FIT3152 – Data Analytics

MONASH University

Phishing Website Classification

Building and evaluating classification models for cybersecurity

Joanna Moy

FIT3152 Data Analytics - Assessment 2

Joanna Moy (32694547)

2024-05-18

Introduction

Phishing is a cyber attack method that involves tricking individuals into providing personal information by posing as legitimate websites. This report explores classification models to predict whether a website is safe or designed for phishing, which aims to steal personal data from users. By using a modified version of the PhiUSIIL Phishing URL dataset from the UCI Machine Learning Archive, tasks such as data pre-processing, exploratory analysis and model building will be performed. The main objective is to understand how these models can help in identifying phishing websites and improve cyber security.

Generative AI was not used in the assignment.

Question 1

```
# Proportion of phishing sites to legitimate sites
table(PD$Class)
```

```
##
##    0    1
## 1298  702
```

```
prop.table(table(PD$Class))
```

```
##
##     0     1
## 0.649 0.351
```

From this output, the first table represents the counts of legitimate (0) and phishing (1) sites, while the second table represents the proportions. As observed, there is a noticeable class imbalance, with legitimate sites making up approximately 65% of the dataset and phishing sites making up approximately 35%.

```
# Descriptive statistics for real-valued attributes (excluding 'Class')
summary(PD)
```

```
##       A01             A02               A03                A04
##  Min.   : 2.00   Min.   : 0.0000   Min.   :0.000000   Min.   :2.00
##  1st Qu.:14.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.00
##  Median :22.00   Median : 0.0000   Median :0.000000   Median :3.00
##  Mean   :24.09   Mean   : 0.1478   Mean   :0.001011   Mean   :2.74
```

```
##  3rd Qu.:33.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.00
##  Max.   :49.00   Max.   :28.0000   Max.   :1.000000   Max.   :7.00
##                  NA's   :18        NA's   :22         NA's   :23
##       A05                A06              A07                A08
##  Min.   : 0.00000   Min.   :0.0000   Min.   :0.000000   Min.   :0.1818
##  1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6842
##  Median : 0.00000   Median :0.0000   Median :0.000000   Median :1.0000
##  Mean   : 0.01257   Mean   :0.1253   Mean   :0.000506   Mean   :0.8538
##  3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :12.00000   Max.   :1.0000   Max.   :1.000000   Max.   :1.0000
##  NA's   :12         NA's   :20       NA's   :24         NA's   :18
##       A09               A10              A11               A12
##  Min.   :0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :111.0
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:232.0
##  Median :0.00000   Median :0.00000   Median : 0.0000   Median :232.0
##  Mean   :0.03239   Mean   :0.03881   Mean   : 0.0502   Mean   :311.2
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:383.0
##  Max.   :1.00000   Max.   :1.00000   Max.   :11.0000   Max.   :692.0
##  NA's   :24        NA's   :16        NA's   :28        NA's   :21
##       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   : 0.006054   Mean   :0.1322   Mean   :0.1302   Mean   :0.04398
##  3rd Qu.: 0.000000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :12.000000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##  NA's   :18          NA's   :18       NA's   :18       NA's   :22
##       A17              A18               A19              A20
##  Min.   :0.000   Min.   :   4.00   Min.   :0.000   Min.   :0.0000
##  1st Qu.:1.000   1st Qu.:  14.00   1st Qu.:0.000   1st Qu.:0.0000
##  Median :1.000   Median :  31.00   Median :0.000   Median :0.0000
##  Mean   :1.151   Mean   :  57.05   Mean   :0.113   Mean   :0.2444
##  3rd Qu.:1.000   3rd Qu.:  89.00   3rd Qu.:0.000   3rd Qu.:0.0000
##  Max.   :5.000   Max.   :1193.00   Max.   :1.000   Max.   :1.0000
##  NA's   :22      NA's   :27        NA's   :18      NA's   :24
##       A21               A22               A23               A24
##  Min.   :0.0000   Min.   :0.01066   Min.   :   0.00   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.05178   1st Qu.:  13.00   1st Qu.:0.00820
##  Median :0.0000   Median :0.05849   Median : 100.00   Median :0.07996
##  Mean   :0.0258   Mean   :0.05638   Mean   :  73.27   Mean   :0.27144
##  3rd Qu.:0.0000   3rd Qu.:0.06327   3rd Qu.: 107.00   3rd Qu.:0.52291
##  Max.   :3.0000   Max.   :0.08507   Max.   :1137.00   Max.   :0.52291
##  NA's   :23       NA's   :18        NA's   :26        NA's   :17
##       A25              Class
##  Min.   :0.0e+00   Min.   :0.000
##  1st Qu.:0.0e+00   1st Qu.:0.000
##  Median :0.0e+00   Median :0.000
##  Mean   :2.6e-05   Mean   :0.351
##  3rd Qu.:0.0e+00   3rd Qu.:1.000
##  Max.   :5.1e-02   Max.   :1.000
##  NA's   :16
```

```
sapply(PD[, 2:26], function(x) if(is.numeric(x)) c(mean = mean(x), sd = sd(x)) else NA)
```

```
##      A02 A03 A04 A05 A06 A07 A08 A09 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19
## mean  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
## sd    NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
##      A20 A21 A22 A23 A24 A25     Class
## mean  NA  NA  NA  NA  NA  NA 0.3510000
## sd    NA  NA  NA  NA  NA  NA 0.4774023
```

For obtaining the descriptions of the predictor variables for real-valued attributes, the summary statistic function was used. However, a boxplot was created for better visualisation of this statistic.

```
library(ggplot2)
```

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

```
# Melt the data for ggplot2
library(reshape2)
```

