# Website Legitimacy Prediction using Classification Models

Cherylyn Yong

```
# Load the library and read the file.
rm(list = ls())

library(ggplot2)
library(dplyr)
library(tidyr)

Phish = read.csv("/Users/cherylynyong/Library/CloudStorage/OneDrive-MonashUniversity/DONE 2024 YEAR
```

```
# create data set
set.seed(32714491)
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
PD$Class=factor(PD$Class)
```

**Question 1**

***Data Exploration***

```
glimpse(PD)
```

```
## Rows: 2,000
## Columns: 26
## $ A01   <int> 20, 14, 26, 24, 23, 26, 15, 15, 30, 20, 24, 25, 26, 15, 20, 23, ~
## $ A02   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A03   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A04   <int> 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 2, 3, 3, 3, 3, 3, 2, 3, 3, 2~
## $ A05   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A06   <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0~
## $ A07   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A08   <dbl> 1.0000000, 1.0000000, 0.3636364, 0.5000000, 0.8461538, 1.0000000~
## $ A09   <int> 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A10   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A11   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, ~
## $ A12   <int> 232, 232, 633, 133, 692, 232, 686, 232, 504, 232, 180, 232, 142,~
## $ A13   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A14   <int> 0, 0, NA, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ A15   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1~
## $ A16   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ A17   <int> 0, 1, 3, 1, 2, 1, 0, 1, 1, 1, 2, 1, 2, 0, 1, 1, 1, 0, 3, 1, 1, 0~
## $ A18   <int> 5, 16, 8, 52, 126, 14, 6, 26, 68, 26, 63, 45, 51, 19, 28, 90, 22~
## $ A19   <int> 0, 0, 0, 0, 0, NA, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ A20   <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1~
## $ A21   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
## $ A22   <dbl> 0.06846515, 0.06529802, 0.05758219, 0.03917048, 0.05748302, 0.05~
## $ A23   <int> 0, 0, 100, 100, 134, 100, 0, 0, 11, 147, 108, 109, 105, 100, 108~
## $ A24   <dbl> 0.5229071, 0.5229071, 0.0002752, 0.0015020, 0.0018398, 0.5229071~
## $ A25   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Class <fct> 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0~
```

Using glimpse() function above, we can identify that A08, A22 and A24 are real-valued attributes.

```
# find number of different sites
no_phising_sites = PD %>%
  filter(Class == 1) %>%
  nrow()

no_legitimate_sites = PD %>%
  filter(Class == 0) %>%
  nrow()

# find the proportion
proportion = no_phising_sites / no_legitimate_sites
proportion
```

```
## [1] 0.4925373
```

