Predictive Modeling for Website Legitimacy

2024-04-23

Predicting Modelling For Website Legitimacy: Differentiating Between Legitimate Websites and Phishing sites.

By Angus Liang

Phishing websites pose a significant threat to online security, as they aim to deceive users into disclosing sensitive information, such as login credentials, financial details and other private user data.

The objective for this report is to create robust predictive models capable of identifying legitimate or Phishing websites. By utilising machine learning algorithms, this paper will leverage accurate classifiers that can distinguish between the two, therefore improving measures to protect users from online fraud.

Creating the data set

This is a modified version of the PhiUSIIL Phishing data, hosted by the UCI Machine Learning Archive. The references is in the appendix at the conclusion of this report.

```r
rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(31484808)
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
```

```r
head(PD)
```

```
##       A01 A02 A03 A04 A05 A06 A07       A08 A09 A10 A11 A12 A13 A14 A15 A16 A17
## 20699  17   0   0   3   0   1   0 0.3333333   0   0   0 133   0   0   0   0   1
## 42132  18   0   0   3   0   0   0 1.0000000   0   0   0 232   0   1   0   0   1
## 63979  30   0   0   3   0   0   0 1.0000000   0   0   0 232   0   0   0   0   1
## 5222   48   0   0   3   0   0   0 0.4285714   0   0   0 133   0   0   1   0   1
## 85232  30   0   0   3   0   1   0 0.6111111   0   0   0 504   0   0   0   0   1
## 221    28   0   0   2   0   0   0 0.5121951   0   0   0 377   0   0   0   0   1
##       A18 A19 A20 A21        A22 A23       A24 A25 Class
## 20699 786   0   1   0 0.05495738 100 0.0015020   0     0
## 42132  96   1   1   0 0.06468293 120 0.5229071   0     1
## 63979  15   0   1   0 0.06532096  53 0.5229071   0     0
## 5222    9   0   0   0 0.04956784   0 0.0015020   0     0
## 85232   9   0   0   0 0.04236572 107 0.0799628   0     0
## 221    12   0   0   0 0.05559717   4 0.0129268   0     1
```

Data Exploration and Pre-processing

First let, us explore the proportions of phishing websites to legitimate websites. Where

- 1 indicates a phishing website
- 0 indicates a legitimate website

```
count <- table(PD$Class)
proportions <- prop.table(count)
print(proportions)
```

```
##
##      0      1
## 0.6565 0.3435
```

There are 65.65% of legitimate websites and 34.35% of phishing websites in this sample.

Let us now explore the data

```
str(PD)
```

```
## 'data.frame':    2000 obs. of  26 variables:
##  $ A01  : int  17 18 30 48 30 28 13 48 30 30 ...
##  $ A02  : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ A03  : int  0 0 0 0 0 0 0 NA 0 0 ...
##  $ A04  : int  3 3 3 3 3 2 2 2 3 3 ...
##  $ A05  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A06  : int  1 0 0 0 1 0 1 0 0 0 ...
##  $ A07  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A08  : num  0.333 1 1 0.429 0.611 ...
##  $ 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  133 232 232 133 504 377 360 304 NA 232 ...
##  $ A13  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A14  : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ A15  : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ A16  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A17  : int  1 1 1 1 1 1 1 2 1 1 ...
##  $ A18  : int  786 96 15 9 9 12 53 47 60 97 ...
##  $ A19  : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ A20  : int  1 1 1 0 0 0 0 NA 0 0 ...
##  $ A21  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ A22  : num  0.055 0.0647 0.0653 0.0496 0.0424 ...
##  $ A23  : int  100 120 53 0 107 4 116 92 22 115 ...
##  $ A24  : num  0.0015 0.5229 0.5229 0.0015 0.08 ...
##  $ A25  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Class: int  0 1 0 0 0 1 0 1 1 0 ...
```

Notably, website attribute with real valued attributes are A01, A04, A08, A12, A17, A18, A22, A23 and A24. The rest of the website attributes seemingly contains binary outcomes. Before finding statistical summary of those with real valued attributes, let us explore other attributes with proportions.

```r
counts <- table(PD[c('A02')])
print(counts)
```

```
## A02
##    0    1    2    3    4    5    6    7    8   18   36   64  105  157
## 1877   73   15    4    2    1    2    2    1    1    1    1    1    1
```

```r
counts <- table(PD[c('A03')])
print(counts)
```

```
## A03
##    0    1
## 1976    5
```

```r
counts <- table(PD[c('A05')])
print(counts)
```

```
## A05
##    0    1    3    4    8   12   14
## 1976    3    1    1    1    2    1
```

```r
counts <- table(PD[c('A06')])
print(counts)
```

```
## A06
##    0    1
## 1731  251
```

```r
counts <- table(PD[c('A07')])
print(counts)
```

```
## A07
##    0    1
## 1982    5
```

```r
counts <- table(PD[c('A09')])
print(counts)
```

```
## A09
##    0    1
## 1925   53
```

- Interestingly attribute A02 contains mostly 0s and 13 other unique non-zero values.
- Attribute A03 has strange proportions with 99.99..% of response is 0s. This column will be omitted.
- A05, similar to A02 contains mostly 0s with 6 other unique non-zero values. However only 9 websites contain differing values out of 2000. This is strange proportions and the attribute will be omitted
- A06 is 13% to 87% split of 1s and 0s.

- A07, similar to A03 has extreme proportions, and therefore will be omitted.
- A09 has only 53 1s response. It may be considered to be omitted.