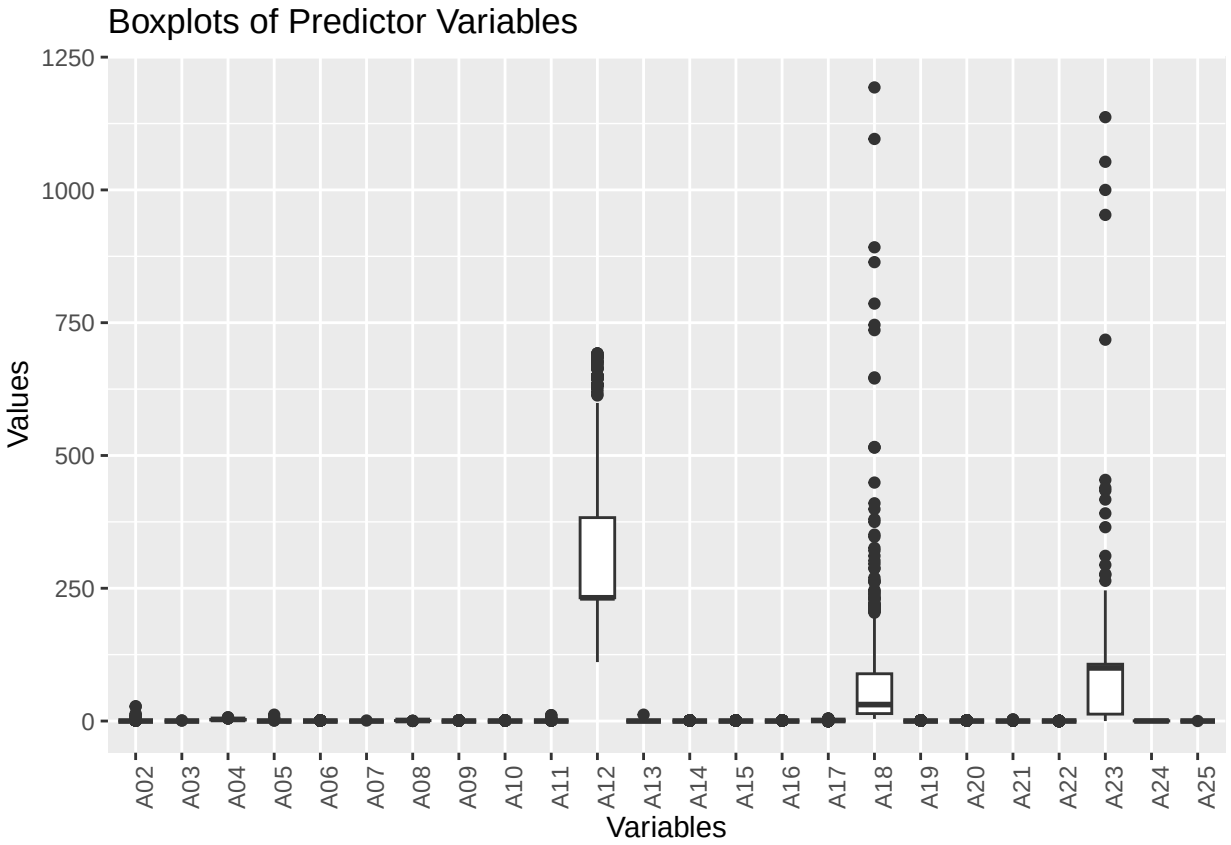
```
## Warning: package 'reshape2' was built under R version 4.3.3
```

```
PD_melted <- melt(PD[, 2:25])
```

```
## No id variables; using all as measure variables
```

```
# Boxplot
ggplot(PD_melted, aes(x = variable, y = value)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Boxplots of Predictor Variables", x = "Variables", y = "Values")
```

```
## Warning: Removed 493 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



As noted from the summary statistics and the boxplot, A02, A05, A11, A13, A12, A17, A18, and A23 have a wider range of values and several outliers compared to the other variables, indicating that some data points are significantly different from the rest.Variables like A12, A18, and A23 shows a skewed distribution with many outliers on the higher end while other variables with compact boxes and short whiskers indicate low variance and tightly clustered values around the median.

Additionally from the summary statistics, attributes with low variance and high number of missing values (NA) such as: A02, A03, A04, A05, A06, A07, A08, A09, A10, A11, A13, A14, A15, A16, A17, A19, A20, A21, A22, A24, and A25 were considered for omission has they may not be informative.

Question 2

To prepare the dataset for model fitting, several pre-processing steps were done. Firstly, as stated in the previous question, attributes that exhibit little to no variance are to be removed as they may contribute minimally to the model. Secondly, we ensured that the class column is a factor. After that, the dataset was checked for missing values. From there, it was decided whether to remove these rows depending on the proportion and importance of the missing data. Next, to handle outliers, those that had zero variance predictors were removed as well. For this data splitting and pre-processing, the 'dplyr' and 'caret' libraries will be used.

```
library(dplyr)
```

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

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

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

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

```
library(caret)
```

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

## Loading required package: lattice

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

```
# Attributes to omit
omit_vars <- c("A02", "A03", "A04", "A05", "A06", "A07", "A08", "A09", "A10",
               "A11", "A13", "A14", "A15", "A16", "A17", "A19", "A20", "A21",
               "A22", "A24", "A25")

PD_clean <- PD[, !(names(PD) %in% omit_vars)]

# Ensure Class column is a factor
PD_clean$Class <- as.factor(PD_clean$Class)

# Remove rows with missing values
PD_clean <- na.omit(PD_clean)

# Remove zero variance predictors
nzv <- nearZeroVar(PD_clean, saveMetrics = TRUE)
nzv_vars <- rownames(nzv[nzv$nzv, ])
PD_clean <- PD_clean[, !(names(PD_clean) %in% nzv_vars)]
```

Question 3

```
# Set the seed for reproducibility
set.seed(32694547) # Student ID as random seed
train.row = sample(1:nrow(PD_clean), 0.7 * nrow(PD_clean))
PD_train = PD_clean[train.row, ]
PD_test = PD_clean[-train.row, ]

# Check the dimensions of the training and test sets
dim(PD_train)
```

```
## [1] 1348    5
```

```
dim(PD_test)
```

```
## [1] 579   5
```

Now, that the data has been split, we now have 1348 rows and 5 columns for the training set and 579 rows and 5 columns for the test set.

Question 4

For the following questions that implements the classification models, it involves installing several packages beforehand. However, since the packages have already been installed beforehand, they will be commented out.

```
# install.packages("rpart")
# install.packages("e1071")
# install.packages("ipred")
# install.packages("adabag")
# install.packages("randomForest")
```

Decision Tree

```
# Implementing decision Tree
library(rpart)
```

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

```
# Train a Decision Tree model
decision_tree_model <- rpart(Class ~ ., data = PD_train, method = "class")

# Predict on the test set
pred_dt <- predict(decision_tree_model, PD_test, type = "class")
```

Naïve Bayes

```
# Implementing Naive Bayes
library(e1071)
```

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

```
# Train a Naive Bayes model
naive_bayes_model <- naiveBayes(Class ~ ., data = PD_train)

# Predict on the test set
pred_nb <- predict(naive_bayes_model, PD_test)
```

Bagging

```
# Implementing bagging
library(ipred)
```

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

```r
# Train a Bagging model
bagging_model <- bagging(Class ~ ., data = PD_train, coob = TRUE)

# Predict on the test set
pred_bagging <- predict(bagging_model, PD_test)
```

Boosting

```r
# Implementing boosting
library(adabag)
```

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

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

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

```
## Loading required package: doParallel
```

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

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

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

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

```
##
## Attaching package: 'adabag'
```

```
## The following object is masked from 'package:ipred':
##
##     bagging
```

```r
boosting_model <- boosting(Class ~ ., data = PD_train, mfinal = 10)

# Predict on the test set
pred_boosting <- predict(boosting_model, newdata = PD_test)
```

Random Forest

```r
# Implementing Random forest
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:dplyr':
##
##     combine

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

```
# Train a Random Forest model
random_forest_model <- randomForest(Class ~ ., data = PD_train)

# Predict on the test set
pred_rf <- predict(random_forest_model, PD_test)
```

Question 5

Decision Tree

```
# Confusion matrix and accuracy for Decision Tree
confusion_matrix_dt <- table(PD_test$Class, pred_dt)
accuracy_dt <- sum(diag(confusion_matrix_dt)) / sum(confusion_matrix_dt)
print(confusion_matrix_dt)
```

```
##    pred_dt
##       0   1
##   0 323  46
##   1 116  94
```

```
print(paste("Accuracy for Decision Tree:", round(accuracy_dt, 4)))
```

```
## [1] "Accuracy for Decision Tree: 0.7202"
```

Naïve Bayes

```
# Confusion matrix and accuracy for Naive Bayes
confusion_matrix_nb <- table(PD_test$Class, pred_nb)
accuracy_nb <- sum(diag(confusion_matrix_nb)) / sum(confusion_matrix_nb)
print(confusion_matrix_nb)
```

```
##    pred_nb
##       0   1
##   0 284  85
##   1 118  92
```

```
print(paste("Accuracy for Naive Bayes:", round(accuracy_nb, 4)))
```

```
## [1] "Accuracy for Naive Bayes: 0.6494"
```

Bagging

```
# Confusion matrix and accuracy for Bagging
confusion_matrix_bagging <- table(PD_test$Class, pred_bagging)
accuracy_bagging <- sum(diag(confusion_matrix_bagging)) / sum(confusion_matrix_bagging)
print(confusion_matrix_bagging)
```

```
##    pred_bagging
##       0   1
##   0 297  72
##   1 107 103
```

```
print(paste("Accuracy for Bagging:", round(accuracy_bagging, 4)))
```

```
## [1] "Accuracy for Bagging: 0.6908"
```