We can see that proportion of phishing sites to legitimate sites is **0.4925373**.

```
# find statistics of real-valued attributes
desc_A08 = PD %>%
  summarise(Mean = mean(A08, na.rm = TRUE),
            Standard_deviation = sd(A08, na.rm = TRUE),
            IQR = IQR(A08, na.rm = TRUE))
desc_A08
```

```
##        Mean Standard_deviation IQR
## 1 0.8550513          0.2129456 0.3
```

```
desc_A22 = PD %>%
  summarise(Mean = mean(A22, na.rm = TRUE),
            Standard_deviation = sd(A22, na.rm = TRUE),
            IQR = IQR(A22, na.rm = TRUE))
desc_A22
```

```
##         Mean Standard_deviation        IQR
## 1 0.05593758         0.01071762 0.01168131
```

```
desc_A24 = PD %>%
  summarise(Mean = mean(A24, na.rm = TRUE),
            Standard_deviation = sd(A24, na.rm = TRUE),
            IQR = IQR(A24, na.rm = TRUE))
desc_A24
```

```
##        Mean Standard_deviation       IQR
## 1 0.2708555          0.2523259 0.5169299
```

```
summary(PD)
```

```
##       A01              A02               A03                A04
##  Min.   : 1.00   Min.   : 0.0000   Min.   :0.000000   Min.   :2.000
##  1st Qu.:15.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.000
##  Median :20.00   Median : 0.0000   Median :0.000000   Median :3.000
##  Mean   :19.55   Mean   : 0.1518   Mean   :0.002016   Mean   :2.768
##  3rd Qu.:25.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.000
##  Max.   :30.00   Max.   :24.0000   Max.   :1.000000   Max.   :7.000
##                  NA's   :24        NA's   :16         NA's   :14
##       A05                A06              A07                A08
##  Min.   : 0.00000   Min.   :0.000   Min.   :0.000000   Min.   :0.1818
##  1st Qu.: 0.00000   1st Qu.:0.000   1st Qu.:0.000000   1st Qu.:0.7000
##  Median : 0.00000   Median :0.000   Median :0.000000   Median :1.0000
##  Mean   : 0.01212   Mean   :0.129   Mean   :0.001003   Mean   :0.8551
##  3rd Qu.: 0.00000   3rd Qu.:0.000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :16.00000   Max.   :1.000   Max.   :1.000000   Max.   :1.0000
##  NA's   :20         NA's   :16      NA's   :6          NA's   :19
##       A09                A10               A11                A12
##  Min.   :0.00000   Min.   :0.00000   Min.   : 0.00000   Min.   : 48.0
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:232.0
##  Median :0.00000   Median :0.00000   Median : 0.00000   Median :232.0
##  Mean   :0.02575   Mean   :0.04127   Mean   : 0.05343   Mean   :314.8
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:398.0
##  Max.   :1.00000   Max.   :1.00000   Max.   :10.00000   Max.   :692.0
##  NA's   :19        NA's   :13        NA's   :16         NA's   :15
##       A13                A14               A15                A16
##  Min.   :0.000000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :0.000000   Median :0.0000   Median :0.0000   Median :0.00000
##  Mean   :0.004541   Mean   :0.1063   Mean   :0.1419   Mean   :0.03937
##  3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :6.000000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##  NA's   :18         NA's   :15       NA's   :12       NA's   :19
##       A17              A18                A19               A20
##  Min.   :0.00   Min.   :   5.00   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:1.00   1st Qu.:  14.00   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.00   Median :  34.00   Median :0.0000   Median :0.0000
##  Mean   :1.15   Mean   :  58.11   Mean   :0.1071   Mean   :0.2272
##  3rd Qu.:1.00   3rd Qu.:  89.00   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :5.00   Max.   :2237.00   Max.   :1.0000   Max.   :1.0000
##  NA's   :22     NA's   :16        NA's   :20       NA's   :24
##       A21                A22                A23                A24
##  Min.   :0.00000   Min.   :0.001209   Min.   :  0.00   Min.   :0.000000
##  1st Qu.:0.00000   1st Qu.:0.051310   1st Qu.: 20.00   1st Qu.:0.005977
##  Median :0.00000   Median :0.058350   Median :100.00   Median :0.079963
##  Mean   :0.02934   Mean   :0.055938   Mean   : 80.06   Mean   :0.270856
##  3rd Qu.:0.00000   3rd Qu.:0.062991   3rd Qu.:108.00   3rd Qu.:0.522907
##  Max.   :3.00000   Max.   :0.084876   Max.   :794.00   Max.   :0.522907
##  NA's   :23        NA's   :26         NA's   :23       NA's   :24
##       A25          Class
##  Min.   :0.0e+00   0:1340
##  1st Qu.:0.0e+00   1: 660
##  Median :0.0e+00
##  Mean   :4.9e-05
##  3rd Qu.:0.0e+00
##  Max.   :6.4e-02
```

```
##  NA's   :23
```

Using the summary() function, we can see that there are some missing values in the attributes. A01 is the only attribute that does not have any missing values.

### Question 2

***Data Pre-processing***

```
cleaned_PD = na.omit(PD)
```

The pre-processing needed to do on the dataset is removing all the missing values on the dataset.

