 <- predict.boosting(iboost2, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =ibpred2$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            299           35
##   phishing (1)               50           83
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8179872
```

```
PD.pred4 = predict.boosting(iboost2, newdata=data_to_split.test)
pred4 <- ROCR :: prediction(PD.pred4$prob[,2], data_to_split.test$Class)
auc_value4 <- auc(data_to_split.test$Class, PD.pred4$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value4
```

```
## Area under the curve: 0.8239
```

```
iboost2$importance
```

```
##        A01        A02        A04        A06        A08        A10        A12 
## 29.4455980  0.9689261  0.9803399  0.6518447  2.5761982  0.2587983  4.5123923 
##        A14        A15        A17        A18        A19        A20        A22 
##  1.0118641  1.3747826  2.6737612 13.3952814  0.5902995  0.8706185 16.6784972 
##        A23        A24 
## 16.8370511  7.1737469
```

```
set.seed(32885741) # Your Student ID is the random seed

iboost2 <- boosting(Class ~ ., data=data_to_split.train[,  c("A01", "A02", "A04", "A06", "A0
8", "A12", "A14", "A15", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "Class")], mfinal=1
0)

ibpred2 <- predict.boosting(iboost2, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =ibpred2$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            304           30
##   phishing (1)               50           83
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8286938
```

```
PD.pred4 = predict.boosting(iboost2, newdata=data_to_split.test)
pred4 <- ROCR :: prediction(PD.pred4$prob[,2], data_to_split.test$Class)
auc_value4 <- auc(data_to_split.test$Class, PD.pred4$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value4
```

```
## Area under the curve: 0.8287
```

```
iboost2$importance
```

```
##        A01        A02        A04        A06        A08        A12        A14 
## 30.1569698  0.9791557  0.9859258  0.0000000  2.6314486  5.2946262  1.2697387 
##        A15        A17        A18        A19        A20        A22        A23 
##  1.6149565  2.5788254 13.6690572  0.5936630  0.6757181 17.1251139 16.6954701 
##        A24 
##  5.7293310
```

```
set.seed(32885741) # Your Student ID is the random seed

iboost3 <- boosting(Class ~ ., data=data_to_split.train[,  c("A01", "A02", "A04", "A08", "A1
2", "A14", "A15", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "Class")
], mfinal=10)

ibpred3 <- predict.boosting(iboost3, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =ibpred3$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            304           30
##   phishing (1)               50           83
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8286938
```

```
PD.pred4 = predict.boosting(iboost3, newdata=data_to_split.test)
pred4 <- ROCR :: prediction(PD.pred4$prob[,2], data_to_split.test$Class)
auc_value4 <- auc(data_to_split.test$Class, PD.pred4$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value4
```

```
## Area under the curve: 0.8287
```

```
iboost3$importance
```

```
##        A01        A02        A04        A08        A12        A14        A15
## 30.1569698  0.9791557  0.9859258  2.6314486  5.2946262  1.2697387  1.6149565
##        A17        A18        A19        A20        A22        A23        A24
##  2.5788254 13.6690572  0.5936630  0.6757181 17.1251139 16.6954701  5.7293310
```

```
set.seed(32885741) # Your Student ID is the random seed

iboost4 <- boosting(Class ~ ., data=data_to_split.train[,  c("A01", "A02",  "A04", "A08", "A1
2", "A14", "A15", "A17", "A18", "A20", "A22", "A23", "A24", "Class")
], mfinal=10)

ibpred4 <- predict.boosting(iboost4, newdata=data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =ibpred4$class)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)             287           47
##   phishing (1)                48           85
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.7965739
```

```
PD.pred4 = predict.boosting(iboost4, newdata=data_to_split.test)
pred4 <- ROCR :: prediction(PD.pred4$prob[,2], data_to_split.test$Class)
auc_value4 <- auc(data_to_split.test$Class, PD.pred4$prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value4
```

```
## Area under the curve: 0.8378
```

```
iboost4$importance
```

```
##        A01        A02        A04        A08        A12        A14        A15
## 30.6975183  0.7829612  1.3145006  3.1303802  7.2854797  0.8154749  1.3538979
##        A17        A18        A20        A22        A23        A24
##  1.9079401 12.1632999  0.6612532 15.7584127 18.0466634  6.0822179
```

For a boosting model, the important predictors are “A01”, “A02”, “A04”, “A06”, “A10”, “A08”, “A12”, “A14”, “A15”, “A17”, “A18”, “A19”, “A20”, “A22”, “A23” and “A24” for the base model with “A01” being the most important predictor with the highest importance score. The second model created with only the important predictors excluding the predictors that have zero importance to the base model gives the same accuracy and AUC score as the base model has only 2 predictors with 0 importance are omitted and these predictors don’t have a significant influence on the model. For the second model, the predictors with the highest importance to the model is “A01” with an importance of 29.4455980 and the predictors with a very low importance is “A10” with an importance of 0.2588. To find the influence of omitted predictors on the accuracy first I omitted the variable “A10” to create a third model based on the second model (with important predictors only) which has an accuracy and AUC higher than the base model and important predictors model which is 0.8286938 and 0.8287 respectively. This is an interesting insight as it suggests that the removing 0 importance predictors and predictor A10 of the model is able to influence the accuracy of the model by increasing the accuracy and AUC of the model (based on the first model with predictors of 0 influence). This suggests that the predictor A10 impacted negatively to the accuracy of the model as it contributed to the overfitting of the base model and induced noise in the base model which diminished the accuracy. Thus, by removing the predictor A10 the boosting model, which works with the weights assigned to the predictors that are hard to classify can focus on the important predictors instead of the non-important predictors, which increased the accuracy of the third model. So instead of the base model, we will find the predictors to be omitted from the third model (with no predictors with 0 importance and predictor A10). To find the influence of omitted predictors on the accuracy next, I omitted the variable “A06” to create a fourth model based on the third model (with important predictors only) and obtained the same accuracy and AUC for both models which is 0.8286938 and 0.8287 respectively. Next, I try to omit the predictor “A19” from the third model (which has a very low importance) and get a fourth model with a significant decrease in the accuracy of the model to 0.7965739 in comparison to the base model. So, I will not consider removing this predictor as it affects the performance of the model greatly. Thus, the predictors that we can remove from the base model with only a slight effect on the performance of the model is predictors “A06”, “A10” and predictors with 0 importance. This leaves the model with the predictors “A01”, “A02”, “A04”, “A08”, “A12”, “A14”, “A15”, “A17”, “A18”, “A19”, “A20”, “A22”, “A23”, “A24”

RANDOM FOREST

```
(fit.rf$importance)
```

```
##     MeanDecreaseGini
## A01      108.0517946
## A02        4.7817396
## A04        5.7826599
## A06        4.5617499
## A08       20.9560813
## A09        0.9157853
## A10        2.4276015
## A11        1.3062086
## A12       17.4267966
## A14       21.2946927
## A15        4.6770651
## A16        2.1267430
## A17        9.1816711
## A18       55.9942373
## A19        4.4634655
## A20        6.0309938
## A21        0.5428806
## A22       46.2121171
## A23       93.7625471
## A24       21.4955196
```

```
# set.seed(105)

set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A09", "A10", 'A11', "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A
23", "A24", "Class")])


rf.pred2 <- predict(fit.rf2, data_to_split.test)


confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                   predicted
## observed           legitimate (0) phishing (1)
##   legitimate (0)             321           13
##   phishing (1)                52           81
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8608137
```

```
PD.pred5 = predict(fit.rf2, data_to_split.test, type = "prob")
pred5 <- ROCR :: prediction(PD.pred5[,2], data_to_split.test$Class)
auc_value5 <- auc(data_to_split.test$Class, PD.pred5[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value5
```

```
## Area under the curve: 0.835
```

```
(fit.rf2$importance)
```

```
##     MeanDecreaseGini
## A01       111.719986
## A02         4.620432
## A04         5.972583
## A06         4.323349
## A08        21.760376
## A09         0.850857
## A10         2.366440
## A11         1.449986
## A12        17.400248
## A14        22.276971
## A15         4.654102
## A16         2.267306
## A17         9.457202
## A18        56.509270
## A19         4.531875
## A20         6.231673
## A22        47.364828
## A23        90.927730
## A24        21.881638
```

```
# set.seed(105)

set.seed(32885741) # Your Student ID is the random seed
fit.rf3 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", 'A11', "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A
24", "Class")])

rf.pred3 <- predict(fit.rf3, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred3)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            319           15
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8543897
```

```
PD.pred5 = predict(fit.rf3, data_to_split.test, type = "prob")
pred5 <- ROCR :: prediction(PD.pred5[,2], data_to_split.test$Class)
auc_value5 <- auc(data_to_split.test$Class, PD.pred5[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
## Setting direction: controls < cases
```

```
auc_value5
```

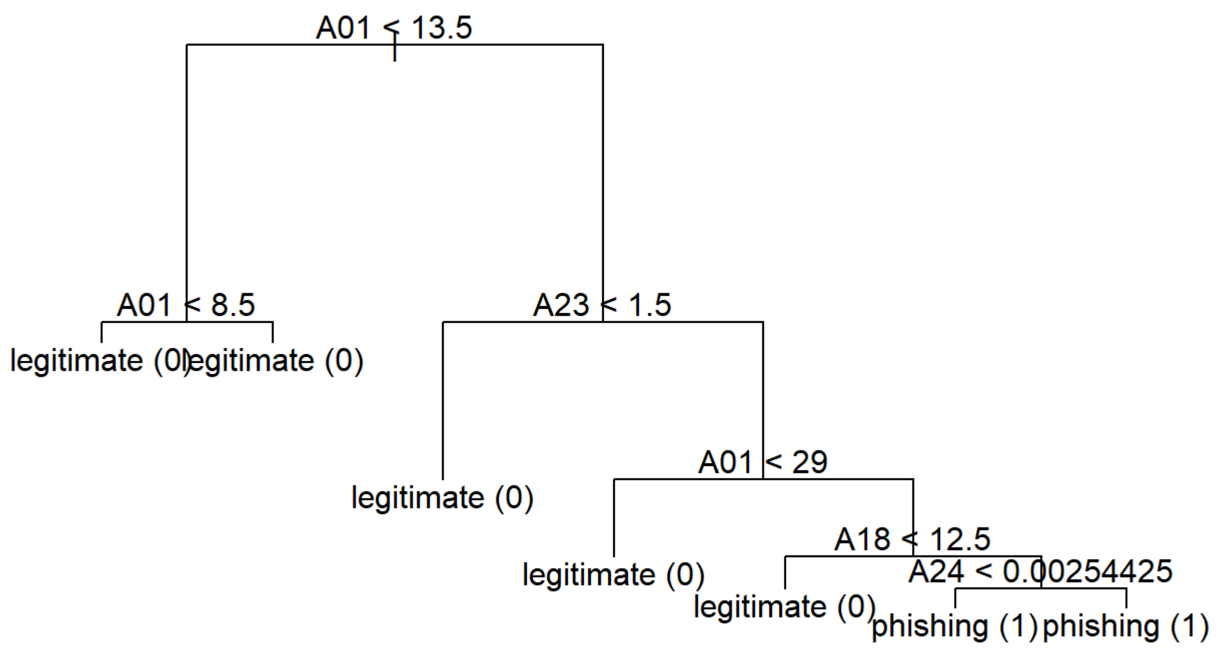
```
## Area under the curve: 0.8381
```

For a random forest model, the important predictors are “A01”, “A02”, “A04”, “A06”, “A08”, “A09”, “A10”, ‘A11’, “A12”, “A14”, “A15”, “A16”, “A17”, “A18”, “A19”, “A20”, “A22”, “A23”, and “A24”for the base model as all these predictors have an influence to the interpretability of the model with “A01” being the most important predictor with the highest MiniDecreaseGini score (Importance score). Since random forest is an ensemble decision tree model, the “MeanDecreaseGini” metrics, which is the average decrease in Gini impurity of the decision trees during feature splitting offers an insight on the importance of predictors as the higher the “MDG” score, the more the contribution of the predictor to the model. For the base model, the predictors with the highest importance to the model is “A01” with an importance of 108.0517946 and the predictors with a very low importance is “A21” with an importance of 0.5428806. The second model is created with all the predictors excluding the predictor with the lowest “MeanDecreaseGini” score, “A21”, which gives an accuracy higher than the base model (0.8522484) which is 0.8608137 with a slight decrease in AUC from 0.836 to 0.835. This is an interesting insight as it suggests that the predictor “A21” has impacted negatively to the accuracy of the model. This is due to redundancy of the variable “A21” where while not providing additional information to the model, it introduces noise to the model prediction capability, which also contributes to the overfitting of the base model. Thus, by removing the predictors “A21” with the lowest MDG score from the base model, which works by row sampling and feature selection can focus on the important predictors instead of the non-important predictors, which increases the accuracy of the second model. So instead of the base model, we will find the predictors to be omitted from the second model (without the “A21” predictor). For the second model, the predictors with the highest importance to the model is “A01” with an importance of 108.0517946 and the predictors with a very low importance is “A09” with an importance of 0.850857. To find the influence of omitted predictors on the accuracy first I omitted the variable “A09” to create a third model based on the second model (with predictors excluding “A21”) and observed a slight decrease in the accuracy of the model to 0.8543897, where the difference is 0.0085653 in comparison to the previous model. Although there is a slight increase in AUC from 0.836 to 0.8381, it has a less impact as the difference is significantly low. Thus, the predictors that we can remove from the base model with only a slight effect on the performance of the model is predictors “A21” and “A09”. This leaves the model with the predictors “A01”, “A02”, “A04”, “A06”, “A08”, “A10”, ‘A11’, “A12”, “A14”, “A15”, “A16”, “A17”, “A18”, “A19”, “A20”, “A22”, “A23” and “A24”.

7. A simple yet effective classifier

The classifier that I have chosen that is as simple as it could be to classify the site to be legitimate or phishing by hand is Decision Tree. Decision trees offers a high interpretability due to the simplicity of the model and allows a clear and intuitive decision to be made without the requirement of advanced knowledge. This is because a single Decision Tree model is a simple model that is created based upon the classification of class labels by splitting of predictors which makes the model the easiest to classify as the model does not require any complex computation. In addition, the decision tree model created in previously has a high accuracy and high AUC with lesser complexity, where the accuracy and AUC of the Decision Tree model is the 3rd highest in comparison to the other classifiers. Due to the simplicity, interpretability and ability to be drawn easily by hand and evaluated, I decided to choose decision tree as the model for classification by hand.

```
plot(ptfit)
text(ptfit, pretty = 0)
```



```
tpredict = predict(ptfit, data_to_split.test, type="class")

confusion_matrix = table(observed = data_to_split.test$Class, predicted = tpredict)
confusion_matrix
```

```
##                   predicted
## observed           legitimate (0) phishing (1)
##   legitimate (0)             313           21
##   phishing (1)                53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8415418
```

```
# Prune the model
testptfit = cv.tree(ptfit, FUN=prune.misclass)
testptfit
```

```
## $size
## [1] 7 5 4 3 1
##
## $dev
## [1] 152 150 149 189 316
##
## $k
## [1] -Inf    0   10   33   76
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

To simplify the decision tree, I have decided to use a technique called pruning on the decision tree model. The pruning technique that I will be using is the post-pruning where I will allow the decision tree model to create the best and biggest possible tree until the data is overfitted and prune the grown tree by replacing the sub-trees with respecting leaf nodes given the rate of error is minimized.

The decision tree is based upon the concept of split of predictors where at each split the goal is to increase the homogeneity of the resulting datasets with respect to the target variable (which we are trying to classify). Thus, the higher the homogeneity of the model the better the prediction accuracy is. So, in terms of pruning of the model to create a classifier that is simple for a person to classify by hand, we have to look into a trade-off between the complexity of the model and the accuracy of the model. Since the complexity depends on the number of predictors and splitting, this is also useful in the selection of attributes for the model. Thus, a successful pruning can give a model with the a slight decrease or same or better accuracy and AUC performance with lesser number of leaves and lesser predictors that contributors to the simplicity of the model.

To find the model, we will use a cross validation with a fun="prune.misclass" to prune the trees to N smaller trees. From the result of the pruning of tree with cv where the sizes of pruned trees are 7, 5, 4, 3, and 1, the pruned tree with size 7, is the initial base decision tree fitted, thus, ignoring that, I fitted a decision tree for the pruned tree of sizes 5, 4 and 3 as well as computing the accuracy and AUC of these models.

```
print("Pruned Model")
```

```
## [1] "Pruned Model"
```

```
prune.ptfit = prune.misclass(ptfit, best = 5)
print(summary(prune.ptfit))
```

```
##
## Classification tree:
## snip.tree(tree = ptfit, nodes = c(2L, 31L))
## Variables actually used in tree construction:
## [1] "A01" "A23" "A18"
## Number of terminal nodes:  5
## Residual mean deviance:  0.7206 = 781.2 / 1084
## Misclassification error rate: 0.1258 = 137 / 1089
```

```
tpredict2 = predict(prune.ptfit, data_to_split.test, type="class")

confusion_matrix2 = table(observed = data_to_split.test$Class, predicted = tpredict2)
print(confusion_matrix2)
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            313           21
##   phishing (1)               53           80
```

```
df2 = as.data.frame(confusion_matrix2)
accuracy = (df2[1,3] + df2[4,3]) / sum(df2$Freq)
cat("Accuracy: ", accuracy, "\n")
```

```
## Accuracy:  0.8415418
```

```
PD.pred.prune = predict(prune.ptfit, data_to_split.test, type = "vector")
pred.prune<- ROCR :: prediction(PD.pred.prune[,2], data_to_split.test$Class)
perf.prune <- performance(pred.prune,"tpr","fpr")
auc_value.prune <- auc(data_to_split.test$Class, PD.pred.prune[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.prune
```

```
## Area under the curve: 0.8092
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
summary(ptfit)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = data_to_split.train)
## Variables actually used in tree construction:
## [1] "A01" "A23" "A18" "A24"
## Number of terminal nodes:  7
## Residual mean deviance:  0.6854 = 741.6 / 1082
## Misclassification error rate: 0.1258 = 137 / 1089
```

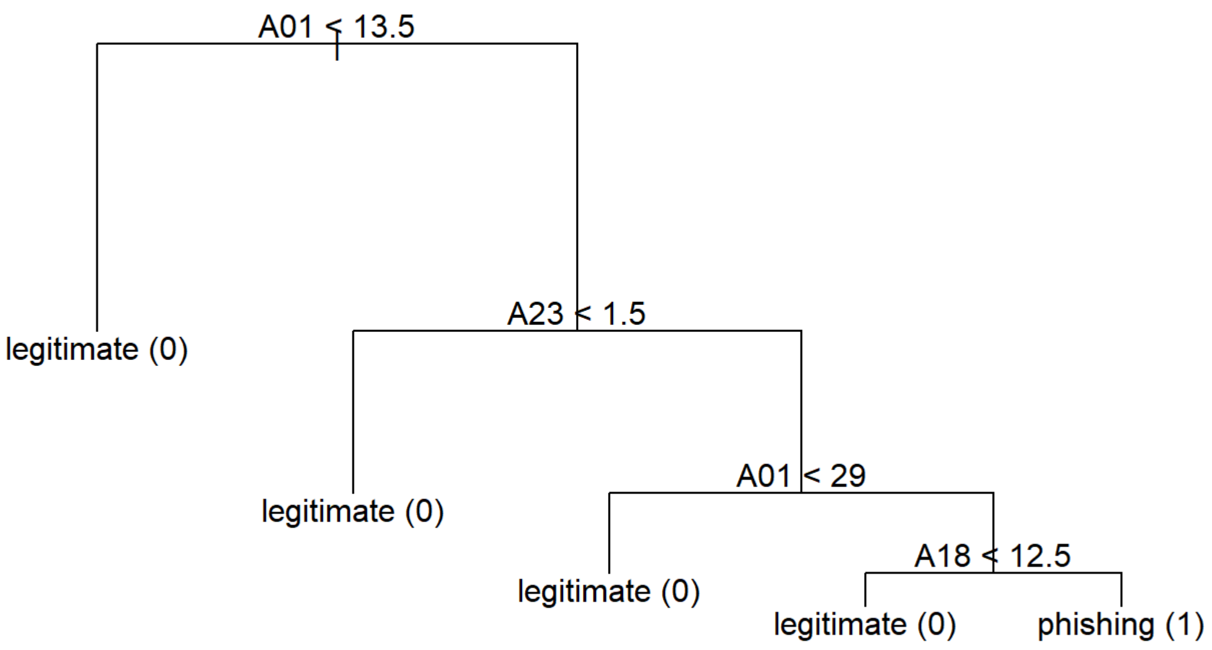
```
cat("Accuracy: ", 0.8415418 , "\n")
```

```
## Accuracy:  0.8415418
```

```
auc_value1  # 0.8163
```

```
## Area under the curve: 0.7529
```

```
plot(prune.ptfit)
text(prune.ptfit, pretty = 0)
```



```
prune.ptfit = prune.misclass(ptfit, best = 4)
print(summary(prune.ptfit))
```

```
##
## Classification tree:
## snip.tree(tree = ptfit, nodes = c(2L, 15L))
## Variables actually used in tree construction:
## [1] "A01" "A23"
## Number of terminal nodes:  4
## Residual mean deviance:  0.7505 = 814.3 / 1085
## Misclassification error rate: 0.135 = 147 / 1089
```

```
tpredict2 = predict(prune.ptfit, data_to_split.test, type="class")

confusion_matrix2 = table(observed = data_to_split.test$Class, predicted = tpredict2)
print(confusion_matrix2)
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            310           24
##   phishing (1)               53           80
```

```
df2 = as.data.frame(confusion_matrix2)
accuracy = (df2[1,3] + df2[4,3]) / sum(df2$Freq)
print(accuracy)
```

```
## [1] 0.8351178
```

```
PD.pred.prune = predict(prune.ptfit, data_to_split.test, type = "vector")
pred.prune<- ROCR :: prediction(PD.pred.prune[,2], data_to_split.test$Class)
perf.prune <- performance(pred.prune,"tpr","fpr")
auc_value.prune <- auc(data_to_split.test$Class, PD.pred.prune[,2])
```

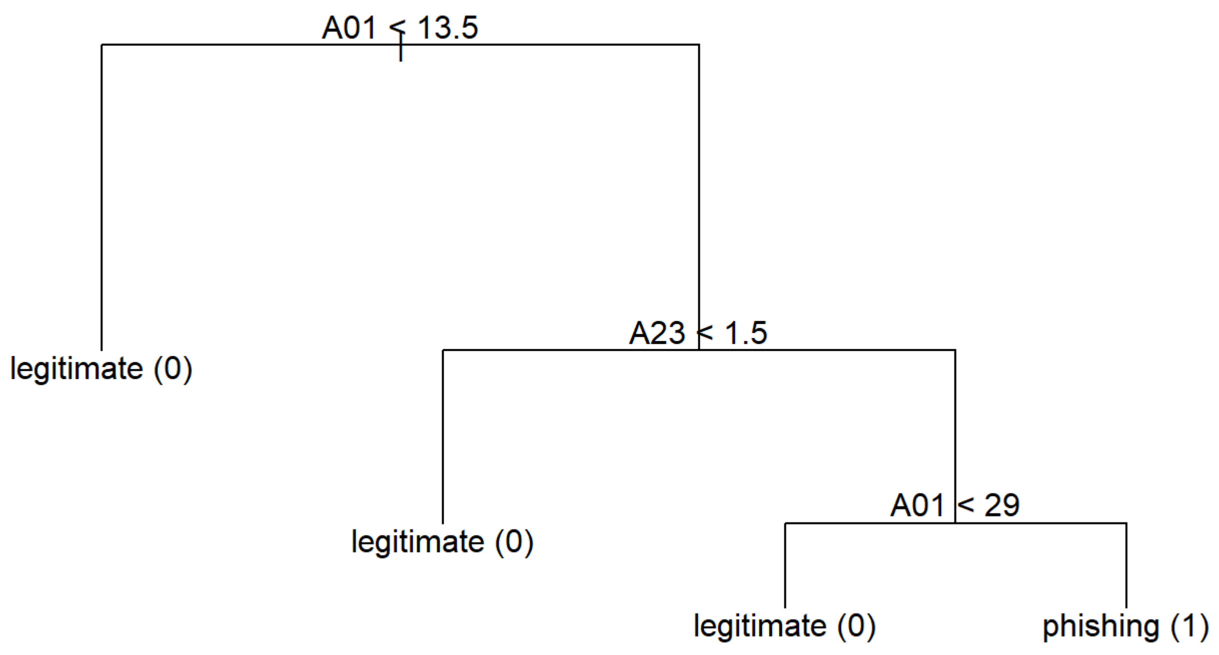
```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.prune
```

```
## Area under the curve: 0.8055
```

```
plot(prune.ptfit)
text(prune.ptfit, pretty = 0)
```



```
prune.ptfit = prune.misclass(ptfit, best = 3)
print(summary(prune.ptfit))
```

```
##
## Classification tree:
## snip.tree(tree = ptfit, nodes = c(2L, 7L))
## Variables actually used in tree construction:
## [1] "A01" "A23"
## Number of terminal nodes:  3
## Residual mean deviance:  0.8226 = 893.3 / 1086
## Misclassification error rate: 0.1653 = 180 / 1089
```

```
tpredict2 = predict(prune.ptfit, data_to_split.test, type="class")

confusion_matrix2 = table(observed = data_to_split.test$Class, predicted = tpredict2)
print(confusion_matrix2)
```

```
##                   predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            282           52
##   phishing (1)               39           94
```

```
df2 = as.data.frame(confusion_matrix2)
accuracy = (df2[1,3] + df2[4,3]) / sum(df2$Freq)
print(accuracy)
```

```
## [1] 0.8051392
```

```
PD.pred.prune = predict(prune.ptfit, data_to_split.test, type = "vector")
pred.prune<- ROCR :: prediction(PD.pred.prune[,2], data_to_split.test$Class)
perf.prune <- performance(pred.prune,"tpr","fpr")
auc_value.prune <- auc(data_to_split.test$Class, PD.pred.prune[,2])
```

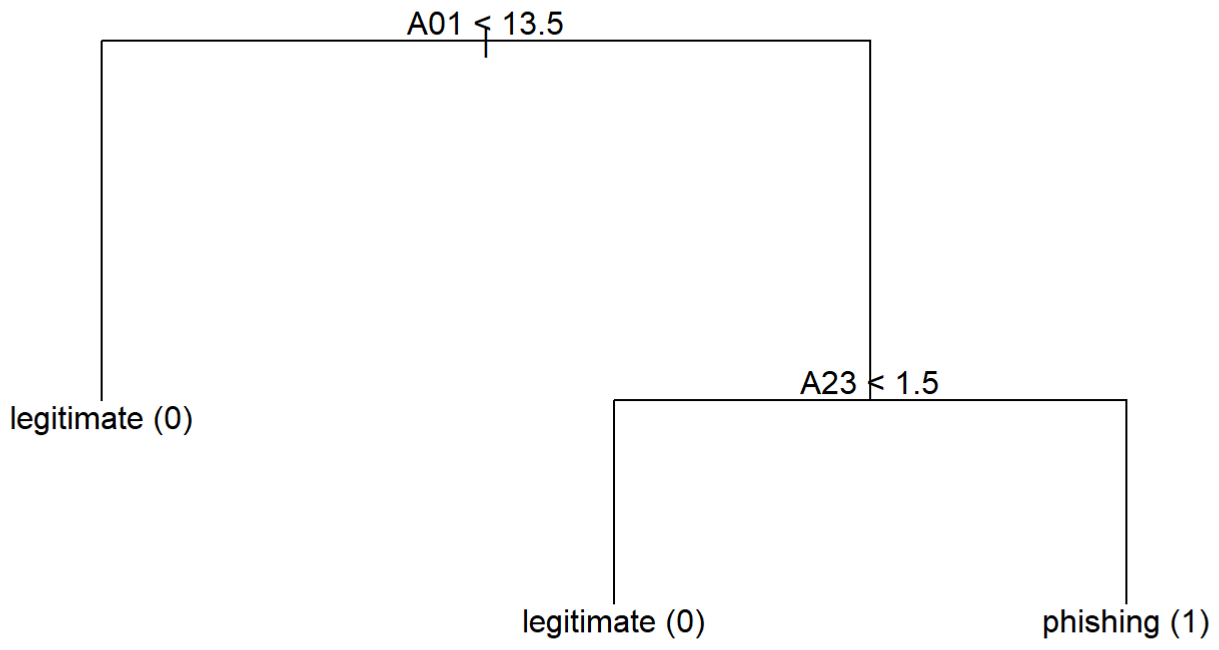
```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.prune
```

```
## Area under the curve: 0.7841
```

```
plot(prune.ptfit)
text(prune.ptfit, pretty = 0)
```



So, analysing the pruned model of N = 5, 4 and 3, I decide to take the model with N = 4 as the model that is simplistic enough to classify by hand and maintains the large portion of the accuracy with the attributes / predictors A01 and A23. This is because comparing N = 5 and N = 4, although the accuracy of N = 4 is smaller by a very small difference, but in terms of complexity N = 4 is significantly better as it uses 1 lesser predictor and comparing N = 4 and N = 3. same number of predictors with one lesser leaf in N = 3, but there is a significant drop in accuracy of approximately 3% which significantly affects the model performance. In comparison to N = 7, the chosen pruned model of N = 4 is significantly less complex as it has 2 less predictors to consider when finding the classification of a data and the model also has 3 lesser leaf nodes in comparison to the base model which has 7 leaves. We can also observe that for the model with N = 7, for the last two nodes in the left and right, the classification of the tree refers to the same classification group suggesting that there might have been an overfitting within the base tree model which can be solved by the model with N = 4. However, we can see a drop in accuracy and AUC for the model with N = 4 compared to the base model with N = 7, but the drop is significantly small where the difference is lesser than 1% which suggests that in terms of complexity and accuracy trade-off of the model, the pruned model with N = 4 is better as it is simple for a person to be able to classify the phishing and legitimate classes by hand and also maintain a relatively high accuracy and AUC even in comparison to the base model.

8. Best tree-based classifier

To create the best tree-based classifier, I decided to choose the random forest model since the base random forest model previously has a better accuracy and AUC that can be improved further for a better statistical analysis. Improving further the accuracy and AUC of this model will induce a better classification result and a better predictive model that can achieve a high-performance level. From previous tests, we have identified that the accuracy and AUC of the original random forest model is equals to 0.8522484 and 0.8358 respectively. To create the best random forest model, I have considered 2 factors which are predictor selection and parameter tuning. I decided to improve the base model based on the predictor selection first, then improve the performance of the model with parameter tuning.

```
fit.rf$importance
```

```
##     MeanDecreaseGini
## A01      108.0517946
## A02        4.7817396
## A04        5.7826599
## A06        4.5617499
## A08       20.9560813
## A09        0.9157853
## A10        2.4276015
## A11        1.3062086
## A12       17.4267966
## A14       21.2946927
## A15        4.6770651
## A16        2.1267430
## A17        9.1816711
## A18       55.9942373
## A19        4.4634655
## A20        6.0309938
## A21        0.5428806
## A22       46.2121171
## A23       93.7625471
## A24       21.4955196
```

REMOVE PREDICTOR A21

```
# set.seed(105)
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A09", "A10", 'A11', "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A
23", "A24", "Class")])

rf.pred2 <- predict(fit.rf2, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                 predicted
## observed         legitimate (0) phishing (1)
##   legitimate (0)            321           13
##   phishing (1)               52           81
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8608137
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.835
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

REMOVE PREDICTOR A09

```
# set.seed(105)
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", 'A11', "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A
24", "Class")])

rf.pred2 <- predict(fit.rf2, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            319           15
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8543897
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8381
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

REMOVE PREDICTOR A11

```
# set.seed(105)
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "C
lass")])


rf.pred2 <- predict(fit.rf2, data_to_split.test)


confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            319           15
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8543897
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8389
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

For the predictor selection process, I used the base model created for random forest initially to find the importance of the predictors using the “MeanDecreaseGini” score, where the lower the MDG score is the less important the predictor is. From the base model, the predictor “A21” has the lowest MDG score so by removing this predictor from the model, the accuracy of the model increased from 8522484 to 0.8608137 but the AUC is still lower than the base model (0.835). This shows that the model’s accuracy has a positive increase but no significant effect on the AUC. Next, I find the predictor with the lowest MDG score from the random forest model (with “A21” removed) and identified that the predictor “A09” has the lowest MDG score. By, removing this predictor, although there is an increase in the accuracy based on the base model, but based on the model (with “A21” removed) I identified that there is a slight decrease in accuracy from 8608137 to 0.8543897, but there is a significant increase in the AUC to 0.8381 which is higher than both the base model and the model with “A21” removed. Next, using the similar step, I found that for the model with “A21” and “A09” removed, the next predictor with the lowest MDG score is “A11” and by removing this variable, there is no increase in the accuracy but a significant increase in AUC is observed where the new AUC value is 0.8389. So, I have decided that due to the betterment of the model with “A21”, “A09” and “A11” removed on the accuracy and AUC in comparison to the base model, I came to a conclusion that the model that will be used for parameter tuning will be the model with the predictors “A21”, “A09” and “A11” removed, leaving the model with the important predictors, “A01”, “A02”, “A04”, “A06”, “A08”, “A10”, “A12”, “A14”, “A15”, “A16”, “A17”, “A18”, “A19”, “A20”, “A22”, “A23” and “A24”.

For the model tuning process, I have decided to use 6 parameters to tune which are ntree, mtry, nodesize, importance, localImp and oob.prob. Based on the parameters above, I created 3 fine-tuned models and calculated their accuracies for the model with predictors “A21”, “A11” and“A09” removed.

RANDOM FOREST PARAMETER TUNING 1

```
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "C
lass")], ntree=10000, mtry=5,nodesize = 3, importance=TRUE, localImp= TRUE, oob.prox=TRUE)

rf.pred2 <- predict(fit.rf2, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            320           14
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.856531
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8379
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

RANDOM FOREST PARAMETER TUNING 2

```
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "C
lass")], ntree=10000, mtry=5,nodesize = 4, importance=TRUE, localImp= TRUE, oob.prox=TRUE)

rf.pred2 <- predict(fit.rf2, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            320           14
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.856531
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8389
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

RANDOM FOREST PARAMETER TUNING 3

```
print("Improved RF")
```

```
## [1] "Improved RF"
```

```
set.seed(32885741) # Your Student ID is the random seed
fit.rf2 <- randomForest(Class ~ ., data=data_to_split.train[, c("A01", "A02", "A04", "A06",
"A08", "A10", "A12", "A14", "A15", "A16", "A17", "A18", "A19", "A20", "A22", "A23", "A24", "C
lass")], ntree=10000, mtry=5,nodesize = 6, importance=TRUE, localImp= TRUE, oob.prox=TRUE)

rf.pred2 <- predict(fit.rf2, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =rf.pred2)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            320           14
##   phishing (1)               53           80
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.856531
```

```
PD.pred.imp = predict(fit.rf2, data_to_split.test, type = "prob")
pred.imp <- ROCR :: prediction(PD.pred.imp[,2], data_to_split.test$Class)
perf.imp <- performance(pred.imp,"tpr","fpr")
auc_value.imp <- auc(data_to_split.test$Class, PD.pred.imp[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8401
```

```
cat("\n\n")
```

```
print("Original Model")
```

```
## [1] "Original Model"
```

```
print("Accuracy: 0.8522484")
```

```
## [1] "Accuracy: 0.8522484"
```

```
auc_value5  # 0.8384
```

```
## Area under the curve: 0.8381
```

Based on the predictor removing process, the best model that could have been created (with “A21”, “A09” and “A11” removed), the accuracy and AUC achieved are 0.8543897 and 0.8389 respectively. Using the parameter tuning, to improve the accuracy or/and AUC I used the parameter values as such: ntree=10000, mtry=5, nodesize=5, importance=TRUE, localImp= TRUE, oob.prox=TRUE, which increased the accuracy from 0.8542897 to 0. 856531 and increase the AUC from 0.8389 to 0.8401. Thus, I have obtained a better model from fine tuning with the knowledge on the improved model from predictor selection.

In conclusion, the new tree classifier created with “A21”, “A09” and “A11” predictors omitted and ntree=10000, mtry=5, nodesize=5, importance=TRUE, localImp= TRUE, oob.prox=TRUE, parameters fine-tuned improves the overall accuracy of the random forest from 0.8522484 to 0.856531 and AUC from 0.8357683 to 0.8401, where we see a significant improvement in the model. This improvement is due to removing insignificant predictors, which helps eliminate noise and prevents the model from learning incorrect patterns in the data. Furthermore, during fine-tuning of the model, by increasing the number of trees, the number of features sampled for each tree, and the minimum size of the terminal nodes, the random forest ensemble model creates multiple trees with multiple random set of features which ensures my model to be able to learn the non-linearity patterns in the data even better. Additionally, by enabling the calculation of variable importance, local importance, and out-of-bag (OOB) proximity, while creating multiple decision trees, my random forest model will take the importance and proximity of predictors into consideration which allows the significant predictors to contribute more to the predictive capability of the model. Although these adjustments adds additional complexity to the model, it also enhances the model’s predictive capabilities.

9. Classification of website with ANN

ANN is a deep learning model using perceptron’s and sophisticated structures with one or more hidden layers to discover complex relationships of predictors to compute the outcome of the network.

```
# Understanding nature of data

df = (data_to_split[, c(1,2,5,9,14,18,19,20)])
# Get minimum values for each column
min_values <- sapply(df, min)

# Get maximum values for each column
max_values <- sapply(df, max)

# Display the results
round(min_values, 2)
```

```
##   A01   A02   A08   A12   A18   A22   A23   A24
##  1.00  0.00  0.16 48.00  4.00  0.01  0.00  0.00
```

```
round(max_values, 2)
```

```
##     A01     A02     A08     A12     A18     A22     A23     A24
##   48.00  135.00    1.00  692.00 2082.00    0.08  832.00    0.52
```

```
# 5 no need, 18 no need, 20 no need
```

DATA PRE-PROCESSING

For the pre-processing of the data for ANN, since I have identified that the predictors “A21”, “A09” and “A11” does not provide meaningful information for the analysis (based on the improved random forest model), I have decided to omit these predictors for the subsequent analysis for ANN model.

Next since some integer predictors were converted to factors (predictors with <10 categories), these predictors need to be converted back to numerical values as ANN which typically works with numerical data’s is unable to handle or interpret predictors encoded as factor or character datatype. So, to convert the categorical predictors to numerical outcomes I used the model.matrix() function on all the factor variables. This function will convert all the categorical variables into numerical dummy variables where each category in the predictor gets a column of its own excluding the first category (as the values for other categories are relatively based upon the first category) and the values are determined by the existence of the category in each rows (if the category is present then 1, else 0). Following that I also normalized some of the numerical columns using the “scale” function, namely the columns “A01”, “A02”, “A12”, “A18”, “A23”. This was due to the scale of the data being uneven and not within the range of 0 and 1. This might induce some bias and incorrect output in the performance of the model.

Thus, to get the training and testing data, I will combine the outcome of the model.matrix() contrast variables with the numerical data (“A08”, “A22”, “A24”) and the normalized data which will be used for model training and model testing for a ANN.

```
# data contrast
ptmm = model.matrix(~A04+A06+A10+A14+A15+A16+A17+A19+A20, data=data_to_split.train)

# data contrast
scaled = as.data.frame((scale(data_to_split.train[, c(1,2,9,14,19)])))

# numerical data
other = as.data.frame(((data_to_split.train[, c(5,18,20)])))

# combine data
ptcomb = cbind(scaled, other, ptmm)
ptcomb$'(Intercept)' = NULL
train_data = cbind(ptcomb, Class = data_to_split.train$Class)
train_data
```

	A01 <dbl>	A02 <dbl>	A12 <dbl>	A18 <dbl>	A23 <dbl>	A08 <dbl>	
80923	1.0406324	-0.07299028	-0.601068722	0.516894787	-0.9610902736	1.0000000	0.0
87866	-0.7452519	-0.07299028	-0.601068722	-0.232830769	0.8057258575	1.0000000	0.0
96752	1.3032624	-0.07299028	-0.635685977	-0.322350238	-1.4027943063	0.8780488	0.0
70437	1.3557884	-0.07299028	-0.061039535	-0.490199243	-1.4027943063	1.0000000	0.0
53181	-1.0604080	-0.07299028	-0.601068722	0.427375318	0.6520896722	1.0000000	0.0
9013	0.9355803	-0.07299028	-0.601068722	-0.031411963	-0.9034767041	1.0000000	0.0
21132	-0.2199919	0.58211754	-0.601068722	0.516894787	-1.1147264589	1.0000000	0.0
50671	1.3032624	-0.07299028	-0.601068722	-0.064981764	-0.7882495651	1.0000000	0.0
3918	1.3557884	-0.07299028	-0.601068722	-0.333540172	-1.4027943063	0.7727273	0.0
91323	1.3032624	-0.07299028	-0.601068722	1.177100874	-0.0008641154	1.0000000	0.0

1-10 of 1,089 rows | 1-8 of 26 columns Previous **1** 2 3 4 5 6 ... 109 Next

```
# data contrast
ptmm = model.matrix(~A04+A06+A10+A14+A15+A16+A17+A19+A20, data=data_to_split.test)

# data contrast
scaled = as.data.frame((scale(data_to_split.test[, c(1,2,9,14,19)]))) # normalize

# numerical data
other = as.data.frame(((data_to_split.test[, c(5,18,20)]))) # dont normalize because range wi
thin 0 and 1

# combine data
ptcomb = cbind(scaled, other, ptmm)
ptcomb$'(Intercept)' = NULL
test_data = cbind(ptcomb, Class = data_to_split.test$Class)
test_data
```

	A01	A02	A12	A18	A23	A08	
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<
48070	1.1811453	-0.1608144	1.63568843	0.542243020	-0.843858494	1.0000000	0.05054
17031	-0.6044767	-0.1608144	-0.64249210	-0.312282825	0.402339892	1.0000000	0.05790
93058	1.4516941	-0.1608144	-0.64249210	-0.402232913	-0.702245041	1.0000000	0.06184
41325	-0.8750255	-0.1608144	-0.64249210	0.209427691	0.558114690	1.0000000	0.05498
49212	-0.6044767	-0.1608144	-0.64249210	-0.267307780	0.487307963	1.0000000	0.06734
98072	1.4516941	-0.1608144	-0.64249210	0.461287940	0.048306259	1.0000000	0.05770
40835	1.1811453	-0.1608144	-0.64249210	0.695158171	0.458985273	0.8095238	0.04559
26296	1.0729258	-0.1608144	-1.05486306	-0.366252878	-0.858019839	1.0000000	0.05549
6211	1.5058039	-0.1608144	1.19627675	-0.312282825	0.331533165	1.0000000	0.05968
48982	-0.6044767	-0.1608144	2.16974261	0.974003446	0.954632358	0.8333333	0.04774

1-10 of 467 rows | 1-9 of 26 columns Previous **1** 2 3 4 5 6 ... 47 Next

ANN MODEL

```
set.seed(32885741)
net <- neuralnet(Class~., train_data, hidden = 10, stepmax=100000, threshold=0.2)
plot(net)
```

```
comp = predict(net, test_data)
data = data_to_split
labels <- c('legitimate (0)', 'phishing (1)')

# Convert prediction probabilities to a data frame
comp_df <- as.data.frame(comp)

# Get the prediction labels
prediction_label <- comp_df %>%
  mutate(pred = labels[max.col(.)]) %>%
  pull(pred)  # pull(pred) is used to extract the pred column as a vector

confusion_matrix = table(observed = data_to_split.test$Class, predicted =prediction_label)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            279           55
##   phishing (1)               42           91
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy  # 0.7773019
```

```
## [1] 0.7922912
```

```
pred.prob.auc <- comp[, 1]
auc_value.imp <- auc(data_to_split.test$Class, pred.prob.auc)
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls > cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.7962
```

For an ANN model, I created the ANN model with the selected predictors with the parameters, hidden = 10, stepmax=100000, threshold=0.2. This gave me an accuracy of 0.7923 and an AUC of 0.796. Upon further fine-tuning with different sizes of hidden layers and varied threshold's, I was unable to obtain any better model representation, so I will use this model for the analysis.

From the accuracy and AUC of the neural network model (ANN), we can see that in terms of accuracy, the ANN model is better than naïve bayes but not as good as the decision tree, boosting, bagging, random forest, the pruned decision tree or the improved random forest. But in terms of AUC, the ANN model performs significantly the worst in comparison to all the other models. Naïve bayes model uses probability calculation based on assumptions that the predictors are independent to each other. Since there is a potential chance that the predictors might be affected by each other (non-independent), the accuracy of the Naïve bayes is impacted highly in comparison to ANN. In addition to that, since the predictors in this analysis have exhibited complex and non-linear relationship, the Naïve bayes model is unable to interpret these complexities very well, which is not the case for ANN as the hyperparameters for ANN model have been tuned to understand the complex relationships in the data

There are a few potential reasons on why the neural network model is unable to perform on par or better than the other tree classifiers, including the pruned and improved models in terms of Accuracy and AUC. Firstly, models such as decision tree, bagging, boosting and random forest requires a significantly smaller input in comparison to a ANN. This is mainly due to the simplicity of the models as models like decision tree, boosting and random forest typically uses a simpler approach to find the complex patterns in the data where decision tree uses the splitting of variables based on information gain, and ensemble methods such as bagging, boosting and random forest computes the model by row sampling, assigning weights to the hard classification predictors and selective features respectively with a decision tree model as the base structure. These mechanisms require less data to achieve a reasonable level of performance. As for the ANN, they are a highly flexible model that is designed to capture the non-linear relationship between features by iteratively adjusting weights on connections between neurons in the network using backpropagation. This comes with a cost in terms of the requirement of the size of the data as a large amount of data is needed to identify these intricate patterns of the data.

The next potential reason might be due to the large number of hyperparameters in an ANN. Since, ANN is a flexible model with the potential of tuning many hyperparameters which are number of neurons per layer, learning rate, threshold, activation functions to name a few, this results in multiple potential combination of hyperparameters to be tuned for an ANN model. Unfortunately, since training an ANN model comes with a high computational cost and computational time as well as the weights of the ANN model not yielding any meaningful values, it is difficult to identify the combinations of parameters that can yield the best potential accuracy and AUC.

10. Classification of website with SVM

“Support Vector Machine” (SVM) is a method in machine learning that is supervised and used for either classification or regression tasks. The training variables are grouped into different categories for classification, and SVMs can classify them separately in different spatial locations. The goal of the algorithm is to increase the margin surrounding the hyperplane while maintaining equilibrium between both sides for optimal data separation. As our SVM model will be utilized for classification purposes, it works by finding a hyperplane that reduces classification errors and maximizes the margin between different classes.

SVM algorithm generally uses a set of mathematical problems called kernels, which convert the original data by assessing the similarity between data point pairs in the feature space and projecting it into a higher-dimensional space to determine a hyperplane that separates the classes in the data but due to the expense of using kernel methods in computations the “svm” in “e1071” package in r uses, a kernel trick to avoid the necessity of explicitly mapping the input data into a higher dimensional feature space during the training of linear learning algorithms. This is helpful when working with nonlinear functions or decision boundaries. In other words, the kernel trick is a mathematical method employed in SVM to convert nonlinear input data into a higher-dimensional feature space. The kernel trick alters the dot product of the mapping function by replacing it with the dot product of the support vectors. One way to achieve this is by identifying an appropriate kernel that can be substituted for the inner product of mapping functions.

The selection of kernel functions has a big impact on the effectiveness of SVMs with kernel trick. Despite the availability of alternative kernel options like linear, polynomial, and sigmoid, due to the similar optimal performance of the radial kernel in terms of accuracy and AUC, I have chosen to utilize the “radial” kernel. The “radial” kernel computes the similarity or closeness of two points by evaluating distance. A nonlinear kernel transforms the input space into a higher-dimensional feature space, enabling the segregation of data points that cannot be separated linearly in the original space.

```
set.seed(32885741)
# Set kernel type (options: linear, polynomial, radial)
svm_model <- e1071 :: svm(Class ~ ., data = data_to_split.train, kernel = "radial", probabili
ty = TRUE)

svm.pred <- predict(svm_model, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =svm.pred)
confusion_matrix
```

```
##                   predicted
## observed           legitimate (0) phishing (1)
##   legitimate (0)            310           24
##   phishing (1)               57           76
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8265525
```

```
svm.pred.prob <- predict(svm_model, data_to_split.test, probability = TRUE)
svm.pred.prob <- attr(svm.pred.prob,"probabilities")
auc_value.imp <- auc(data_to_split.test$Class, svm.pred.prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8207
```

```
set.seed(32885741)

# Set kernel type (options: linear, polynomial, radial)
svm_model <- e1071 :: svm(Class ~ ., data = data_to_split.train, kernel = "radial", gamma =
0.1, probability = TRUE)

svm.pred <- predict(svm_model, data_to_split.test)

confusion_matrix = table(observed = data_to_split.test$Class, predicted =svm.pred)
confusion_matrix
```

```
##                  predicted
## observed          legitimate (0) phishing (1)
##   legitimate (0)            316           18
##   phishing (1)               60           73
```

```
df = as.data.frame(confusion_matrix)
accuracy = (df[1,3] + df[4,3]) / sum(df$Freq)
accuracy
```

```
## [1] 0.8329764
```

```
svm.pred.prob <- predict(svm_model, data_to_split.test, probability = TRUE)
svm.pred.prob <- attr(svm.pred.prob,"probabilities")
auc_value.imp <- auc(data_to_split.test$Class, svm.pred.prob[,2])
```

```
## Setting levels: control = legitimate (0), case = phishing (1)
```

```
## Setting direction: controls < cases
```

```
auc_value.imp
```

```
## Area under the curve: 0.8217
```

To use the SVM model in R, we will be using the “e1071” package in R for the implementation of the “svm” function. The code used to generate the svm model is “e1071 :: svm(Class ~ ., data = data_to_split.train, kernel =”radial”)”. Here I am calling the svm function on the training dataset with the kernel “radial”. The accuracy obtained for the base SVM model is 0.8266 and the AUC is 0.8207. To fine tune the model, after multiple tests, I observed that by changing the “gamma” parameter (controls the trade-off between model complexity and model performance) to 0.1, the model’s accuracy increased to 0.833 while the AUC also increased to 0.8217. Thus, in terms of accuracy an accuracy of 83.3% and AUC of 0.8217 for a basic model of SVM is relatively high indicating the model is good enough in predicting the classification of the classes in the target variable.

In comparison to other tree-based models, in terms of accuracy, SVM is better than the Boosting ensemble model (0.817987) but does not reach the same height of accuracy as other tree classifiers such as bagging, random forest, decision tree, pruned decision tree and improved SVM. In terms of AUC, in comparison to other tree-based models, SVM has a better AUC than Decision Tree and Bagging but a slightly lower AUC than Boosting and Random Forest. Even so, we can see that the accuracy of the SVM model of 0.833 and AUC of 0.8217 is not very far off the accuracy of the other more accurate models which shows the predictive capability in terms of accuracy and AUC of the SVM model. This suggests that although the tree models are based on variable splitting and SVM is based on fitting a hyperplane in high-dimensional space to determine the classification, their performance are high enough to be considered a good predictive model for the classification of legitimate and phishing sites.

In comparison to the ANN and Naïve Bayes, we can clearly see a significant betterment in terms of accuracy and AUC for the SVM in comparison to these models. In terms of ANN and SVM, ANN requires extremely large dataset to obtain the optimal performance for classification to learn the patterns of the predictors efficiently which is not the case in terms of SVM, SVM which uses hyperplanes to divide and classify the models only requires a sufficient size of dataset to capture the pattern as it learns from the closest data points to the decision boundary. Moreover, ANN requires a complex finetuning which might increase the computational cost extensively which is not the case in SVM as the finetuning is much simpler and more efficient. In terms of Naïve Bayes and SVM, SVM is often better than Naive Bayes because it does not assume independence between features, allowing it to capture complex relationships between them by maximizing the margin between different classes. In contrast, Naive Bayes assumes that all features are independent of each other given the class label, meaning the presence or absence of one feature does not affect the presence or absence of another feature, conditioned on the class.