Boosting

```
# Confusion matrix and accuracy for Boosting
confusion_matrix_boosting <- table(PD_test$Class, pred_boosting$class)
accuracy_boosting <- sum(diag(confusion_matrix_boosting)) / sum(confusion_matrix_boosting)
print(confusion_matrix_boosting)
```

```
##
##       0   1
##   0 317  52
##   1 109 101
```

```
print(paste("Accuracy for Boosting:", round(accuracy_boosting, 4)))
```

```
## [1] "Accuracy for Boosting: 0.7219"
```

Random Forest

```
# Confusion matrix and accuracy for Random Forest
confusion_matrix_rf <- table(PD_test$Class, pred_rf)
accuracy_rf <- sum(diag(confusion_matrix_rf)) / sum(confusion_matrix_rf)
print(confusion_matrix_rf)
```

```
##    pred_rf
##       0   1
##   0 302  67
##   1 109 101
```

```r
print(paste("Accuracy for Random Forest:", round(accuracy_rf, 4)))
```

```
## [1] "Accuracy for Random Forest: 0.696"
```

Question 6

```r
library(rpart)
library(e1071)
library(ipred)
library(adabag)
library(randomForest)
library(ROCR)
```

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

```r
library(caret)
```

To assess this question, the confidence of predicting 'phishing' for each case will be calculated. Following that, ROC curves will be constructed for each classifier, plotting all the curves on the same axis with different colours. Additionally, the AUC for each classifier will be shown. ### Decision Tree

```r
# Train a Decision Tree model
decision_tree_model <- rpart(Class ~ ., data = PD_train, method = "class")

# Predict on the test set
pred_dt_prob <- predict(decision_tree_model, PD_test, type = "prob")[, 2] # Probability of class 1
```

Naïve Bayes

```r
# Train a Naive Bayes model
naive_bayes_model <- naiveBayes(Class ~ ., data = PD_train)

# Predict on the test set
pred_nb_prob <- predict(naive_bayes_model, PD_test, type = "raw")[, 2] # Probability of class 1
```

Bagging

```r
# Train a Bagging model
bagging_model <- train(Class ~ ., data = PD_train, method = "treebag")

# Predict on the test set
pred_bagging_prob <- predict(bagging_model, PD_test, type = "prob")[, 2] # Probability of class 1
```

Boosting

```
# Train a Boosting model
boosting_model <- boosting(Class ~ ., data = PD_train, mfinal = 10)

# Predict on the test set
pred_boosting_prob <- predict(boosting_model, newdata = PD_test)$prob[, 2] # Probability of class 1
```

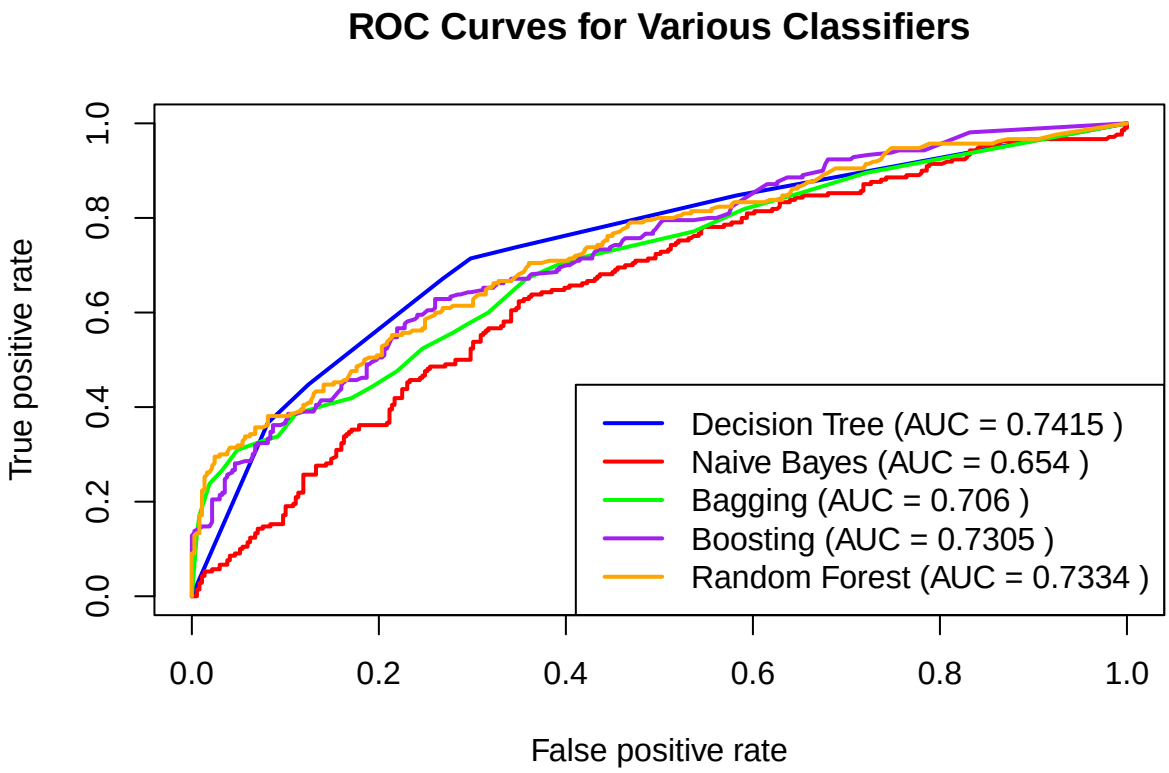
Random Forest

```
# Train a Random Forest model
random_forest_model <- randomForest(Class ~ ., data = PD_train)

# Predict on the test set
pred_rf_prob <- predict(random_forest_model, PD_test, type = "prob")[, 2] # Probability of class 1
```

Constructing ROC Curves and Calculating AUC

```
# Create prediction objects for each model
pred_dt_roc <- prediction(pred_dt_prob, PD_test$Class)
pred_nb_roc <- prediction(pred_nb_prob, PD_test$Class)
pred_bagging_roc <- prediction(pred_bagging_prob, PD_test$Class)
pred_boosting_roc <- prediction(pred_boosting_prob, PD_test$Class)
pred_rf_roc <- prediction(pred_rf_prob, PD_test$Class)

# Create performance objects for ROC curves
perf_dt <- performance(pred_dt_roc, "tpr", "fpr")
perf_nb <- performance(pred_nb_roc, "tpr", "fpr")
perf_bagging <- performance(pred_bagging_roc, "tpr", "fpr")
perf_boosting <- performance(pred_boosting_roc, "tpr", "fpr")
perf_rf <- performance(pred_rf_roc, "tpr", "fpr")

# Calculate AUC
auc_dt <- performance(pred_dt_roc, "auc")@y.values[[1]]
auc_nb <- performance(pred_nb_roc, "auc")@y.values[[1]]
auc_bagging <- performance(pred_bagging_roc, "auc")@y.values[[1]]
auc_boosting <- performance(pred_boosting_roc, "auc")@y.values[[1]]
auc_rf <- performance(pred_rf_roc, "auc")@y.values[[1]]

# Plot ROC curves
plot(perf_dt, col = "blue", lwd = 2, main = "ROC Curves for Various Classifiers")
plot(perf_nb, col = "red", lwd = 2, add = TRUE)
plot(perf_bagging, col = "green", lwd = 2, add = TRUE)
plot(perf_boosting, col = "purple", lwd = 2, add = TRUE)
plot(perf_rf, col = "orange", lwd = 2, add = TRUE)
legend("bottomright", legend = c(paste("Decision Tree (AUC =", round(auc_dt, 4), ")"),
                                 paste("Naive Bayes (AUC =", round(auc_nb, 4), ")"),
                                 paste("Bagging (AUC =", round(auc_bagging, 4), ")"),
                                 paste("Boosting (AUC =", round(auc_boosting, 4), ")"),
```

```
                                  paste("Random Forest (AUC =", round(auc_rf, 4), ")")),
       col = c("blue", "red", "green", "purple", "orange"), lwd = 2)
```