### Question 3

***Create training and testing data***

```
set.seed(32714491)
PD.train.row = sample(1:nrow(cleaned_PD), 0.7*nrow(cleaned_PD))
PD.train = cleaned_PD[PD.train.row,]
PD.test = cleaned_PD[-PD.train.row,]
```

### Question 4

***Implement classification models***

```
library(tree)
library(e1071)
library(ROCR)
library(pROC)
library(rpart)
library(adabag)
library(randomForest)

# Decision Tree
PD_tree = tree(Class ~ ., data = PD.train)

# Naïve Bayes
PD_bayes = naiveBayes(Class ~ ., data = PD.train)

# Bagging
PD_bagging = bagging(Class ~ ., data = PD.train, mfinal = 10)

# Boosting
PD_boosting = boosting(Class ~ ., data = PD.train, mfinal = 10)

# Random Forest
PD_rf = randomForest(Class ~ ., data = PD.train)
```

### Question 5

***Confusion Matrix and Accuracy***

```r
# Decision Tree
tree_predict = predict(PD_tree, PD.test, type = "class")
tree_table = table(actual = PD.test$Class, predicted = tree_predict)
tree_acc = sum(diag(tree_table)) / sum(tree_table)
```

The accuracy of model using Decision Tree is **0.6325678**.

```r
# Naïve Bayes
bayes_predict = predict(PD_bayes, PD.test)
bayes_table = table(actual = PD.test$Class, predicted = bayes_predict)
bayes_acc = sum(diag(bayes_table)) / sum(bayes_table)
```

The accuracy of model using Naïve Bayes is **0.3841336**.

```r
# Bagging
bagging_predict = predict(PD_bagging, PD.test)
bagging_table = table(actual = PD.test$Class, predicted = bagging_predict$class)
bagging_acc = sum(diag(bagging_table)) / sum(bagging_table)
```

The accuracy of model using bagging is **0.6430063**.

```r
# Boosting
boosting_predict = predict(PD_boosting, PD.test)
boosting_table = table(actual = PD.test$Class, predicted = boosting_predict$class)
boosting_acc = sum(diag(boosting_table)) / sum(boosting_table)
```

The accuracy of model using boosting is **0.6555324**.

```r
# Random Forest
rf_predict = predict(PD_rf, PD.test)
rf_table = table(actual = PD.test$Class, predicted = rf_predict)
rf_acc = sum(diag(rf_table)) / sum(rf_table)
```

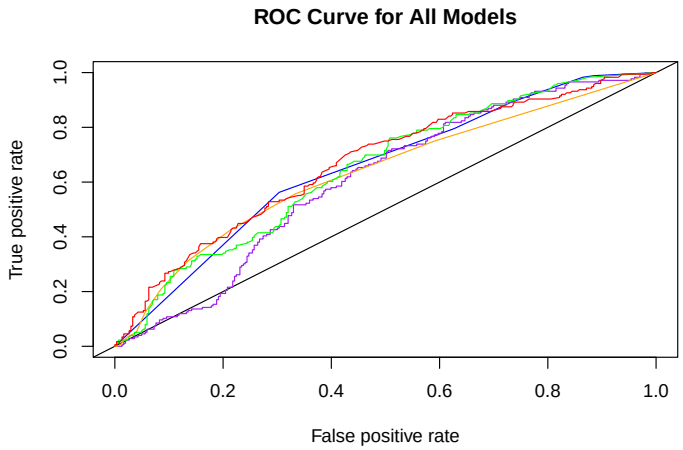
The accuracy of model using Random Forest is **0.6638831**.

### Question 6

***Confidence, ROC & AUC***

```r
# confidence & ROC for Decision Tree
tree_predict2 = predict(PD_tree, PD.test, type = "vector")
PDpred_tree = prediction(tree_predict2[,2], PD.test$Class)
PDperf_tree = performance(PDpred_tree,"tpr","fpr")
plot(PDperf_tree, col = "blue", main = "ROC Curve for All Models")
abline(0,1, col="black")

# confidence & ROC for Naïve Bayes
bayes_predict2 = predict(PD_bayes, PD.test, type = "raw")
PDpred_bayes = prediction(bayes_predict2[,2], PD.test$Class)
PDperf_bayes = performance(PDpred_bayes,"tpr","fpr")
plot(PDperf_bayes, add=TRUE, col = "purple")

# confidence & ROC for Bagging
bagging_predict2 = predict(PD_bagging, PD.test, type = "raw")
PDpred_bagging = prediction(bagging_predict2$prob[,2], PD.test$Class)
```

```r
PDperf_bagging = performance(PDpred_bagging,"tpr","fpr")
plot(PDperf_bagging, add=TRUE, col = "orange")

# confidence & ROC for Boosting
boosting_predict2 = predict(PD_boosting, PD.test, type = "raw")
PDpred_boosting = prediction(boosting_predict2$prob[,2], PD.test$Class)
PDperf_boosting = performance(PDpred_boosting,"tpr","fpr")
plot(PDperf_boosting, add=TRUE, col = "green")

# confidence & ROC for Random Forest
rf_predict2 = predict(PD_rf, PD.test, type = "prob")
PDpred_rf = prediction(rf_predict2[,2], PD.test$Class)
PDperf_rf = performance(PDpred_rf,"tpr","fpr")
plot(PDperf_rf, add=TRUE, col = "red")
```




```r
# AUC for each classifier
auc_tree = performance(PDpred_tree, "auc")
auc_tree_y = as.numeric(auc_tree@y.values)

auc_bayes = performance(PDpred_bayes, "auc")
auc_bayes_y = as.numeric(auc_bayes@y.values)

auc_bagging = performance(PDpred_bagging, "auc")
auc_bagging_y = as.numeric(auc_bagging@y.values)

auc_boosting = performance(PDpred_boosting, "auc")
auc_boosting_y = as.numeric(auc_boosting@y.values)

auc_rf = performance(PDpred_rf, "auc")
auc_rf_y = as.numeric(auc_rf@y.values)
```