Let us continue the exploration

```
counts <- table(PD[c('A10')])
print(counts)
```

```
## A10
##    0    1
## 1885   93
```

```
counts <- table(PD[c('A11')])
print(counts)
```

```
## A11
##    0    1    2    3    4    5    6    7    8    9   11
## 1928   26    4    7    1    4    1    1    2    1    1
```

```
counts <- table(PD[c('A13')])
print(counts)
```

```
## A13
##    0    3    9   12
## 1973    2    1    2
```

```
counts <- table(PD[c('A14')])
print(counts)
```

```
## A14
##    0    1
## 1756  220
```

```
counts <- table(PD[c('A15')])
print(counts)
```

```
## A15
##    0    1
## 1722  256
```

```
counts <- table(PD[c('A16')])
print(counts)
```

```
## A16
##    0    1
## 1892   82
```

```
counts <- table(PD[c('A19')])
print(counts)
```

```
## A19
##    0    1
## 1812  172
```

```
counts <- table(PD[c('A20')])
print(counts)
```

```
## A20
##    0    1
## 1485  493
```

- A10 has only 93 1s response compared to 1885 0s.
- A11 has 1928 zeros, with 10 other unique values. This will be omitted
- A13 has 1973 zeros, with 3 other unique values. This will be omitted
- A14 has 1756 0s with 220 1s
- A15 has 1722 zeros and 256 1s
- A16 has 1892 zeros with 82 1s. It may be considered to be omitted
- A19 has 1812 0s and 172 1s. It may be considered to be omitted
- A20 has 1485 0s and 493 1s.

Let us finally explore the last seemingly binary outcomes attributes

```
counts <- table(PD[c('A21')])
print(counts)
```

```
## A21
##    0    1    2    3    4
## 1934   46    2    3    1
```

```
counts <- table(PD[c('A25')])
print(counts)
```

```
## A25
##     0 0.053 0.056  0.06  0.08
##  1973     1     2     1     1
```

- A21 has 1934 0s response, with 4 other unique non zero values. It will be omitted
- A25 has 1973 0s and 5 other nonzero values. This will be omitted.

Let us now explore the summary statistics for the real-valued attribute A01, A04, A08, A12, A17, A18, A22, A23 and A24

```r
summary_statistics <- function(x){
    c(mean = mean(x, na.rm = TRUE),
    sd = sd(x, na.rm = TRUE),
    median = median(x, na.rm = TRUE),
    min = min(x, na.rm = TRUE),
    max = max(x, na.rm = TRUE))
}

sapply(PD[c('A01', 'A04', 'A08', 'A12', 'A17', 'A18', 'A22', 'A23', 'A24')], summary_statistics)
```

```
##             A01       A04       A08      A12       A17        A18         A22
## mean   22.87900 2.7529055 0.8487286 320.9152 1.1897773   61.74181 0.055911009
## sd     12.79595 0.5216112 0.2158716 143.5487 0.6235194  102.27286 0.009905865
## median 25.00000 3.0000000 1.0000000 232.0000 1.0000000   34.00000 0.057939903
## min     5.00000 2.0000000 0.1750000  48.0000 0.0000000    4.00000 0.014815335
## max    48.00000 6.0000000 1.0000000 692.0000 5.0000000 2082.00000 0.080636638
##              A23       A24
## mean    71.11788 0.2685232
## sd      59.93157 0.2508276
## median 100.00000 0.0799628
## min      0.00000 0.0000000
## max    886.00000 0.5229071
```

Notably, there are multiple missing values (NA) in each attribute except A01. There are seemingly outliers in some attributes such as A01.

Let us now pre-process the data before modelling.

Remove websites with missing values.

```r
PD_clean <- na.omit(PD)
dim(PD_clean)
```

```
## [1] 1568   26
```

Approximately 21.6% of websites in the sample has been removed.

Let us now remove attributes that will not be contained in model fitting. This is due to poor data quality, with value distributions that is likely not necessary to improve performance of classification of phishing sites. These website attributes are: A03, A05, A07, A11, A13, A21 and A25.

```r
attributes_removal <- c('A03', 'A05', 'A07', 'A11', 'A13', 'A21', 'A25')
PD_clean <- PD_clean[, !names(PD) %in% attributes_removal]
dim(PD_clean)
```

```
## [1] 1568   19
```

Although there are some attributes considered to be removed, further optimisations in model's performance in classification will determine if this step is necessary or not. I will further note future data transformation for performance optimisations.

Training and Testing Data for Modelling

Dividing the PD data into a 70% training and 30% test set

```r
set.seed(31484808)
PD_clean$Class = as.factor(PD_clean$Class)
# Split 70 30
train.row = sample(1:nrow(PD_clean), 0.7*nrow(PD_clean))
PD_clean.train = PD_clean[train.row,]
PD_clean.test = PD_clean[-train.row,]
```

Modelling

In this section, we are implementing various classification models.

Let us first load all the packages.

```r
suppressWarnings({
 library(knitr, quietly = TRUE)
 library(tree, quietly = TRUE)
 library(e1071, quietly = TRUE)
 library(ROCR, quietly = TRUE)
 library(randomForest, quietly = TRUE)
 library(adabag, quietly = TRUE)
 library(rpart, quietly = TRUE)
})
```

```
## randomForest 4.7-1.1

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

##
## Attaching package: 'ggplot2'

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

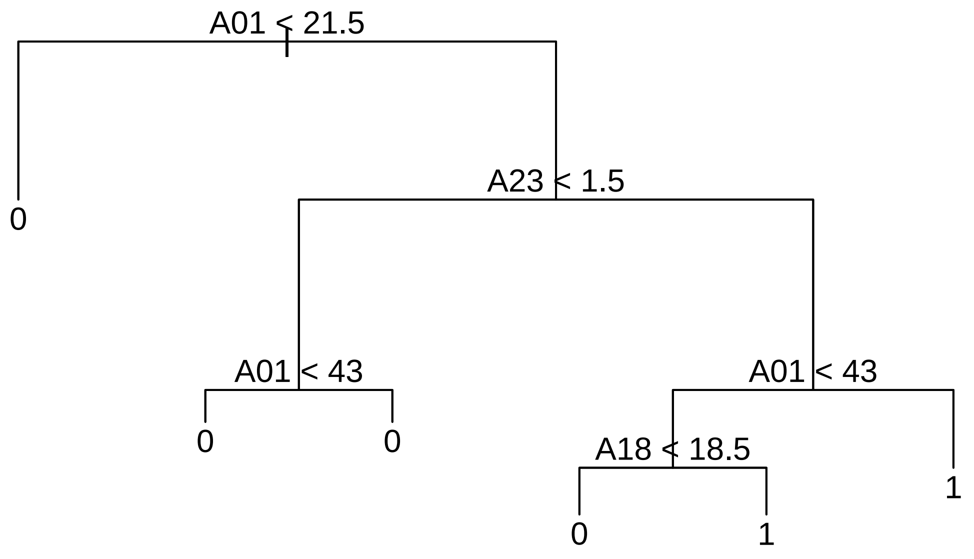
```r