Question 7

For this question, a table has been created to compare the results from question 5 and 6 for all classifiers.Based on the calculated accuracy and AUC, the best classifier based on accuracy was determined as the Boosting model and the best classifier based on AUC was identified as the Decision Tree model.

```
# Create a data frame to store the results
results <- data.frame(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
  Accuracy = c(accuracy_dt, accuracy_nb, accuracy_bagging, accuracy_boosting, accuracy_rf),
  AUC = c(auc_dt, auc_nb, auc_bagging, auc_boosting, auc_rf)
)

# Print the results table
print(results)
```

```
##      Classifier  Accuracy       AUC
## 1 Decision Tree 0.7202073 0.7414828
## 2   Naive Bayes 0.6493955 0.6540005
## 3       Bagging 0.6908463 0.7059750
## 4      Boosting 0.7219344 0.7305265
```

```
## 5 Random Forest 0.6960276 0.7333527
```

```
# Determine the best classifier based on Accuracy and AUC
best_accuracy <- results[which.max(results$Accuracy), ]
best_auc <- results[which.max(results$AUC), ]

print("Best Classifier based on Accuracy:")
```

```
## [1] "Best Classifier based on Accuracy:"
```

```
print(best_accuracy)
```

```
##   Classifier  Accuracy       AUC
## 4   Boosting 0.7219344 0.7305265
```

```
print("Best Classifier based on AUC:")
```

```
## [1] "Best Classifier based on AUC:"
```

```
print(best_auc)
```

```
##      Classifier  Accuracy       AUC
## 1 Decision Tree 0.7202073 0.7414828
```

Question 8

Decision Tree

Based on the output for the decision tree model, A23 and A18 are the most important variables as they have significantly higher importance scores. These will be the variables that will contribute the most to predicting whether a website will be phishing or legitimate. However, since A12 has a lower importance score compared to other variables, it could be considered for omission with minimal impact on the model's performance.

```
# Variable importance for Decision Tree
library(rpart)
decision_tree_importance <- varImp(decision_tree_model)
print("Variable importance for Decision Tree:")
```

```
## [1] "Variable importance for Decision Tree:"
```

```
print(decision_tree_importance)
```

```
##       Overall
## A01  97.35914
## A12  21.64752
## A18 105.19355
## A23 108.95071
```

Naïve Bayes

For the Naive Bayes model's output, the interpretation of the variable importance can be examined by the differences in conditional probabilities for each class across the variables. Based on the output, A01 and A23 appear to be the most importance variables due to their significant differences in values between the classes. The A12 variable could be considered for omission because its values for both classes have a relatively bigger difference, indicating that it might have less influence on the model's ability to distinguish between phishing and legitimate websites. However, since the difference is not that much, it could also be considered not to be omitted.

```
# Variable importance for Naive Bayes
library(e1071)
naive_bayes_importance <- naive_bayes_model$tables
print("Variable importance for Naive Bayes:")
```

```
## [1] "Variable importance for Naive Bayes:"
```

```
print(naive_bayes_importance)
```

```
## $A01
##    A01
## Y       [,1]     [,2]
##   0 20.82054 13.02377
##   1 30.35065 12.20711
##
## $A12
##    A12
## Y       [,1]     [,2]
##   0 306.5023 138.2821
##   1 320.9978 141.9324
##
## $A18
##    A18
## Y       [,1]     [,2]
##   0 57.35440 96.75270
##   1 63.31169 57.03964
##
## $A23
##    A23
## Y       [,1]     [,2]
##   0 74.21106 63.65223
##   1 71.05411 76.69218
```

Bagging

The bagging model, on the other hand, shows that A18 has the highest importance score, indicating it is the most important variable for the bagging model. Additionally, A23 can also be considered as highly important with a score of 72.87287. However, since A01 and A12 have significantly lower important scores, this means that they can be omitted from the data with no effect on the model's performance due to its low importance scores.

```r
# Variable importance for Bagging
library(ipred)
bagging_importance <- varImp(bagging_model)
print("Variable importance for Bagging:")
```

```
## [1] "Variable importance for Bagging:"
```

```r
print(bagging_importance)
```

```
## treebag variable importance
##
##     Overall
## A18 100.000
## A23  72.873
## A01   1.755
## A12   0.000
```

Boosting

For the boosting model, A01 seems to have the highest importance score, indicating it is the most important variable for the boosting model. Additionally, A23 and A18 have similar importance scores, showing that they are also important but less so than A01. Therefore, since A12 has the lowest important scores, it suggests that it has the least impact on the model's performance amonng the listed variables.

```r
# Variable importance for Boosting
library(adabag)
boosting_importance <- boosting_model$importance
print("Variable importance for Boosting:")
```

```
## [1] "Variable importance for Boosting:"
```

```r
print(boosting_importance)
```

```
##       A01       A12       A18       A23
## 41.971996  9.765093 23.792604 24.470307
```

Random Forest

Based on the output for the random forest model, A18 has the highest important score, indicating that it is the most important variable for this model. Similar to the boosting model, A23 and A01 have similar important scores, showing that they are also highly important. However, even though A12 has the lowest important score amongst the other variables, it still has a considerable impact and so should be considered for omission only after further validation across other models to ensure it does no significantly affect the overall performance.

```r
# Variable importance for Random Forest
library(randomForest)
random_forest_importance <- importance(random_forest_model)
print("Variable importance for Random Forest:")
```

```
## [1] "Variable importance for Random Forest:"
```

```
print(random_forest_importance)
```

```
##     MeanDecreaseGini
## A01        148.29092
## A12         93.52838
## A18        181.74437
## A23        148.62381
```

Question 9

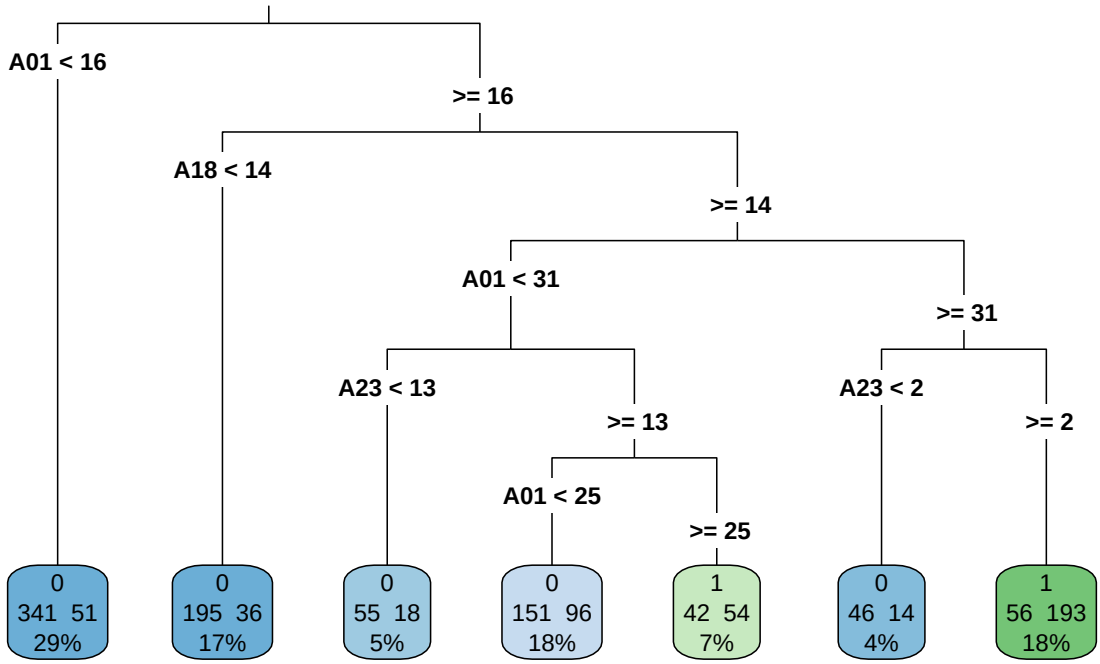
To create a simple, human-interpretable classifier, the Decision Tree model can be used as it can be easily visualised, making it suitable for manual classification. First, the decision tree will be pruned to make it simpler and more interpretable. Then, a diagram of the pruned tree will be created. Finally, the simplified model's performance will be compared to that of the original models.