### Question 7

***Comparison using Table***

| Classifier | Accuracy | AUC |
|---|---|---|
| Decision Tree | 0.6325678 | 0.6522465 |
| Naïve Bayes | 0.3841336 | 0.6078795 |
| Bagging | 0.6430063 | 0.6342916 |

| Classifier | Accuracy | AUC |
|---|---|---|
| Boosting | 0.6555324 | 0.6493306 |
| Random Forest | 0.6638831 | 0.6680356 |

From the table, we can see that the best classifier is Random Forest. It has the highest accuracy and AUC.

### Question 8

***Investigate variables***

```
# check importance of variables
summary(PD_tree)
PD_bayes
sort(PD_bagging$importance, decreasing = TRUE)
sort(PD_boosting$importance, decreasing = TRUE)
sort(PD_rf$importance[,1], decreasing = TRUE)
```

We can observe that the most **important** variables are **A01, A18, A22, and A23**. These attributes are crucial for tree construction in the decision tree model. Additionally, they exhibit higher conditional probabilities in the Bayes model. Furthermore, they demonstrate higher importance values in bagging, boosting, and random forest models compared to other variables. The variables that could be **omitted** from the data are **A25, A13, A07, and A05**. These attributes hold no importance and have the lowest conditional probabilities for performance.

### Question 9

***Create simple classifier***

```
# Simple Decision Tree
simple_tree = tree(Class ~ A01 + A22, data = PD.train)
simple_pred = predict(simple_tree, PD.test, type = "class")
simple_table = table(actual = PD.test$Class, predicted = simple_pred)
simple_acc = sum(diag(simple_table)) / sum(simple_table)

# confidence & ROC for Decision Tree
simple_pred2 = predict(simple_tree, PD.test, type = "vector")
PDpred_simple = prediction(simple_pred2[,2], PD.test$Class)
auc_simple = performance(PDpred_simple, "auc")
auc_simple_y = as.numeric(auc_simple@y.values)
```

First, I opt for the decision tree as the classifier due to its high interpretability. This characteristic makes it easier for individuals to classify websites, as they can navigate the tree's branches and observe the leaf nodes for classification. I specifically choose attributes **A01 and A22** because they are the strongest predictors, as evidenced by their high importance values. While the previous decision model utilized all attributes for tree construction, I experiment with fewer attributes to assess potential performance improvements or declines.

In this simplified model, the root node starts with the most critical attribute, A01, followed by branching to A22. Ultimately, the leaf nodes aid in classifying whether a site is phishing or legitimate.