PD.tree = tree(Class~., data = PD_clean.train)
summary(PD.tree)
```

i. Decision Tree

```
##
## Classification tree:
## tree(formula = Class ~ ., data = PD_clean.train)
## Variables actually used in tree construction:
## [1] "A01" "A23" "A18"
## Number of terminal nodes:  6
## Residual mean deviance:  1.065 = 1162 / 1091
## Misclassification error rate: 0.258 = 283 / 1097
```

```r
plot(PD.tree)
text(PD.tree, pretty = 0)
```



Using the test data, let us create a confusion matrix of this decision tree model; classifying 'phishing (1)' and 'legitimate (0)' websites

```r
PD.predtree = predict(PD.tree, PD_clean.test, type = 'class')

t1 = table(Predicted_Class = PD.predtree, Actual_Class = PD_clean.test$Class)
print(t1)
```

```
##                Actual_Class
## Predicted_Class   0   1
##               0 257  69
##               1  53  92
```

From this decision tree performance, the classification accuracy is **74.10**%

```r
# Create Bayes model
PD.bayes = naiveBayes(Class ~. , data = PD_clean.train )
```

ii. Naive Bayes Using the test data, let us create a confusion matrix of this Naive Bayes model; classifying 'phishing (1)' and 'legitimate (0)' websites

```r
# Predictions and Confusion matrix
PD.predbayes = predict(PD.bayes, PD_clean.test)
t2 = table(Predicted_Class = PD.predbayes, Actual_Class = PD_clean.test$Class)
print(t2)
```

```
##                Actual_Class
## Predicted_Class   0   1
##               0 110  32
##               1 200 129
```

From this Naive Bayes classification performance, the accuracy = **50.74**%

```r
# Create Bagging Model
PD.bag = bagging(Class ~., data = PD_clean.train, mfinals = 5)
```

```r
PDpred.bag = predict.bagging(PD.bag, PD_clean.test)
# Predicted Class from Bagging Model
pred_class = as.factor(PDpred.bag$class)
# Confusion Matrix
t3 = PDpred.bag$confusion
t3
```

iii. Bagging

```
##                Observed Class
## Predicted Class   0   1
##               0 277  84
##               1  33  77
```

As shown, the classification accuracy of the Bagging Model is **75.15**%

```r
# Create Boosting Model
PD.Boost = boosting(Class~. , data = PD_clean.train, mfinal=10)
```

```r
# Prediction and Confusion Matrix
PDpred.boost <- predict.boosting(PD.Boost, newdata=PD_clean.test)
t4 = PDpred.boost$confusion
t4
```

Boosting

```
##                Observed Class
## Predicted Class   0   1
##               0 274  84
##               1  36  77
```

As shown, the classification accuracy of the Boosting Model is **74.52**%

```r
# Create Model
set.seed(31484808)
PD.rf = randomForest(Class ~. , data = PD_clean.train, na.action = na.exclude)
```

```r
# Model prediction and confusion matrix
PDpredrf = predict(PD.rf, PD_clean.test)
t5 = table(Predicted_Class = PDpredrf, Actual_Class = PD_clean.test$Class)
t5
```

Random Forest

```
##                Actual_Class
## Predicted_Class   0   1
##               0 282  90
##               1  28  71
```

As shown, the classification accuracy of the Random Forest Model is **74.73**%

ROC Curve and AUC score for all Classifiers

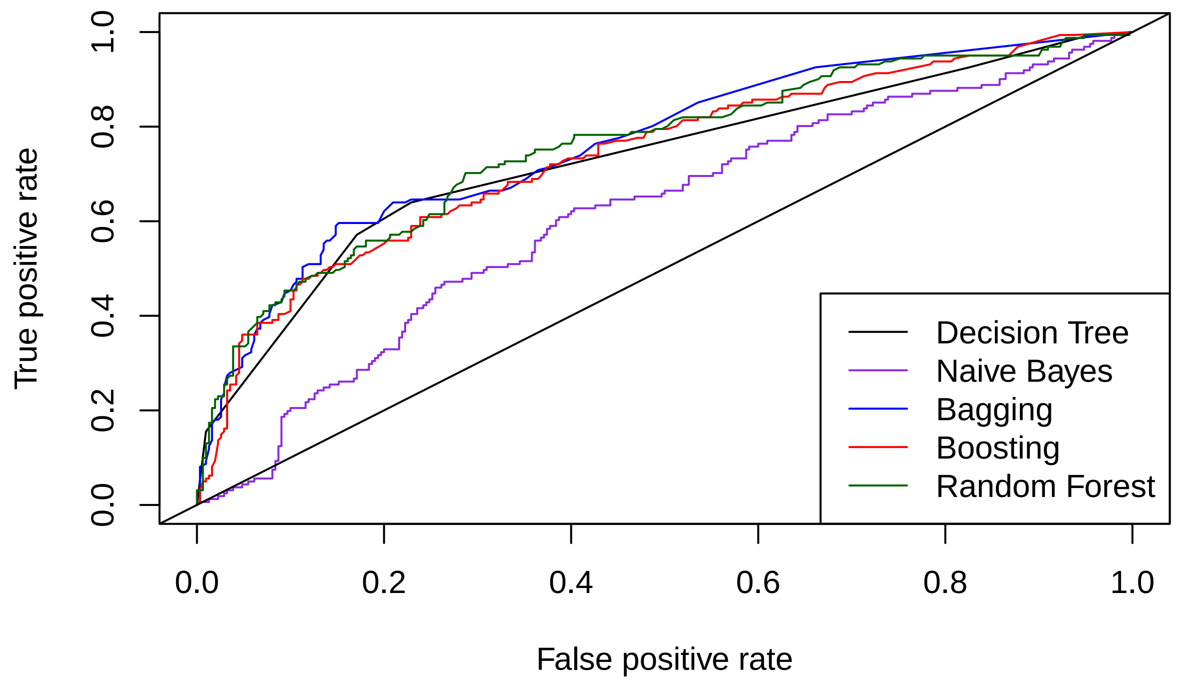
```r
# Computing a simple ROC Curve (x-axis: fpr, y-axis: tpr)
# Labels are actual values, predictors are probabilities of class
PD.pred.tree = predict(PD.tree, PD_clean.test, type = 'vector')
PDDpred = prediction(PD.pred.tree[,2], PD_clean.test$Class)
PDDperf = performance(PDDpred, "tpr", "fpr")

# Naive Bayes Performance
PDpred.bayes = predict(PD.bayes, PD_clean.test, type = 'raw')
PDBpred = prediction(PDpred.bayes[,2], PD_clean.test$Class)
PDBperf = performance(PDBpred, "tpr", "fpr")
# Bagging Performance
PDBagpred <- prediction(PDpred.bag$prob[,2], PD_clean.test$Class)
PDBagperf <- performance(PDBagpred,"tpr","fpr")
# Boosting Performance
PDBoostpred <- prediction(PDpred.boost$prob[,2], PD_clean.test$Class)
PDBoostperf <- performance(PDBoostpred,"tpr","fpr")
# Random Forest
PDpred.rf <- predict(PD.rf, PD_clean.test, type="prob")
PDFpred <- prediction(PDpred.rf[,2], PD_clean.test$Class)
PDFperf <- performance(PDFpred,"tpr","fpr")

# plotting ROCs
plot(PDDperf)
plot(PDBperf, add= TRUE, col = 'blueviolet')
plot(PDBagperf, add = TRUE, col = 'blue')
plot(PDBoostperf, add = TRUE, col = 'red')
plot(PDFperf, add=TRUE, col = 'darkgreen')

# line
abline(0,1)

# legend
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest")
```



```
# AUC Scores
# Decision Tree
dauc = performance(PDDpred, "auc")
# Naive Bayes
Bauc = performance(PDBpred, "auc")
# Bagging
Bagauc = performance(PDBagpred, "auc")
# Boosting
Boostauc = performance(PDBoostpred, "auc")
# Random Forest
Fauc = performance(PDFpred, "auc")

AUC_table <- data.frame(
  Classifer = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
  # All AUC Scores
  AUC = c(as.numeric(dauc@y.values),
          as.numeric(Bauc@y.values),
          as.numeric(Bagauc@y.values),
          as.numeric(Boostauc@y.values),
          as.numeric(Fauc@y.values)),
  # Each table is a confusion matrix
  Accuracy = c(sum(diag(t1))/sum(t1) * 100,
               sum(diag(t2))/sum(t2) * 100,
               sum(diag(t3))/sum(t3) * 100,
               sum(diag(t4))/sum(t4) * 100,
               sum(diag(t5))/sum(t5) * 100
               )
)
AUC_table
```

```
##       Classifer       AUC Accuracy
## 1 Decision Tree 0.7304047 74.09766
## 2   Naive Bayes 0.6138149 50.74310
## 3       Bagging 0.7634743 75.15924
## 4      Boosting 0.7396213 74.52229
## 5 Random Forest 0.7550992 74.94692
```

The best classification accuracy for Phishing and legitimate websites is 75.16% by the Bagging Classifier. In addition, it also contains the highest AUC score of 0.76.

By the AUC criterion, Bagging is the 'best' Classifier. Notably, Random Forest and Boosting are strong performers with high Accuracy and AUC score.

Surprisingly, the Decision Tree performed reasonably well in comparison to the ensemble methods. Naive Bayes, unfortunately perform the weakest; the lowest accuracy and lowest AUC score.

Investigative Section

Attribute Importance

```
print(summary(PD.tree))
```

i. Decision Tree

```
##
## Classification tree:
## tree(formula = Class ~ ., data = PD_clean.train)
## Variables actually used in tree construction:
## [1] "A01" "A23" "A18"
## Number of terminal nodes:  6
## Residual mean deviance:  1.065 = 1162 / 1091
## Misclassification error rate: 0.258 = 283 / 1097
```

The three important attributes in this decision tree is A01, A23 and A18. The reason why all other website attributes can be omitted is that the algorithm greedily finds the best splits in the Phishing data, so that the entropy is maximised in the split as measured using the outcome. All other website attributes besides A01, A23 and A18 can be omitted.

```r
PD.bag$importance
```

i. Bagging

```
##         A01         A02         A04         A06         A08         A09 
## 34.17798882  0.38981066  0.35276060  0.34631369  4.30470949  0.02775899 
##         A10         A12         A14         A15         A16         A17 
##  0.06023149  3.20203399  0.36610996  0.40373505  0.35552160  0.23895769 
##         A18         A19         A20         A22         A23         A24 
## 13.14076495  0.64633097  0.39695031  8.30183399 30.14516927  3.14301847
```

The most important attributes contributing from Bagging, in descending order in rank, A01, A23, A22, A18 and A08. Variable importance in this context of bagging shows a lot more higher activity than other website attributes. Given the score for attributes not mentioned to be less than 1, indicates their little to no contribution into the performance.

```r
PD.Boost$importance
```

```
##        A01        A02        A04        A06        A08        A09        A10 
## 20.2351960  1.4369681  0.8809051  0.0000000 10.5066137  0.0000000  0.2219700 
##        A12        A14        A15        A16        A17        A18        A19 
##  4.9378519  0.9499041  0.5334666  1.1277435  2.0986899 13.1300220  0.0000000 
##        A20        A22        A23        A24 
##  1.6469118 17.9695032 20.7661588  3.5580955
```

Likewise, the ranking of importance is the same as previous models: A01, A23, A22, A18 then A08. Their higher scores indicates their respective involvement on the classification model.

```r
print(PD.rf$importance)
```

```
##     MeanDecreaseGini
## A01        76.288159
## A02         6.445870
## A04         7.190461
## A06         6.292209
## A08        31.574452
## A09         2.420766
## A10         2.544400
## A12        27.983459
## A14         8.854564
## A15         6.569987
## A16         5.314713
## A17        10.590660
## A18        70.506923
## A19         6.764632
## A20         9.206317
## A22        71.871474
## A23        69.105883
## A24        27.734380
```

Likewise, the ranking of importance by the Gini score is consistent with the metrics from previous models. The five website attributes A01, A23, A22, A18 then A08 are the most important variables in predicting whether a website will be phishing or legitimate. All the other attributes, from all 4 good classifiers indicates a consistent result of all other attributes could be omitted.

Simple Model Creation Let us now create a classifier that is simple enough for a person to be able to classify whether a site is phishing or legitimate by hand. Let us construct a decision tree with two website attributes; A01 and A23

- If A01 $<$ 21.5, its legitimate,
- else, if A23 $<$ 1.5, its legitimate
- else its phishing.

From the previous section, A01 and A23 were two of the three most important variables in the decision tree. Let us code this classifier

```r
Classifier <- function(A01, A23){
  if (A01 < 21.5){
    return(0) # Legitimate
  } else{
    if (A23 < 1.5){
      return(0) # Legitimate
    } else {
      return(1) # Phishing
    }
  }
}
```

Perform the Classifier on the test data set

```r
# Create copy of test data.
Test_data <- PD_clean.test
# Classifier output goes to the new column prediction
Test_data$prediction <- mapply(Classifier, Test_data$A01, Test_data$A23)
```

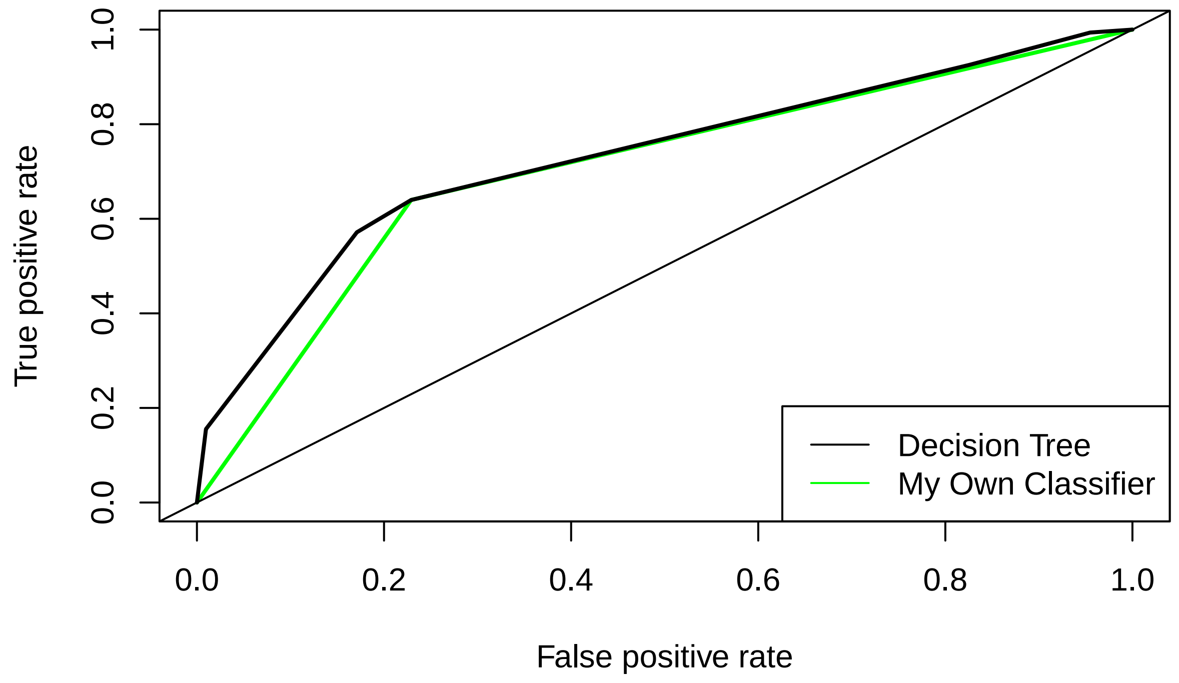
Let us create a confusion matrix

```r
# Confusion_matrix
t6 <- table(Predicted = Test_data$prediction, Actual = Test_data$Class)
t6
```

```
##          Actual
## Predicted   0   1
##         0 239  58
##         1  71 103
```

The classification accuracy of my classifier in the test set is **72.61**%

```
preds = prediction(Test_data$prediction, Test_data$Class)
# RoC Curve
roc_curve = performance(preds, "tpr", "fpr")
plot(roc_curve, col = 'green', lwd = 2)
plot(PDDperf, add = TRUE ,col = 'black', lwd = 2)
abline(0,1)
# Legend
legend("bottomright",
       legend = c("Decision Tree", "My Own Classifier"),
       col = c('black', 'green'), lty = 1)
```



```
Mauc = performance(preds, "auc")
# Add the results to our current AUC table
AUC_table[nrow(AUC_table) + 1,] = c('MyClassifier',
                                    Mauc@y.values,
                                    sum(diag(t6))/sum(t6) * 100)
AUC_table
```

```
##       Classifer       AUC Accuracy
## 1 Decision Tree 0.7304047 74.09766
## 2   Naive Bayes 0.6138149 50.74310
## 3       Bagging 0.7634743 75.15924
## 4      Boosting 0.7396213 74.52229
## 5 Random Forest 0.7550992 74.94692
## 6  MyClassifier 0.7053596 72.61146
```

Although my classifier is simple, it has performed reasonably well compared to other classifiers. It has performed slightly worst than the decision tree classifier, however it is a stronger classifier than the Naive Bayes classifier. The AUC score is 0.705 and the classification accuracy is 72.61%

Optimal Tree-based Classifier

Let us attempt to improve upon the Bagging Classifier by cross validating.

```
set.seed(1) # For repeatiblity
# Create Bagging Model
PD2.bag = bagging(Class ~ A01 + A08 + A18 + A22 + A23,
                  data = PD_clean.train,
                  mfinals = 3,
                  trControl = trainControl(method = "cv", number = 10),
                  nbagg = 100)
```

```
PDDpred.bag = predict.bagging(PD2.bag, PD_clean.test)
# Predicted Class from Bagging Model
pred_class = as.factor(PDDpred.bag$class)
# Confusion Matrix
t7 = PDDpred.bag$confusion
t7
```

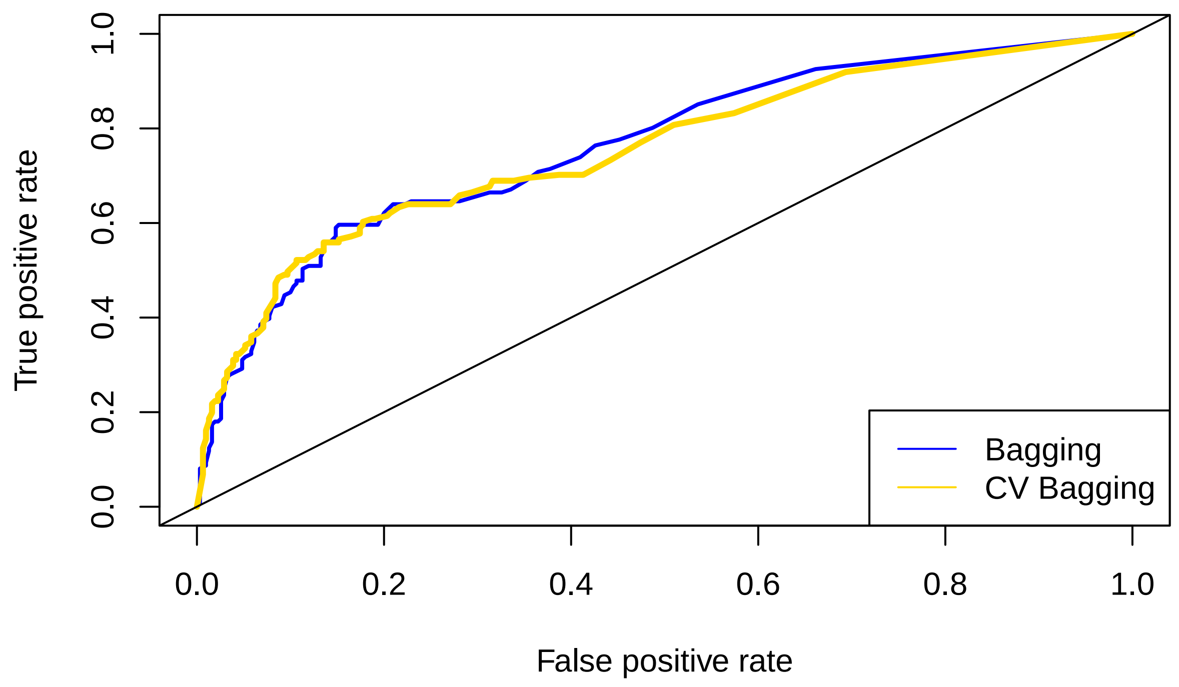
```
##                  Observed Class
## Predicted Class    0   1
##                0 281  82
##                1  29  79
```

```
# Default Bagging Performance
PDBagpred <- prediction(PDpred.bag$prob[,2], PD_clean.test$Class)
PDBagperf <- performance(PDBagpred,"tpr","fpr")
# Cross Validated Bagging Performance
PDCVBagpred <- prediction(PDDpred.bag$prob[,2], PD_clean.test$Class)
PDCVBagperf <- performance(PDCVBagpred,"tpr","fpr")
```

```
# plotting ROCs
plot(PDBagperf, col = 'blue', lwd = 2)
plot(PDCVBagperf, add = TRUE, col = 'gold', lwd = 3)

# line
abline(0,1)

# legend
legend("bottomright", legend = c("Bagging", "CV Bagging"), col = c('blue', 'gold'), lty = 1)
```



```r
CVauc = performance(PDCVBagpred, "auc")
# Add the results to our current AUC table
AUC_table[nrow(AUC_table) + 1,] = c('CV Bagging',
                                    CVauc@y.values,
                                    sum(diag(t7))/sum(t7) * 100)
AUC_table
```