For the first step, the 'rpart.plot' function will generate a diagram of the pruned Decision Tree as shown below. The diagram displays the decision rules and the outcomes at each node. The attributes A01, A18, and A23 used were selected based on their importance in the original, unpruned model as we evaluated from Question 8. These attributes have shown to be significant in distinguishing between phishing and legitimate websites

```
# install.packages("rpart.plot")
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.3.3
```

```
library(ROCR)

# Train a Decision Tree model
decision_tree_model <- rpart(Class ~ ., data = PD_train, method = "class")

# Prune the Decision Tree
pruned_tree <- prune(decision_tree_model, cp = 0.01)

# Visualize the pruned Decision Tree
rpart.plot(pruned_tree, main = "Pruned Decision Tree", type = 3, extra = 101)
```



To evaluate this pruned model, we can compare its performance metrics, such as accuracy, confusion matrix, ROC curve, and AUC, with the original models from Question 4.Given the confusion matrix for the pruned decision tree, we can conclude that the accuracy of the pruned decision tree is 0.7202. Additionally the AUC for the pruned decision tree is 0.7415. Therefore, when comparing the results for this pruned decision model with the original model from previous questions, they both appear to have the same results. This suggests that the pruning process has successfully simplified the model without compromising its performance.

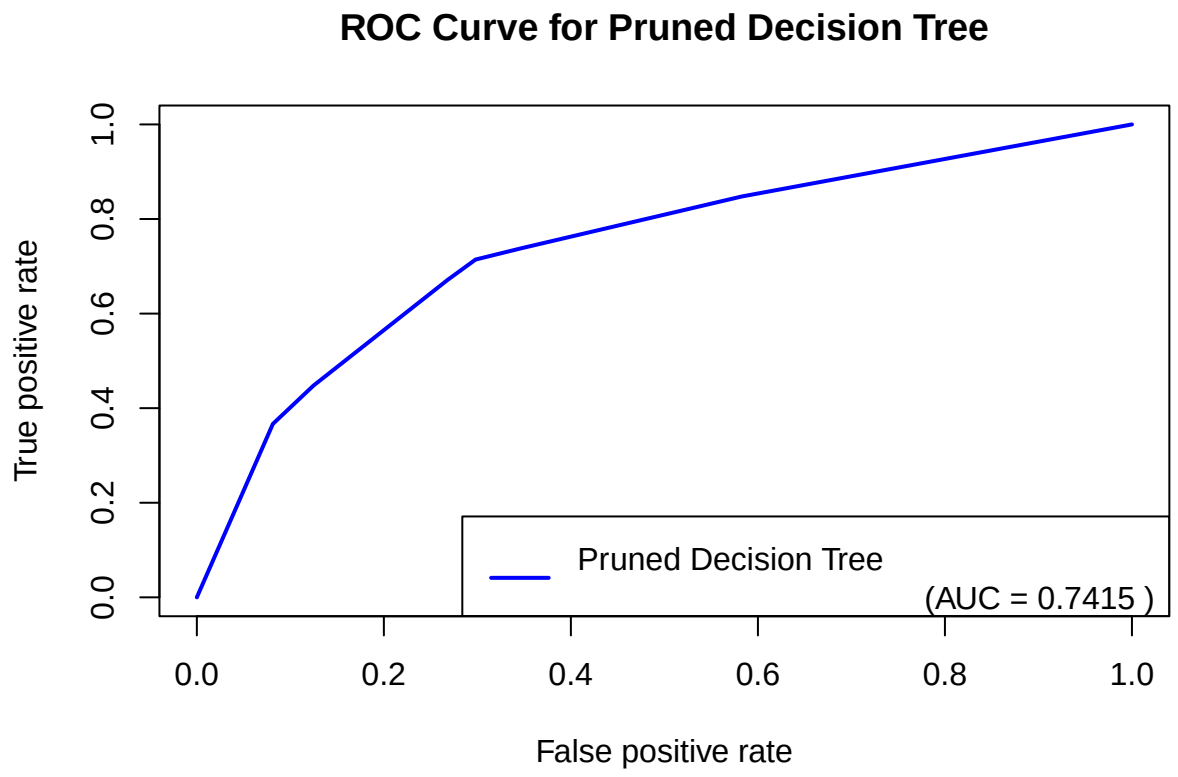
```
# Predict on the test set using the pruned Decision Tree
pred_pruned_dt <- predict(pruned_tree, PD_test, type = "class")

# Confusion matrix and accuracy
confusion_matrix_pruned_dt <- table(PD_test$Class, pred_pruned_dt)
accuracy_pruned_dt <- sum(diag(confusion_matrix_pruned_dt)) / sum(confusion_matrix_pruned_dt)
print(confusion_matrix_pruned_dt)
```

```
##    pred_pruned_dt
##       0   1
##   0 323  46
##   1 116  94
```

```
print(paste("Accuracy for Pruned Decision Tree:", round(accuracy_pruned_dt, 4)))
```

```
## [1] "Accuracy for Pruned Decision Tree: 0.7202"
```

```r
# ROC curve and AUC
library(ROCR)

# Predict probabilities for ROC curve
pred_pruned_dt_prob <- predict(pruned_tree, PD_test, type = "prob")[, 2]

# Create prediction object
pred_pruned_dt_roc <- prediction(pred_pruned_dt_prob, PD_test$Class)

# Create performance object for ROC curve
perf_pruned_dt <- performance(pred_pruned_dt_roc, "tpr", "fpr")

# Calculate AUC
auc_pruned_dt <- performance(pred_pruned_dt_roc, "auc")@y.values[[1]]

# Plot ROC curve
plot(perf_pruned_dt, col = "blue", lwd = 2, main = "ROC Curve for Pruned Decision Tree")
legend("bottomright", legend = c(paste("Pruned Decision Tree
                                       (AUC =", round(auc_pruned_dt, 4), ")")), col = "blue", lwd = 2)
```




```r
print(paste("AUC for Pruned Decision Tree:", round(auc_pruned_dt, 4)))
```

```
## [1] "AUC for Pruned Decision Tree: 0.7415"
```

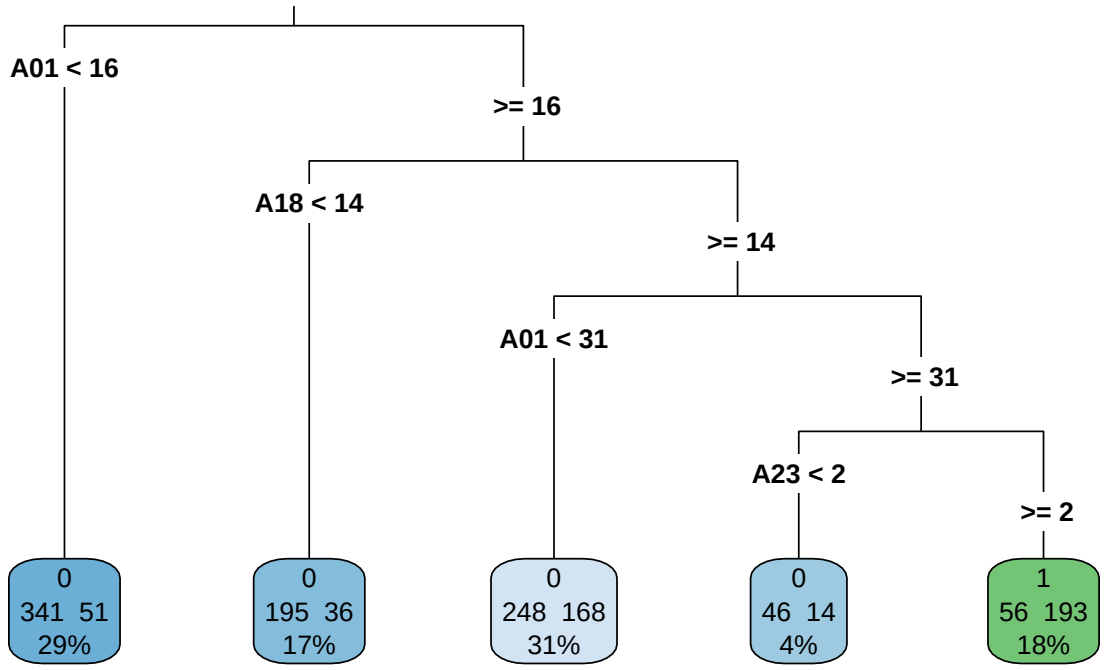
Question 10

To create the best tree-based classifier, we can improve the Decision Tree model by optimising its parameters using cross-validation and optimise the complexity parameter for the Decision Tree model. This done to find the optimal balance between model complexity and performance.

```
library(caret)
library(rpart)
library(rpart.plot)
library(ROCR)

# Set up cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the grid of hyperparameters to search
tune_grid <- expand.grid(cp = seq(0.001, 0.05, by = 0.001))

# Train the model using cross-validation
set.seed(32694547) # Ensure reproducibility
tuned_tree_model <- train(Class ~ ., data = PD_train, method = "rpart",
                          trControl = train_control, tuneGrid = tune_grid)

# Display the best cp value
best_cp <- tuned_tree_model$bestTune$cp
print(paste("Best cp value:", best_cp))
```

```
## [1] "Best cp value: 0.05"
```

```
# Train the final model with the best cp value
final_tree_model <- rpart(Class ~ ., data = PD_train, method = "class", cp = best_cp)

# Visualize the final tree
rpart.plot(final_tree_model, main = "Optimized Decision Tree", type = 3, extra = 101)
```



Here, we will compare the performance metrics of the optimised model with the original models. As displayed from the output, the optimal complexity parameter value found through cross-validation is 0.05. This parameter controls the trade-off between tree complexity and accuracy, meaning that a higher cp value could lead to a simpler tree. Additionally the accuracy of the optimised decision tree was shown to be 0.7185 and that the AUC was shown as 0.74. This indicates a moderate discriminative ability, though, slightly improved from the pruned Decision Tree but also consistent with the original model's performance.

Comparing these results with the original models, this model had just a slightly lower accuracy. However, the AUC value for this model was shown to have almost the same value as the original model. This suggests that the model complexity was effectively managed without losing much predictive power.

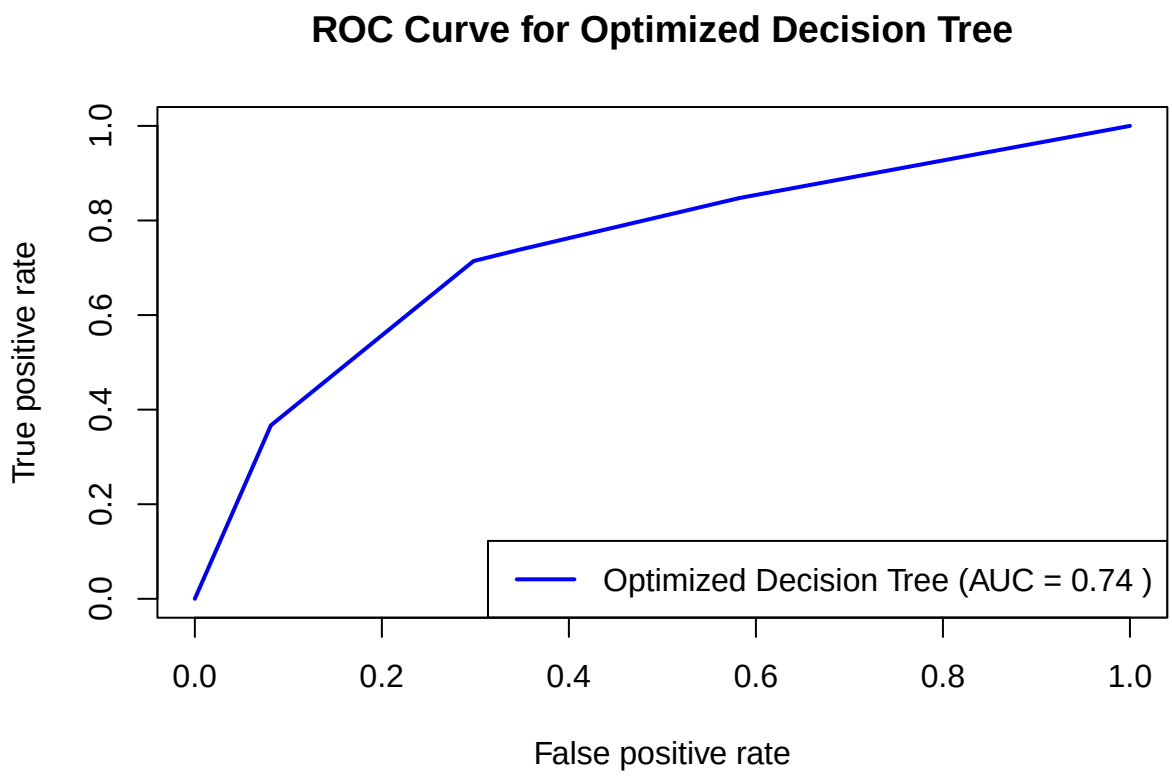
```
# Predict on the test set using the optimized Decision Tree
pred_final_tree <- predict(final_tree_model, PD_test, type = "class")

# Confusion matrix and accuracy
confusion_matrix_final_tree <- table(PD_test$Class, pred_final_tree)
accuracy_final_tree <- sum(diag(confusion_matrix_final_tree)) / sum(confusion_matrix_final_tree)
print(confusion_matrix_final_tree)
```

```
##    pred_final_tree
##       0   1
##   0 339  30
##   1 133  77
```

```
print(paste("Accuracy for Optimized Decision Tree:", round(accuracy_final_tree, 4)))
```

```
## [1] "Accuracy for Optimized Decision Tree: 0.7185"
```

```r
# ROC curve and AUC
# Predict probabilities for ROC curve
pred_final_tree_prob <- predict(final_tree_model, PD_test, type = "prob")[, 2]

# Create prediction object
pred_final_tree_roc <- prediction(pred_final_tree_prob, PD_test$Class)

# Create performance object for ROC curve
perf_final_tree <- performance(pred_final_tree_roc, "tpr", "fpr")

# Calculate AUC
auc_final_tree <- performance(pred_final_tree_roc, "auc")@y.values[[1]]

# Plot ROC curve
plot(perf_final_tree, col = "blue", lwd = 2, main = "ROC Curve for Optimized Decision Tree")
legend("bottomright", legend = c(paste("Optimized Decision Tree (AUC =", round(auc_final_tree, 4), ")"))
       col = "blue", lwd = 2)
```




```r
print(paste("AUC for Optimized Decision Tree:", round(auc_final_tree, 4)))
```

```
## [1] "AUC for Optimized Decision Tree: 0.74"
```

Question 11

To implement an Aritifical Neural Network (ANN) classifier, we'll user the 'nnet' package, which allows for creating and training neural networks. For this task, we will also consider data pre-processing, training the model, and also evaluating its performance.

All predictor variables were standardised and used in the ANN model. The importance of attributes used and the pre-processing methods undertaken, such as standardising of predictor variables, ensures that they contribute effectively to the model, and that the ANN could learn effectively. Proper tuning of parameters (size and decay) and cross-validation were also used to help mitigate over fitting.

```
library(nnet)
library(caret)
library(ROCR)

# Standardize the predictor variables
preprocess_params <- preProcess(PD_train[, -ncol(PD_train)], method = c("center", "scale"))
PD_train_scaled <- predict(preprocess_params, PD_train)
PD_test_scaled <- predict(preprocess_params, PD_test)

# Set up cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define the grid of hyperparameters to search
tune_grid <- expand.grid(size = c(5, 10, 15), decay = c(0.1, 0.01, 0.001))

# Train the ANN model using cross-validation
set.seed(32694547) # Ensure reproducibility
ann_model <- train(Class ~ ., data = PD_train_scaled,
                   method = "nnet",
                   trControl = train_control,
                   tuneGrid = tune_grid, linout = FALSE, trace = FALSE)
```

Based on the following results from the ANN Model output, the best parameters found through cross-validation were 5 for the number of neurons in the hidden layers, and 0.001 as the weight decay parameter.

Additionally, from the confusion matrix, the accuracy of the ANN model is shown as 0.7219, which is slightly higher than the optimised and pruned decision tree. The AUC of the ANN, 0.7599, was also shown to be higher than the optimised Decision Tree's AUC of 0.74, indicating better discriminative ability. Overall, comparing with other models, the ANN model's accuracy was slightly better, but remained the same accuracy with the original boosting model. Furthermore, the ANN's AUC was also better than that of the optimised Decision Tree model and other original models too.

Therefore, with the ANN model, with its slightly higher accuracy and better AUC, it suggests a strong ability to classify phishing and legitimate websites. While it is more complex and less interpretable than tree-based models, its improved performance makes it a valuable classifier.

```
# Display the best parameters
best_params <- ann_model$bestTune
print(best_params)
```

```
##   size decay
## 1    5 0.001
```

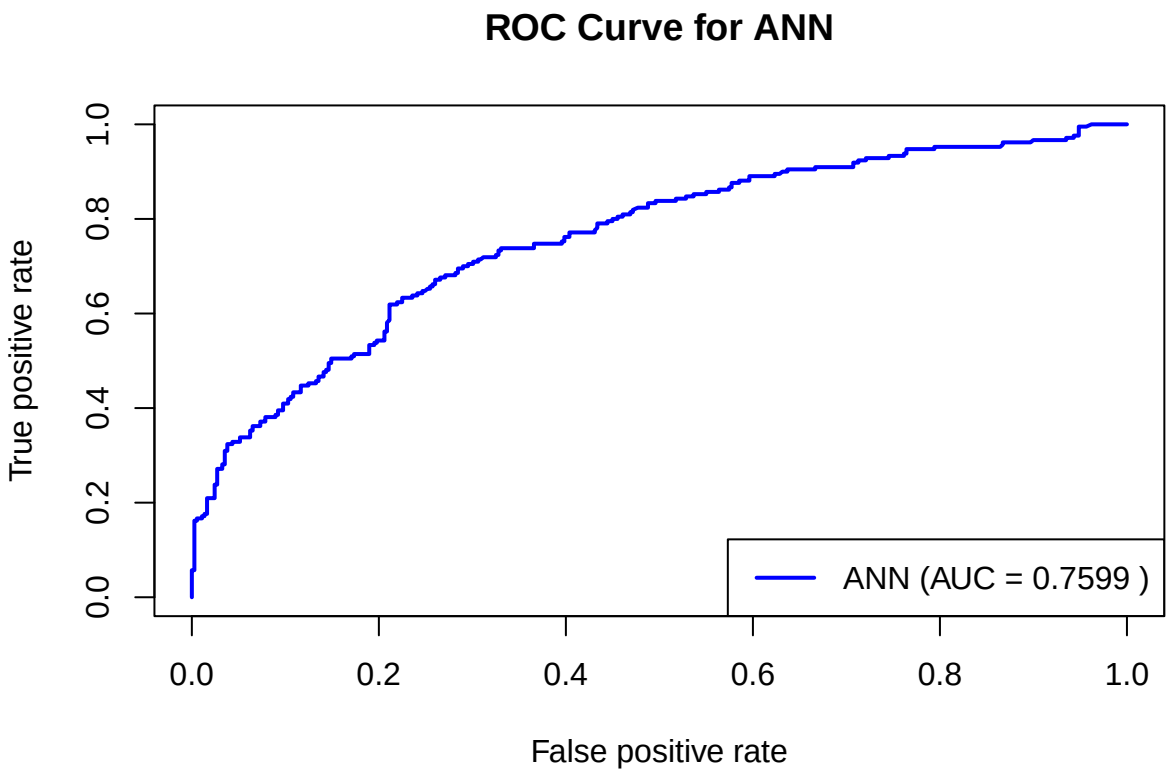
```r
# Train the final model with the best parameters
final_ann_model <- nnet(Class ~ ., data = PD_train_scaled,
                        size = best_params$size,
                        decay = best_params$decay,
                        linout = FALSE, trace = FALSE)

# Predict on the test set using the ANN model
pred_ann_prob <- predict(final_ann_model, PD_test_scaled, type = "raw")

# Convert probabilities to class labels (assuming threshold of 0.5)
pred_ann <- ifelse(pred_ann_prob > 0.5, 1, 0)

# Confusion matrix and accuracy
confusion_matrix_ann <- table(PD_test$Class, pred_ann)
accuracy_ann <- sum(diag(confusion_matrix_ann)) / sum(confusion_matrix_ann)
print(confusion_matrix_ann)
```

```
##    pred_ann
##       0   1
##   0 327  42
##   1 119  91
```

```r
print(paste("Accuracy for ANN:", round(accuracy_ann, 4)))
```

```
## [1] "Accuracy for ANN: 0.7219"
```

```r
# ROC curve and AUC
# Create prediction object
pred_ann_roc <- prediction(pred_ann_prob, PD_test_scaled$Class)

# Create performance object for ROC curve
perf_ann <- performance(pred_ann_roc, "tpr", "fpr")

# Calculate AUC
auc_ann <- performance(pred_ann_roc, "auc")@y.values[[1]]

# Plot ROC curve
plot(perf_ann, col = "blue", lwd = 2, main = "ROC Curve for ANN")
legend("bottomright", legend = c(paste("ANN (AUC =", round(auc_ann, 4), ")")),
       col = "blue", lwd = 2)
```




```
print(paste("AUC for ANN:", round(auc_ann, 4)))
```

```
## [1] "AUC for ANN: 0.7599"
```

Question 12

For this task, a Support Vector Machine (SVM) classifier will be implemented using the 'e1071' package. SVMs are powerful classifiers known for their effectiveness in high-dimensional spaces and adaptiveness to overfitting, especially in cases where the number of dimension exceeds the number of samples.

A Support Vector Machine (SVM) constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate different classes. The SVM aims to find the hyperplane that maximises the margin between the classes. Points closest to the hyperplane are called support vectors. Additionally, SVM can also use different kernel functions, such as linear, polynomial, and radial basis functions, to handle non-linear separations.

The 'e1071' package provides an interface to the SVM implementation in the 'libsvm' library. It includes functions for training SVM models, predicting with SVMs, and tuning model parameters.

The web link to the package details can be found here: https://cran.r-project.org/web/packages/e1071/index.html

```
library(e1071)
library(caret)
library(ROCR)

# Standardised data used for the ANN model will be used here as well
```

```r
preprocess_params <- preProcess(PD_train[, -ncol(PD_train)], method = c("center", "scale"))
PD_train_scaled <- predict(preprocess_params, PD_train)
PD_test_scaled <- predict(preprocess_params, PD_test)

# Set up cross-validation with fewer folds
train_control <- trainControl(method = "cv", number = 5)

# Define a smaller grid of hyperparameters to search
tune_grid <- expand.grid(sigma = 2^(-2:1), C = 2^(0:2))

# Train the SVM model using cross-validation with probability = TRUE
set.seed(32694547) # Ensure reproducibility
svm_model <- train(Class ~ ., data = PD_train_scaled, method = "svmRadial",
                   trControl = train_control, tuneGrid = tune_grid,
                   preProc = c("center", "scale"), prob.model = TRUE)

# Display the best parameters
best_params <- svm_model$bestTune
print(best_params)
```

```
##   sigma C
## 3  0.25 4
```

```r
# Train the final model with the best parameters and probability = TRUE
final_svm_model <- svm(Class ~ ., data = PD_train_scaled, kernel = "radial",
                       cost = best_params$C, gamma = best_params$sigma,
                       probability = TRUE)
```

Based on the following output, the best parameters found through cross-validation for the SVM model are 0.25 for Sigma, and 4 for cost. These parameters were selected to optimise the balance between the model's ability to fit the training data.

From the confusion matrix and calculated accuracy, we can see that the SVM model has an accuracy of 0.6926 and that the AUC with the best parameters had a value of 0.6965. This result is lower compared to some of the original classifier models like the Decision Treem Naive Bayes, Bagging, Boosting, Random Forest, and ANN. This suggests that the SVM did not perform as well as these classifiers on this dataset.

Therefore, with the SVM model, despite using optimized parameters, provided a moderate performance with an accuracy of 0.6926 and an AUC of 0.6965. While SVMs are powerful classifiers, in this particular case, they did not outperform other models like the Decision Tree, ANN, or ensemble methods.

```r
# Predict on the test set using the SVM model with probabilities
pred_svm_prob <- predict(final_svm_model, PD_test_scaled, probability = TRUE)
pred_svm_prob <- attr(pred_svm_prob, "probabilities")[, 2]

# Convert probabilities to class labels (assuming threshold of 0.5)
pred_svm <- ifelse(pred_svm_prob > 0.5, 1, 0)

# Ensure both pred_svm and PD_test$Class are factors with the same levels
pred_svm <- factor(pred_svm, levels = c(0, 1))
PD_test$Class <- factor(PD_test$Class, levels = c(0, 1))

# Confusion matrix and accuracy
```

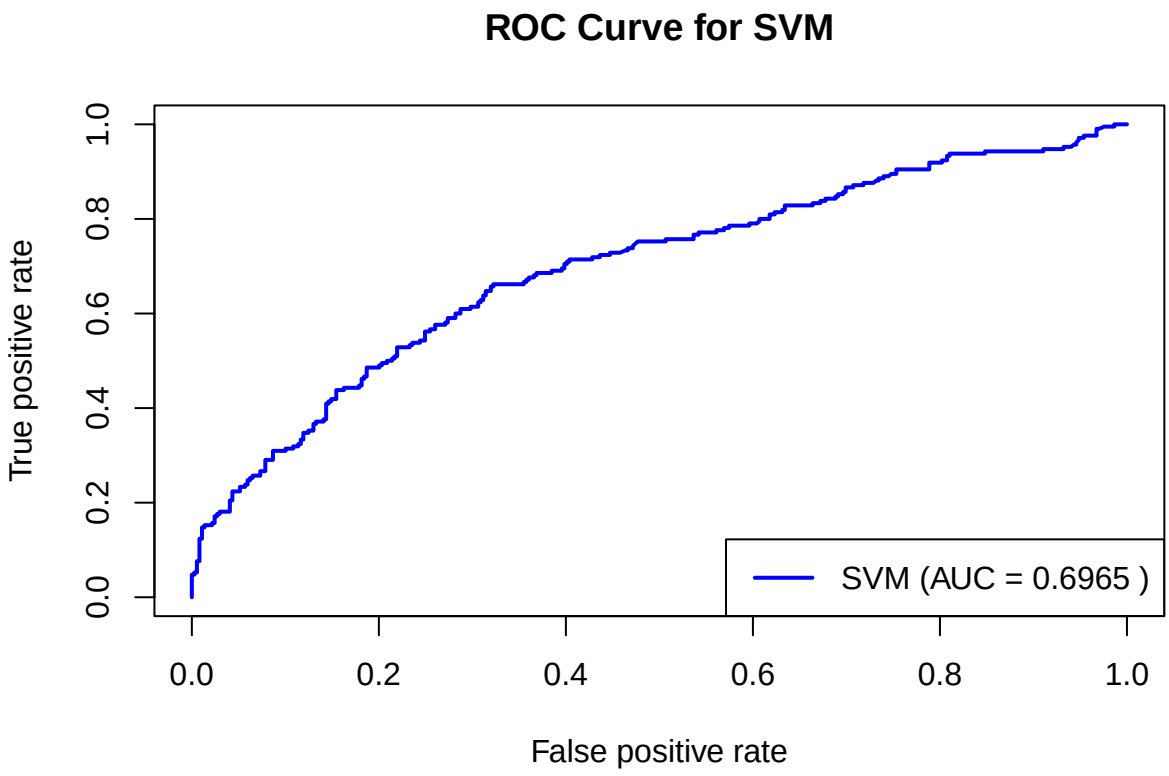
```r
confusion_matrix_svm <- table(PD_test$Class, pred_svm)
accuracy_svm <- sum(diag(confusion_matrix_svm)) / sum(confusion_matrix_svm)
print(confusion_matrix_svm)
```

```
##    pred_svm
##       0   1
##   0 337  32
##   1 146  64
```

```r
print(paste("Accuracy for SVM:", round(accuracy_svm, 4)))
```

```
## [1] "Accuracy for SVM: 0.6926"
```

```r
# ROC curve and AUC
# Create prediction object
pred_svm_roc <- prediction(pred_svm_prob, PD_test_scaled$Class)

# Create performance object for ROC curve
perf_svm <- performance(pred_svm_roc, "tpr", "fpr")

# Calculate AUC
auc_svm <- performance(pred_svm_roc, "auc")@y.values[[1]]

# Plot ROC curve
plot(perf_svm, col = "blue", lwd = 2, main = "ROC Curve for SVM")
legend("bottomright", legend = c(paste("SVM (AUC =", round(auc_svm, 4), ")")), col = "blue", lwd = 2)
```




```
print(paste("AUC for SVM:", round(auc_svm, 4)))
```

```
## [1] "AUC for SVM: 0.6965"
```