This simpler model achieves an accuracy of **0.6325678** and an AUC of **0.6339446**. In comparison, the previous decision tree model achieved the same accuracy of **0.6325678** but had a higher AUC of **0.6522465** This indicates that while the simpler model performs equally well in terms of accuracy, it is slightly less effective at distinguishing between phishing and legitimate sites as reflected by the lower

AUC value. The higher AUC in the previous model suggests that it has a better overall ability to classify the sites correctly, despite its increased complexity.

| Classifier | Accuracy | AUC |
|---|---|---|
| Decision Tree | 0.6325678 | 0.6522465 |
| Naïve Bays | 0.3841336 | 0.6078795 |
| Bagging | 0.6430063 | 0.6342916 |
| Boosting | 0.6555324 | 0.6493306 |
| Random Forest | 0.6638831 | 0.6680356 |
| Simple Model | 0.6325678 | 0.6339446 |

### Question 10

***Create best classifier***

```
PD_baggingcv = bagging.cv(Class ~ ., data = PD.train, mfinal = 10)
baggingcv_acc = sum(diag(PD_baggingcv$confusion)) / sum(PD_baggingcv$confusion)
```

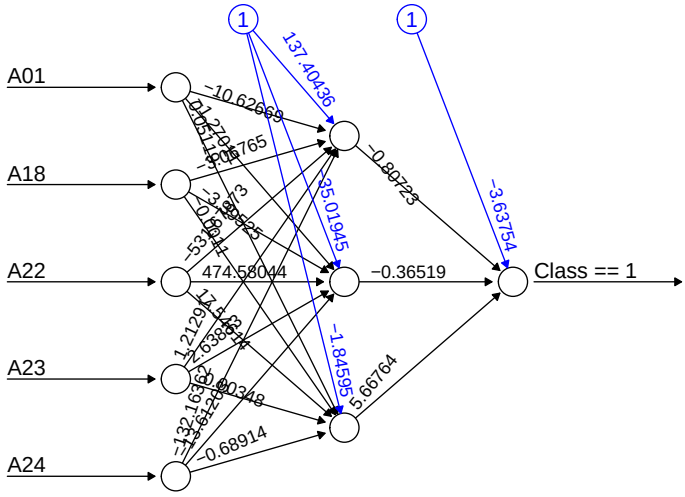
**Bagging with cross-validation (CV)** involves creating 10 models by repeatedly sampling subsets of the training data and then aggregating their predictions to improve accuracy and robustness. Bagging was selected because it reduces overfitting by averaging the predictions of multiple models, such as decision trees, which individually might overfit the training data. By leveraging the strengths of each individual model, bagging creates a combined model that performs better than any single model alone. In terms of accuracy, this classifier has the best accuracy among all classifiers, with an accuracy of **0.691137**. There is no AUC as the object did not return probabilities that allow us to create a prediction object. In conclusion, it outperforms other classifiers by utilizing multiple models to reduce overfitting and variance, leveraging the strengths of each model, and providing a more robust and accurate final prediction, especially in the presence of noisy or complex data.

| Classifier | Accuracy | AUC |
|---|---|---|
| Decision Tree | 0.6325678 | 0.6522465 |
| Naïve Bayes | 0.3841336 | 0.6078795 |
| Bagging | 0.6430063 | 0.6342916 |
| Boosting | 0.6555324 | 0.6493306 |
| Random Forest | 0.6638831 | 0.6680356 |
| CV Bagging | 0.691137 | NA |

### Question 11

***Create Artificial Neural Network (ANN) classifier***

```
library(neuralnet)

PD.nn =neuralnet(Class==1 ~ A01 + A18 + A22 + A23 + A24, PD.train, hidden = 3, linear.output = FALSE
PD.pred =compute(PD.nn, PD.test[,c("A01", "A18", "A22", "A23", "A24")])
nn.prob  = PD.pred$net.result
nn.pred = ifelse(nn.prob > 0.5 , 1, 0)
nn_table = table(observed = PD.test$Class, predicted = nn.pred)
nn_acc = sum(diag(nn_table)) / sum(nn_table)

plot(PD.nn, rep = "best")
```




Error: 109.642589 Steps: 36046

From question 2, I handled the missing values and created a new dataframe called 'cleaned_PD'. Since all the attributes consist of integer, double, and factor data types, which are accepted by ANN, I did not need to modify them. From question 8, I determined that the most important variables in predicting whether a website is phishing or legitimate are A01, A18, A22, A23, and A24. Therefore, I used these five attributes as my input nodes. The accuracy of my ANN is **0.6450939**. It ranks in the middle of the accuracy list, similar to bagging. The accuracy may be low due to overfitting. However, unlike tree-based models, ANNs can model complex, non-linear relationships in the data due to their multi-layer structure. Additionally, ANNs can process information in parallel across multiple neurons, which leads to improved accuracy.