```
##       Classifer       AUC Accuracy
## 1 Decision Tree 0.7304047 74.09766
## 2   Naive Bayes 0.6138149 50.74310
## 3       Bagging 0.7634743 75.15924
## 4      Boosting 0.7396213 74.52229
## 5 Random Forest 0.7550992 74.94692
## 6  MyClassifier 0.7053596 72.61146
## 7    CV Bagging 0.7552795 76.43312
```

Interestingly the Cross validation Bagging Classifier has improve on the classification accuracy (76.43%) compared to Bagging (75.15%). However AUC score has decreased slightly of 0.755 compared to the Bagging 0.763.

The attributes I chose in this model are A01, A23, A22, A18 and A08. These feature selections I believe improves upon the model's performance. Not only it reduces the computational complexity by reducing all the other website features, but it improves on the interpretability by focusing on the most important features and removing noise or irrelevant information. As indicated from the previous sections, these attributes contribute the most in terms of classifying phishing and legitimate website.

However, with the current results, it is unclear that the model has overall improved due to the reduction of the AUC score.

Artificial Neural Network

In this Neural Network, we will utilise 5 website attributes, based on their importance. That is, A01, A08, A18, A22 and A23. We will utilise Abishek's improved solution method to classify phishing websites from Applied Session 09.

```
suppressWarnings({require(neuralnet, quietly = TRUE)})
```

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

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

```
# Create a new copy of PD dataset
PD2 <- PD
# Remove NAs
PD2 = PD2[complete.cases(PD2),]
PD2$Class = as.numeric(PD2$Class)
```

```
set.seed(31484808)
train.row = sample(1:nrow(PD2), 0.7*nrow(PD2))
PD2.train = PD2[train.row,]
PD2.test = PD2[-train.row,]
# Binomial Classification: Predict the probability of belong to class Phishing (1) and if the probabili
# Create ANN
PD2.nn = neuralnet(Class ~ A01 + A08 + A18 + A22 + A23, PD2.train, hidden = 3, linear.output = FALSE)
```

```
# From attribute importance, we use A01, A08, A18, A22 and A23
prob = compute(PD2.nn, PD2.test[c(1, 8, 18, 22, 23)])
prob.results = prob$net.result
ANNpred = ifelse(prob.results > 0.5, 1, 0)
```

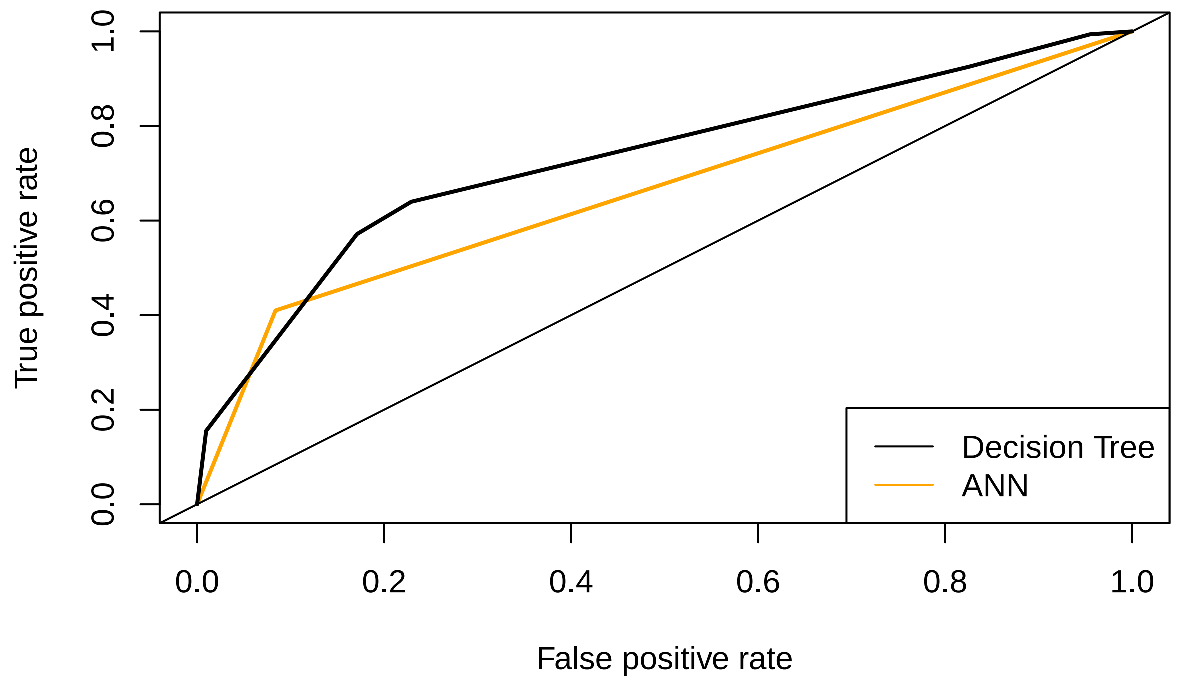
```
t8 = table(observed = PD2.test$Class, predicted = ANNpred)
t8
```

Training and test sets

```
##          predicted
## observed   0   1
##        0 284  26
##        1  95  66
```

```r
# Detach Neural net package to avoid $ operator is invalid for atomic vectors
detach(package:neuralnet, unload = T)
```

```r
nn.preds = prediction(ANNpred, PD2.test$Class)
# ROC Curve for ANN
ANNperf = performance(nn.preds, "tpr", "fpr")
plot(ANNperf, col = 'orange', lwd = 2)
plot(PDDperf, add = TRUE ,col = 'black', lwd = 2)
abline(0,1)
# Legend
legend("bottomright", legend = c("Decision Tree", "ANN"), col = c('black', 'orange'), lty = 1)
```



```r
ANNauc = performance(nn.preds, "auc")
# Add the results to our current AUC table
AUC_table[nrow(AUC_table) + 1,] = c('ANN',
                                    ANNauc@y.values,
                                    sum(diag(t8))/sum(t8) * 100)
AUC_table
```

```
##        Classifer       AUC Accuracy
## 1 Decision Tree 0.7304047 74.09766
## 2   Naive Bayes 0.6138149 50.74310
## 3       Bagging 0.7634743 75.15924
## 4      Boosting 0.7396213 74.52229
## 5 Random Forest 0.7550992 74.94692
## 6  MyClassifier 0.7053596 72.61146
## 7    CV Bagging 0.7552795 76.43312
## 8           ANN 0.6630335 74.30998
```

Although the classification accuracy is reasonably high, the AUC score is on the lower end. The ANN classifier performed better than Naive Bayes Classifier, however it cannot be said for Decision Tree; deciding whether the metric accuracy or AUC score is more important is contextual.

In this scenario, where we are detecting Phishing websites, the positive case (Class = 1) is much rarer than the legitimate case. A higher accuracy might simply be predicting the majority of cases well, while a higher AUC would indicate model's strength in distinguishing between the two cases. In my opinion, the decision tree is the stronger classifier than the ANN.

XGBoost Classifier

XGBoost is an another machine learning algorithm that belongs to the ensemble learning category that we have explored in the previous section; It is specifically the gradient boosting framework. It uses decision trees as base learners and employs regularization techniques to improve generalisation.

It is known to be very efficient, compared to ada boosting.

Xgboost builds the predictive model by combining the predictions of multiple models, in our case, decision trees, in an iterative manner.

It works by sequentially adding weak learners to the ensemble, with each learner focusing on correcting the errors made by the existing ones.

More information can be found in https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article

The R package regarding XGBoost info: https://xgboost.readthedocs.io/en/stable/R-package/xgboostPresentation.html

My approach is based on https://www.projectpro.io/recipes/apply-xgboost-for-classification-r

```r
library(xgboost)
```

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

```r


# Define Independent and Dependent Variables
X_train = data.matrix(PD_clean.train[, -ncol(PD_clean)]) # Exlude Class & as a matrix
Y_train = PD_clean.train[, ncol(PD_clean)] # Only Class

X_test = data.matrix(PD_clean.test[, -ncol(PD_clean)]) # Exlude Class & as a matrix
Y_test = PD_clean.test[, ncol(PD_clean)] # Only Class
```

Train and Test Data Convert the train and test data into XGBoost matrix type

```r
xgboost_train = xgb.DMatrix(data = X_train, label = Y_train)
xgboost_test = xgb.DMatrix(data = X_test, label = Y_test)
```

Create XGBoost Model

```r
PDXGBoost <- xgboost(data = xgboost_train,    # the data
                 max.depth = 7,
                 eta = 0.1,      # learning rate
                 nrounds= 50,)  # max number of boosting iterations
```

```
## [1]  train-rmse:0.873836
## [2]  train-rmse:0.805029
## [3]  train-rmse:0.744092
## [4]  train-rmse:0.689576
## [5]  train-rmse:0.641554
## [6]  train-rmse:0.598693
## [7]  train-rmse:0.560310
## [8]  train-rmse:0.524472
## [9]  train-rmse:0.495118
## [10] train-rmse:0.468028
## [11] train-rmse:0.443533
## [12] train-rmse:0.422028
## [13] train-rmse:0.403603
## [14] train-rmse:0.386970
## [15] train-rmse:0.371760
## [16] train-rmse:0.357925
## [17] train-rmse:0.346055
## [18] train-rmse:0.334988
## [19] train-rmse:0.325898
## [20] train-rmse:0.317063
## [21] train-rmse:0.310307
## [22] train-rmse:0.303176
## [23] train-rmse:0.297997
## [24] train-rmse:0.291537
## [25] train-rmse:0.287336
## [26] train-rmse:0.284915
## [27] train-rmse:0.280075
## [28] train-rmse:0.274834
## [29] train-rmse:0.272255
## [30] train-rmse:0.270762
## [31] train-rmse:0.267998
## [32] train-rmse:0.265806
## [33] train-rmse:0.263381
## [34] train-rmse:0.260768
## [35] train-rmse:0.258317
## [36] train-rmse:0.256222
## [37] train-rmse:0.255304
## [38] train-rmse:0.252962
## [39] train-rmse:0.251319
## [40] train-rmse:0.247949
## [41] train-rmse:0.246930
## [42] train-rmse:0.246178
## [43] train-rmse:0.241661
## [44] train-rmse:0.240907
## [45] train-rmse:0.237845
## [46] train-rmse:0.236341
## [47] train-rmse:0.235499
## [48] train-rmse:0.233340
## [49] train-rmse:0.232900
## [50] train-rmse:0.231139
```

Make predictions on the test data set

```
XGBpred = predict(PDXGBoost, xgboost_test)
# Convert predicted values into factor values
XGBpred = as.factor((levels(Y_test))[round(XGBpred)])
```

```
conf_mat = confusionMatrix(Y_test, XGBpred)
print(conf_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 269  41
##          1  85  76
##
##                Accuracy : 0.7325
##                  95% CI : (0.6901, 0.772)
##     No Information Rate : 0.7516
##     P-Value [Acc > NIR] : 0.8444622
##
##                   Kappa : 0.3637
##
##  Mcnemar's Test P-Value : 0.0001278
##
##             Sensitivity : 0.7599
##             Specificity : 0.6496
##          Pos Pred Value : 0.8677
##          Neg Pred Value : 0.4720
##              Prevalence : 0.7516
##          Detection Rate : 0.5711
##    Detection Prevalence : 0.6582
##       Balanced Accuracy : 0.7047
##
##        'Positive' Class : 0
##
```

Comparatively, it has performed better than ANN, Bayes Naive and my own simple classifier in terms of accuracy. However, it is not as a better performer than decision tree, Bagging and 'Ada' Boosting.

Appendix

A modified version of the PhiUSIIL Phishing data, hosted by the UCI Machine Learning Archive https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset.

A research paper based on this data is available here https://doi.org/10.1016/j.cose.2023.103545.

XGBoost approach:

https://www.projectpro.io/recipes/apply-xgboost-for-classification-r

XGBoost info

https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.