| Classifier | Accuracy | AUC |
|---|---|---|
| Decision Tree | 0.6325678 | 0.6522465 |
| Naïve Bays | 0.3841336 | 0.6078795 |
| Bagging | 0.6430063 | 0.6342916 |
| Boosting | 0.6555324 | 0.6493306 |
| Random Forest | 0.6638831 | 0.6680356 |
| ANN | 0.6450939 | NA |

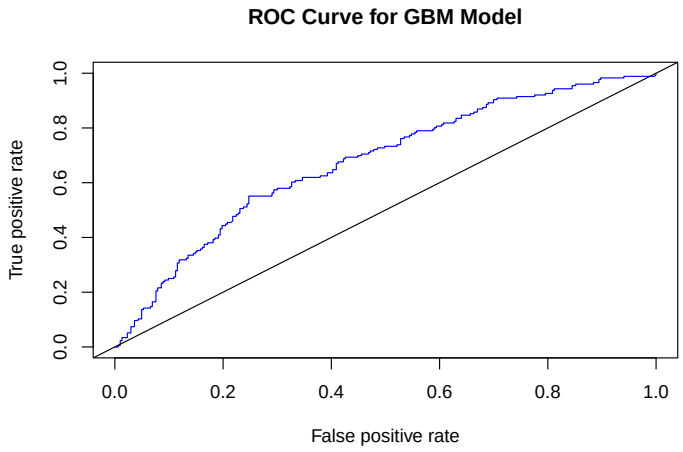
### Question 12

***Create new classifier***

```
detach(package:neuralnet)
library(gbm)
library(caret)
library(ROCR)

# Fit GBM model
gbm_model = train(Class ~ ., data = PD.train, method = "gbm")
```

```
# Check accuracy
gbm_pred = predict(gbm_model, PD.test)
gbm_table = table(observed = PD.test$Class, predicted = gbm_pred)
gbm_acc = sum(diag(gbm_table)) / sum(gbm_table)

# Check confidence & ROC
gbm_prob = predict(gbm_model, PD.test, type = "prob")
```

```r
gbm_pred = prediction(gbm_prob[,2], PD.test$Class)
gbm_perf = performance(gbm_pred, "tpr", "fpr")

# Plot ROC Curve
plot(gbm_perf, col = "blue", main = "ROC Curve for GBM Model")
abline(0, 1, col = "black")
```




```r
# Calculate AUC
gbm_auc = performance(gbm_pred, "auc")
gbm_auc_y = as.numeric(gbm_auc@y.values)
```

I developed a new classifier called the Gradient Boosting Machine (GBM). Before implementing it, I installed and loaded the 'gbm' package. Here is the weblink to the package details: https://www.rdocumentation.org/packages/gbm/versions/2.1.9.

In the gradient boosting algorithm (GBM), each new tree is fitted on a modified version of the data. Gradient boosting trains models in a gradual and sequential manner, using gradients of the loss function to identify the shortcomings of weak learners. The loss function measures how well the model's predictions fit the data and varies based on the optimization goal. One of the key advantages of gradient boosting is its ability to optimize a user-specified cost function. This sequential learning and optimization process enables gradient boosting to iteratively reduce errors and enhance model performance.

Regarding the performance of this new model, it achieved an accuracy of **0.6555324** and AUC of **0.6758363**. It has the slightly similar accuracy with random forest in question 4, but surprisingly, the AUC turned out to be the highest. Despite this, the GBM's overall performance is strong and demonstrates its effectiveness in analyzing the dataset.

| Classifier | Accuracy | AUC |
|---|---|---|
| Decision Tree | 0.6325678 | 0.6522465 |
| Naïve Bays | 0.3841336 | 0.6078795 |
| Bagging | 0.6430063 | 0.6342916 |
| Boosting | 0.6555324 | 0.6493306 |
| Random Forest | 0.6638831 | 0.6680356 |
| New Model (GBM) | 0.6555324 | 0.6758363 |

### Reference

Singh, H. (2018). Understanding Gradient Boosting Machines. Retrieved May 16, 2024 from https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab.