Name	Unit	Assignment
Wong Jun Wei	FIT 3152	Assignment 2

Question 1

The dataset obtained has 2,000 rows with 26 columns. Out of these 26 columns (variables), 4 are of type "num", while 22 are of type "integer", and this includes the variable of interest "Class". 1,221 out of the 2,000 rows in this dataset are phishing sites, therefore making it approximately 61.05% of the data, while the remaining 779 rows of the dataset are legitimate sites, therefore making up 38.95% of the data.

From the summary of the data, each column contains a rather low number of NA values each, averaging around 18 NA values per column.

Question 2

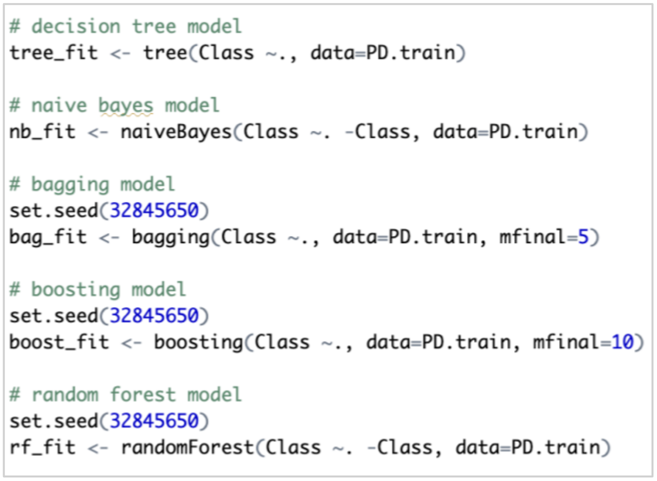
For preprocessing, all that was done was omitting NA values as the models would not be able to work with NA values later. The number of rows left after omitting was 1,571 from 2,000. The variable "Class" was converted into a factor data type. This is because the modeling functions in R assume that categorical variables are represented as factors. Therefore, using them as numeric or character types could potentially lead to incorrect results or misinterpretations.

Question 3

I divided the dataset into portions of 70% and 30% for training and testing. The training dataset has 1,099 rows while the testing dataset has 472 rows. Both datasets have 26 variables (which includes the "Class" target variable).

Question 4

5 models were created. (Decision Tree, Naïve Bayes Classifier, bagging model, boosting model, and random forest model).



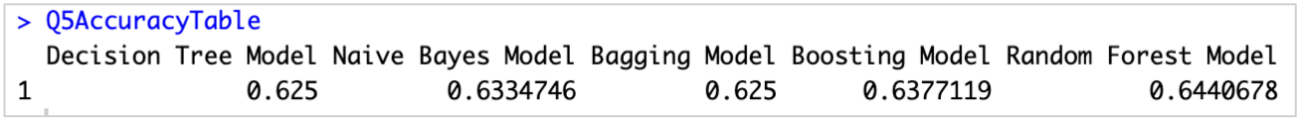
```
# decision tree model
tree_fit <- tree(Class ~., data=PD.train)

# naive bayes model
nb_fit <- naiveBayes(Class ~. -Class, data=PD.train)

# bagging model
set.seed(32845650)
bag_fit <- bagging(Class ~., data=PD.train, mfinal=5)

# boosting model
set.seed(32845650)
boost_fit <- boosting(Class ~., data=PD.train, mfinal=10)

# random forest model
set.seed(32845650)
rf_fit <- randomForest(Class ~. -Class, data=PD.train)
```

Question 5

Here, I calculated the accuracy scores of each model. The accuracy was calculated based on this formula below:

$$Accuracy = \frac{(True\ Positive + True\ Negative)}{(True\ Positive + True\ Negative + False\ Positive + False\ Negative)}$$

These are the results of the accuracy of each model, tabulated below:

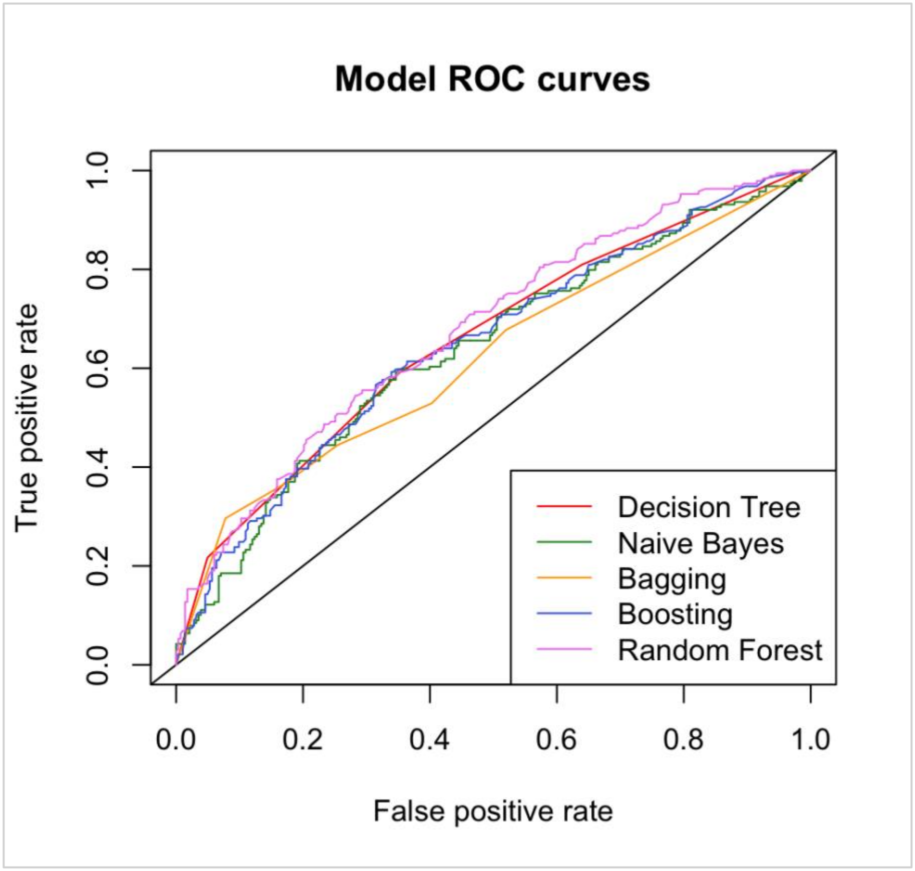


```
> Q5AccuracyTable
  Decision Tree Model Naive Bayes Model Bagging Model Boosting Model Random Forest Model
1               0.625         0.6334746         0.625      0.6377119           0.6440678
```

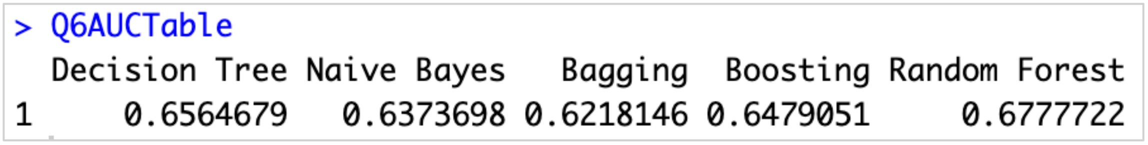
We can see that the random forest model is the most accurate model, having the highest accuracy of 0.6441. On the other hand, the decision tree and bagging model have the lowest accuracy of 0.625.

Question 6

In this question, I plotted the receiver operating characteristic (ROC) curve to visualize the performance for the 5 models by plotting its True Positive Rate (TPR) against its False Positive Rate (FPR). Below is the plot of the ROC curves of all the models.

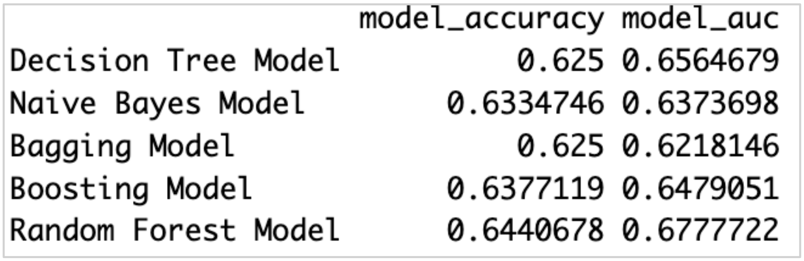


The table below shows the AUC of each model. By observing this table, we can see that the random forest model again outperforms the other models, having an area under the curve of 0.6777. On the other hand, the bagging model has the lowest AUC of 0.6218.


```
> Q6AUCTable
  Decision Tree Naive Bayes   Bagging  Boosting Random Forest
1     0.6564679   0.6373698 0.6218146 0.6479051     0.6777722
```

Question 7

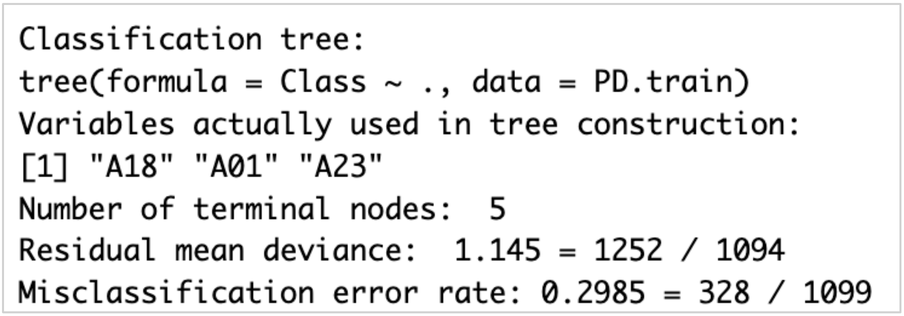
I created a table that records all the model accuracy and AUC values. The diagram below shows this table. It is apparent that the random forest model has both the highest accuracy and largest area under the curve, thus making it the single best classifier. The bagging model performed the worst, as it had both the lowest model accuracy score (0.625) and the smallest area under the curve of 0.6218.


```
                    model_accuracy model_auc
Decision Tree Model          0.625 0.6564679
Naive Bayes Model        0.6334746 0.6373698
Bagging Model                0.625 0.6218146
Boosting Model           0.6377119 0.6479051
Random Forest Model      0.6440678 0.6777722
```

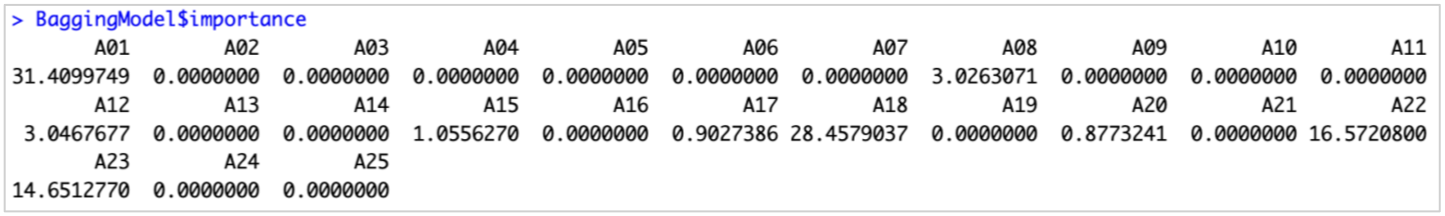
Question 8

In this question, we find out the most important predictors of each model.

Upon observing the summary of the decision tree model, the only predictors that are used in the construction of the tree are the attributes “A18”, “A01”, and “A23”. Therefore, other predictors are unnecessary to be included in the model and can be omitted.

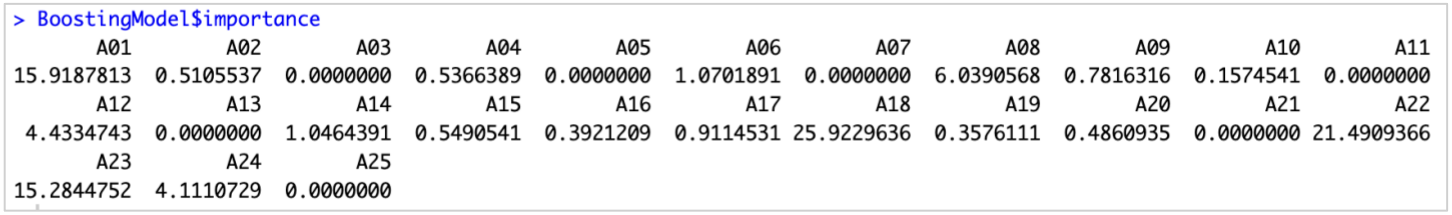

```
Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A18" "A01" "A23"
Number of terminal nodes:  5
Residual mean deviance:  1.145 = 1252 / 1094
Misclassification error rate: 0.2985 = 328 / 1099
```

For the bagging model, the 4 most important predictors are A01, A18, A22, A23. Out of the 25 predictors, the importance value of 16 of them are 0. This means that these 16 attributes can be omitted as they have no effect on the model.

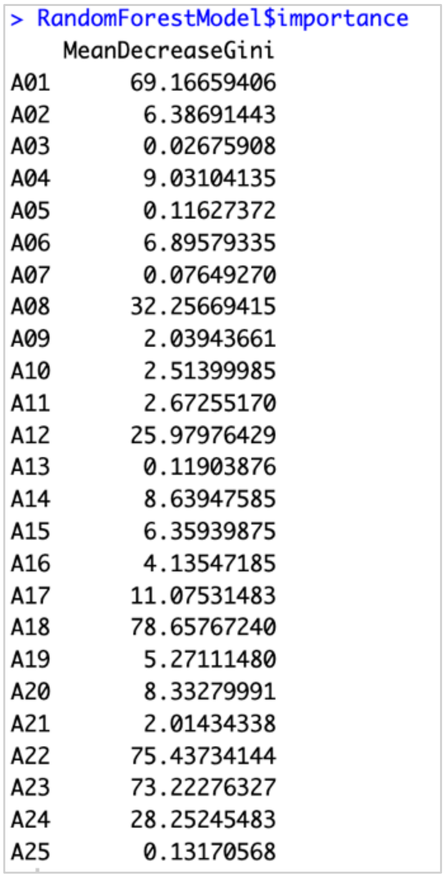


```
> BaggingModel$importance
       A01        A02        A03        A04        A05        A06        A07        A08        A09        A10        A11 
31.4099749  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  3.0263071  0.0000000  0.0000000  0.0000000 
       A12        A13        A14        A15        A16        A17        A18        A19        A20        A21        A22 
 3.0467677  0.0000000  0.0000000  1.0556270  0.0000000  0.9027386 28.4579037  0.0000000  0.8773241  0.0000000 16.5720800 
       A23        A24        A25 
14.6512770  0.0000000  0.0000000 
```

For the boosting model, the 4 most important predictors are the same as the bagging model (A01, A18, A22, A23). However, only 7 out of the 25 predictors have an importance value of 0 now.



```
> BoostingModel$importance
       A01        A02        A03        A04        A05        A06        A07        A08        A09        A10        A11 
15.9187813  0.5105537  0.0000000  0.5366389  0.0000000  1.0701891  0.0000000  6.0390568  0.7816316  0.1574541  0.0000000 
       A12        A13        A14        A15        A16        A17        A18        A19        A20        A21        A22 
 4.4334743  0.0000000  1.0464391  0.5490541  0.3921209  0.9114531 25.9229636  0.3576111  0.4860935  0.0000000 21.4909366 
       A23        A24        A25 
15.2844752  4.1110729  0.0000000 
```



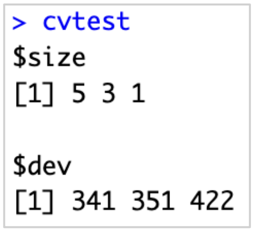
```
> RandomForestModel$importance
    MeanDecreaseGini
A01      69.16659406
A02       6.38691443
A03       0.02675908
A04       9.03104135
A05       0.11627372
A06       6.89579335
A07       0.07649270
A08      32.25669415
A09       2.03943661
A10       2.51399985
A11       2.67255170
A12      25.97976429
A13       0.11903876
A14       8.63947585
A15       6.35939875
A16       4.13547185
A17      11.07531483
A18      78.65767240
A19       5.27111480
A20       8.33279991
A21       2.01434338
A22      75.43734144
A23      73.22276327
A24      28.25245483
A25       0.13170568
```

The image on the left shows the importance values for the random forest model, the 4 most important predictors (in order) are: A18, A22, A23, A01. These predictors have a higher mean decrease Gini score which caused a larger decrease in Gini impurity when they were used for splits in the tree. This implies that they play a more significant role in distinguishing between classes.

In summary, the attributes A01, A18, A23, A22 seem to be consistently important across all the models, while the variables A13, A03, A05, A07, A25 consistently have low importance scores across all the models, which indicates that they can be likely omitted with little effect on performance.

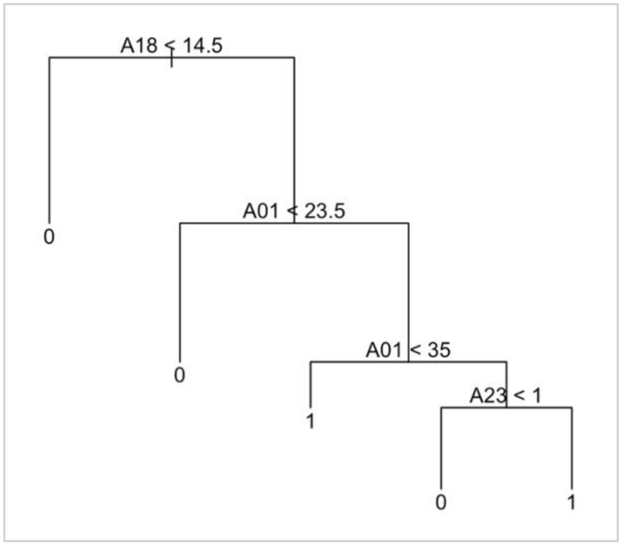
Question 9

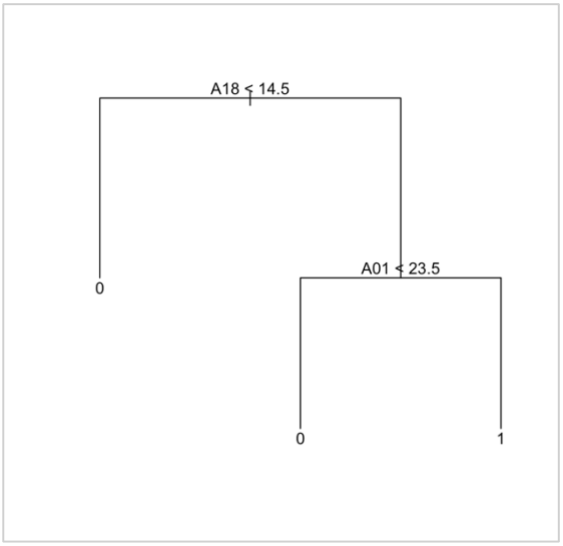
Based on one of the previous classifiers I made, I chose the decision tree for this question. This is because it is easy to visualize, and simple enough for a person to be able to classify whether a site is phishing or legitimate by hand. To create this simplified model, I applied pruning techniques, specifically using cross validation to find the optimal size based on the error classification rate. The lowest misclassification rate was for size = 5. Since my initial decision tree was already only using 3 attributes to fit the tree, this led to no difference, as the tree fitted was the same tree as Question 4. I also tried using the value of 4, but this again led to the exact same tree as before. Therefore, I decided to use the next best size, which is 3.



```
> cvtest
$size
[1] 5 3 1

$dev
[1] 341 351 422
```

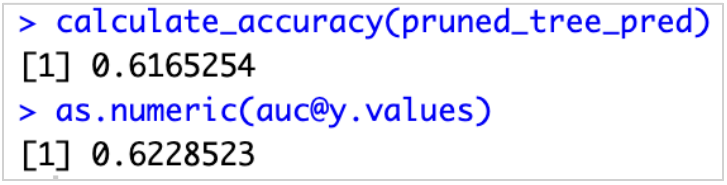
This resulted in an overly simple decision tree that only used 2 attributes. The left image shows the decision tree from Q4 while the right image shows the new pruned decision tree for this question.





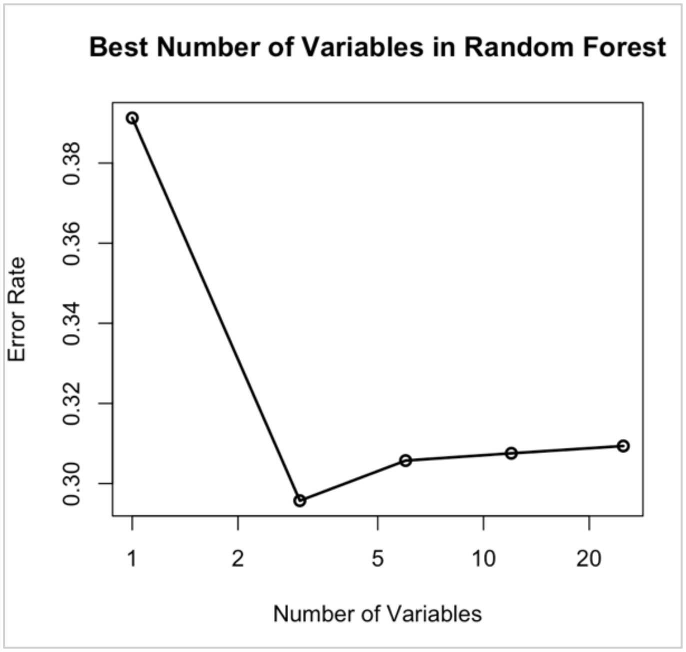
In comparison to the initial decision tree fitted, which only uses the attributes A01, A18, and A23 for splitting, this tree uses only 2 attributes, leaving out A23.

With a value of 3, too much of the tree is being cut away, which resulted in an overly simplistic model that underfits and performs poorly. These are the results of calculating the accuracy score and AUC of this simple decision tree model. As we can observe, it performs worse than the initial decision tree that we fit without pruning. In fact, this is the worst-performing model out of all. This suggests that this model is too simple and not accurate, implying that it is not useful for predictions.



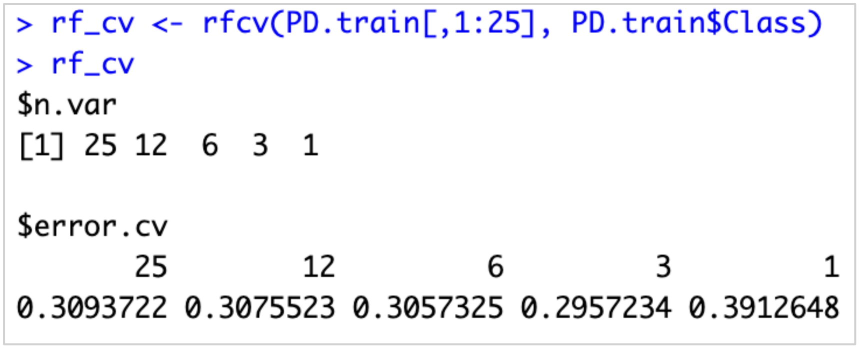
```
> calculate_accuracy(pruned_tree_pred)
[1] 0.6165254
> as.numeric(auc@y.values)
[1] 0.6228523
```

<u>Question 10</u>

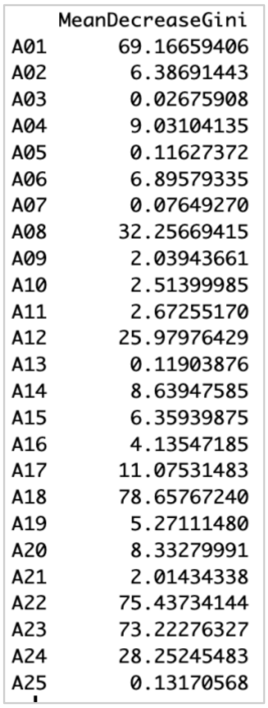


Among all the models fitted earlier, the random forest model performed the best, due to its highest accuracy and AUC. Therefore, I built the best classifier based on this model. I then performed cross validation to find out the optimal number of variables. As we can observe from the plot on the left, the optimal number of variables is chosen as the point right before the error rate starts to gradually increase, which is 3.

According to cross validation, the most optimal number of variables to include in the model is 3, as it has the lowest error of 0.2957234. However, this would mean only using 3 out of the 25 predictors that I have. Each predictor variable potentially carries valuable information about the patterns in the data, and removing too many of them might discard important details that could help the model distinguish between legitimate and phishing sites. Additionally, there is a risk of underfitting as the model might become too simple to capture the underlying complexities in the data, therefore I think using only 3 predictors might be insufficient for the model to learn effectively.



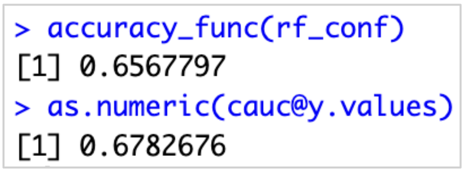
```
> rf_cv <- rfcv(PD.train[,1:25], PD.train$Class)
> rf_cv
$n.var
[1] 25 12  6  3  1

$error.cv
       25        12         6         3         1 
0.3093722 0.3075523 0.3057325 0.2957234 0.3912648 
```



```
    MeanDecreaseGini
A01      69.16659406
A02       6.38691443
A03       0.02675908
A04       9.03104135
A05       0.11627372
A06       6.89579335
A07       0.07649270
A08      32.25669415
A09       2.03943661
A10       2.51399985
A11       2.67255170
A12      25.97976429
A13       0.11903876
A14       8.63947585
A15       6.35939875
A16       4.13547185
A17      11.07531483
A18      78.65767240
A19       5.27111480
A20       8.33279991
A21       2.01434338
A22      75.43734144
A23      73.22276327
A24      28.25245483
A25       0.13170568
```

Therefore, I chose to exclude the attributes that have the lowest importance scores. Upon observing the importance scores of the attributes, I chose to exclude the attributes A03, A05, A07, A13, and A25.

Leaving out these predictors, I am left with an improved model as the accuracy and AUC values for this new random forest model are higher than the initial random forest model. It is apparent that this model outperforms all the other models in terms of accuracy as well as AUC.



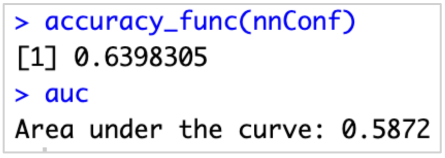
```
> accuracy_func(rf_conf)
[1] 0.6567797
> as.numeric(cauc@y.values)
[1] 0.6782676
```

Question 11

In the preprocessing stage, I performed data cleaning, normalization, and splitting the dataset into training and testing sets. The training set had 70% of the data (1099 rows), while the other 30% (472 rows) was allocated for the testing dataset. I also created indicators for categorical variables. The next step involved omitting rows containing NA values, as a neural network is unable to work with NA values. The step of data normalization was necessary as neural networks learn by adjusting their weights through a process called gradient descent. This basically calculates the gradients of the loss function with respect to the weights. If the features have vastly different scales, like if one feature ranges from 0 to 1, while another ranges from 0 to 10,000, the gradients become extremely imbalanced. As a result, features with larger scales will dominate the updates, which then hinders the model's ability to learn properly. Therefore, normalization helps features to have similar scales, making the gradients more balanced and potentially leading to a better model.

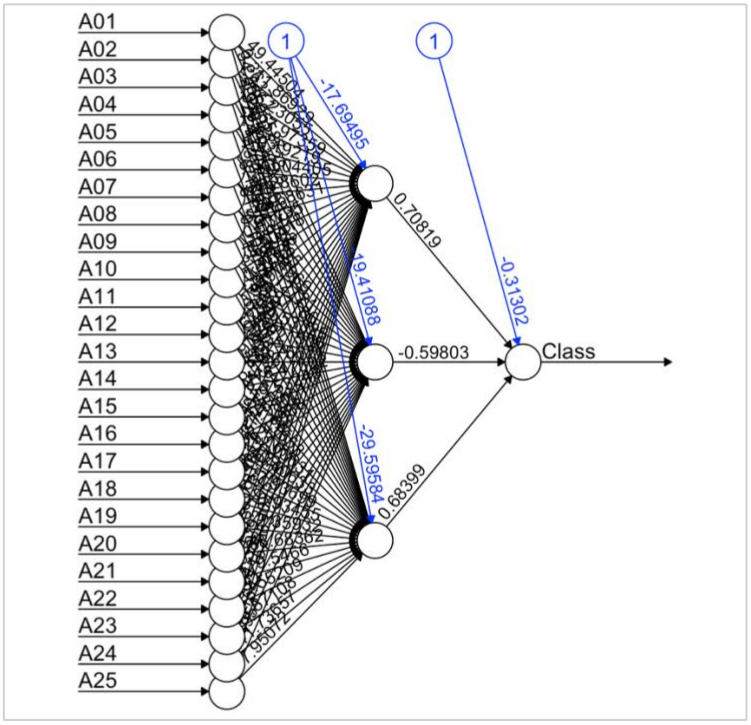
Attributes

The predictions are returned as numerical values because the target variable 'Class' was of numeric format. I set a threshold value of 0.5 to be the value which determines whether the model should predict a value to be 0 or 1. Initially, I fitted my ANN model to include all the predictors (A01 to A25). This was the accuracy and area under the curve of the model:

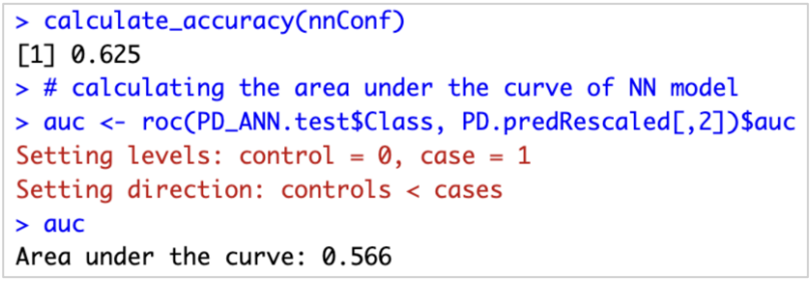


```
> accuracy_func(nnConf)
[1] 0.6398305
> auc
Area under the curve: 0.5872
```

The figure below shows the ANN, with all 25 predictors used.



When observing the previous models, the variables “A03”, “A05”, “A07”, “A13”, and “A25” seem to have the lowest importance. I then chose to exclude these variables from the ANN model, expecting to see an improvement in both the accuracy and AUC of the model. However, the effect of removing those predictors resulted in a lower accuracy and AUC:



```
> calculate_accuracy(nnConf)
[1] 0.625
> # calculating the area under the curve of NN model
> auc <- roc(PD_ANN.test$Class, PD.predRescaled[,2])$auc
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc
Area under the curve: 0.566
```

The explanation for this is perhaps those features, despite seeming unimportant based on their individual scores, might have contained complementary information to other features. Even if a single feature seems to have a low importance, it might work together with others to improve predictions. Removing such features can disrupt these interactions, causing overall performance to decrease. I therefore chose to keep all 25 predictors.

Evaluating the performance of the model

I chose to keep all 25 predictors in my ANN model. Therefore, the final accuracy of the model is 0.6398. This value is slightly lower than the random forest model, but higher than all the other models that I fitted earlier (Decision Tree, Naïve Bayes, Bagging, Boosting). This indicates that the ANN model can be powerful, but it may not always outperform the random forest model. A reason for this might be because ANNs can learn powerful internal representations of data through hidden layers. This allows ANNs to discover complex patterns, which simpler models are unable to do so. Additionally, ANNs have many hyperparameters (number of layers, neurons per layer, activation functions, learning rate etc.). Finding the optimal configuration is very important for good performance. This, however, is very challenging, and often requires significant experimentation or advanced optimization techniques. A poorly tuned ANN can underperform simpler models. Random forest models don’t have such parameters to take into consideration.

Question 12

For this question, I chose to fit a support vector machine as my classifier.

Link to package: https://cran.r-project.org/web/packages/e1071/index.html

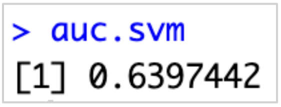
A support vector machine (SVM) is a method in supervised learning that is used in categorizing data. This is achieved by identifying the best line or hyperplane that creates the largest gap between different classes in a multidimensional space. SVMs are frequently employed for classification tasks, distinguishing between the 2 classes by finding the optimal hyperplane that maximizes the separation between the nearest data points from each class. The hyperplane can be a line (in 2D space) or a higher-dimensional plane, depending on the number of features in the data. Among the multiple possible hyperplanes, maximizing the margin ensures the SVM finds the most effective decision boundary, allowing it to generalize well and accurately predict new data points. The data points defining this maximal margin are known as support vectors.

Evaluating the performance of the model

Upon fitting the model, the model's accuracy score was calculated, and a value of 0.6525424 was obtained. This implies that it performs slightly worse than the improved random forest model, but higher than all the other models that I fitted earlier, meaning to say that this support vector machine model performed the 2nd best out of all the models in terms of accuracy.

```
> svm_accuracy
 Accuracy
0.6525424
```

As for the model's AUC, the value obtained was 0.6397442, which is somewhat like the AUC of all the other models.



```
> auc.svm
[1] 0.6397442
```

Therefore, it can be said that my SVM model performs the 5th highest in terms of AUC against all the other models. Since the dataset that we are working with is considered relatively small, the advantages of a more complex model like SVM might not be fully realized, which leads to similar accuracy and AUC scores with the simpler models fitted earlier.

Generative AI statement

I used Google Gemini to better understand the some of the concepts behind the models when doing this assignment.

```r
# setting working directory
setwd("/Users/junwei/Desktop/3152 A2")

install.packages("tree")
install.packages("rpart")
install.packages("adabag")
install.packages("e1071")
install.packages("randomForest")
install.packages("pROC")
install.packages("ROCR")
install.packages("neuralnet")
install.packages("caret")

library(neuralnet)
library(caret)
library(tree)
library(rpart)
library(adabag)
library(e1071)
library(randomForest)
library(pROC)
library(ROCR)

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32845650)
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

# Question 1

dim(PD)
# observing the structure of the data
str(PD)

# proportion of the variable "Class" that is non-phishing
sum(PD$Class == 0, na.rm = TRUE) / nrow(PD)

# proportion of the variable "Class" that is phishing
sum(PD$Class == 1, na.rm = TRUE) / nrow(PD)

summary(PD) # summary of data

# Question 2

# converting the class variable to type "factor"
PD$Class <- as.factor(PD$Class)

# number of NA values in the PD dataset
sum(is.na(PD))

# omitting rows containing NA values
PD_omit = na.omit(PD)

# Question 3
set.seed(32845650)
train.row = sample(1:nrow(PD_omit), 0.7 * nrow(PD_omit))

# creating the training and testing datasets
PD.train = PD_omit[train.row,]
PD.test = PD_omit[-train.row,]

# Question 4

# in this question, i fit the 5 models
```

```r
tree_fit <- tree(Class ~., data=PD.train)

nb_fit <- naiveBayes(Class ~. -Class, data=PD.train)

set.seed(32845650)
bag_fit <- bagging(Class ~., data=PD.train, mfinal=5)

set.seed(32845650)
boost_fit <- boosting(Class ~., data=PD.train, mfinal=10)

set.seed(32845650)
rf_fit <- randomForest(Class ~. -Class, data=PD.train)

# Question 5

# calculating the accuracy for the decision tree model
tree_pred <- predict(tree_fit, PD.test, type="class")
tree_conf <- confusionMatrix(data=tree_pred, reference=PD.test$Class)
tree_accuracy <- tree_conf$overall["Accuracy"]

# calculating the accuracy for naive bayes model
nb_pred <- predict(nb_fit, PD.test)
nb_conf <- confusionMatrix(data=nb_pred, reference=PD.test$Class)
nb_accuracy <- nb_conf$overall["Accuracy"]

# this function is used to calculate the accuracy of the bagging and boosting models
calculate_accuracy = function(conf_matrix) {
  True_Positive <- conf_matrix[2, 2]
  True_Negative <- conf_matrix[1, 1]
  False_Positive <- conf_matrix[1, 2]
  False_Negative <- conf_matrix[2, 1]

  acc <- (True_Positive + True_Negative) / (True_Positive + True_Negative +
False_Positive + False_Negative)
  return(acc)
}

# calculating the accuracy for the bagging model
bag_pred <- predict.bagging(bag_fit, PD.test)

# accuracy for the boosting model
boost_pred <- predict.boosting(boost_fit, PD.test)

# accuracy for random forest model
rf_pred <- predict(rf_fit, PD.test, type = "class")
rf_conf <- confusionMatrix(data=rf_pred, reference=PD.test$Class)
rf_accuracy <- rf_conf$overall["Accuracy"]

# list to keep track of the accuracy scores of every model
model_accuracy = list()
model_accuracy = append(model_accuracy, tree_accuracy)
model_accuracy = append(model_accuracy, nb_accuracy)
model_accuracy = append(model_accuracy, calculate_accuracy(bag_pred$confusion))
model_accuracy = append(model_accuracy, calculate_accuracy(boost_pred$confusion))
model_accuracy = append(model_accuracy, rf_accuracy)

# creating a dataframe that stores all the model accuracies
Q5AccuracyTable <- as.data.frame(t(model_accuracy))
colnames(Q5AccuracyTable) <- c("Decision Tree Model", "Naive Bayes Model", "Bagging
Model", "Boosting Model", "Random Forest Model")
Q5AccuracyTable


# Question 6

# calculating auc for decision tree
tree_predict <- predict(tree_fit, PD.test, type = "vector")
tree_predictions <- prediction(tree_predict[,2], PD.test$Class)
```

```
# calculating auc for naive bayes
nb_predict <- predict(nb_fit, PD.test, type='raw')
nb_predictions <- prediction(nb_predict[,2], PD.test$Class)

# calculating auc for bagging
bag_predictions <- prediction(bag_pred$prob[,2], PD.test$Class)

# calculating auc for boosting
boost_predictions <- prediction(boost_pred$prob[,2], PD.test$Class)

# calculating auc for random forest
rf_predict = predict(rf_fit, PD.test, type="prob")
rf_predictions <- prediction(rf_predict[,2], PD.test$Class)

# adding every model's AUC value to a list to keep track of them
model_auc = list()
model_auc = append(model_auc, as.numeric(performance(tree_predictions,
"auc")@y.values))
model_auc = append(model_auc, as.numeric(performance(nb_predictions,"auc")@y.values))
model_auc = append(model_auc, as.numeric(performance(bag_predictions, "auc")@y.values))
model_auc = append(model_auc, as.numeric(performance(boost_predictions,
"auc")@y.values))
model_auc = append(model_auc, as.numeric(performance(rf_predictions, "auc")@y.values))

# creating a dataframe that stores all the model AUC values
Q6AUCTable <- as.data.frame(t(model_auc))
colnames(Q6AUCTable) <- c("Decision Tree Model", "Naive Bayes Model", "Bagging Model",
"Boosting Model", "Random Forest Model")
Q6AUCTable

# plotting the ROC curves to visualize the performance of the models
plot(performance(tree_predictions, "tpr","fpr"), main="Comparing ROC curves of all the
models", col="red") # tree ROC curve
plot(performance(nb_predictions,"tpr","fpr"), add=TRUE, col="forestgreen") # naive
bayes ROC
plot(performance(bag_predictions,"tpr","fpr"), add = TRUE, col="orange") # bagging ROC
plot(performance(boost_predictions,"tpr","fpr"), add=TRUE, col ="royalblue") # boosting
ROC
plot(performance(rf_predictions, "tpr","fpr"), add=TRUE, col="violet") # random forest
ROC
abline(0,1)

legend("bottomright",legend=c("Decision Tree","Naive Bayes",
"Bagging","Boosting","Random Forest"),
       col=c("red","forestgreen","orange","royalblue","violet"),
       lty = 1)

# Question 7

# this dataframe stores performance measures of each model (accuracy and AUC)
pm_df = cbind(model_accuracy, model_auc)
pm_df = cbind(c("Decision Tree","Naive Bayes", "Bagging","Boosting","Random Forest"),
pm_df)
pm_df <- data.frame(pm_df[,-1], row.names = pm_df[,1])
pm_df

# Question 8

# identifying the important attributes for the models
summary(tree_fit)
# the naive bayes model does not identify most important attributes
bag_fit$importance
boost_fit$importance
rf_fit$importance

# Question 9
```

```r
#cross validation test to find which value to use to prune
cvtest = cv.tree(tree_fit, FUN = prune.misclass)
cvtest

pruned_treefit = prune.misclass(tree_fit, best = 3)
summary(pruned_treefit)
plot(pruned_treefit)
text(pruned_treefit, pretty = 0)

# calculating accuracy
pruned_tree.predict = predict(pruned_treefit, PD.test, type = "class")
pruned_tree_pred <- table(actual = PD.test$Class, predicted = pruned_tree.predict)
calculate_accuracy(pruned_tree_pred)


# calculating the area under the curve
pruned_tree_conf = predict(pruned_treefit, PD.test, type = "vector")
prunedtree_pred_obj <- prediction(pruned_tree_conf[,2], PD.test$Class)
auc = performance(prunedtree_pred_obj, "auc")
as.numeric(auc@y.values)

# Question 10

set.seed(32845650)

# performing cross validation on the random forest model
rf_cv <- rfcv(PD.train[,1:25], PD.train$Class)
rf_cv

# checking the importance of predictors to see which can be removed
rf_fit$importance

# removing the lowest 5 predictors with the lowest importance scores
PD.train_reduced = PD.train[,-c(3,5,7,13,25)]

set.seed(32845650)
# fitting the new random forest model
rf_fit <- randomForest(Class~., data = PD.train_reduced, ntree = 2500, max.depth = 150,
min.node.size = 2)
rf_pred <- predict(rf_fit, newdata = PD.test, type = "class")

# calculating the accuracy of this new random forest model
rf_conf <- table(observed = PD.test$Class, predicted=rf_pred)
calculate_accuracy(rf_conf)

# calculating the area under the curve
rf_cv_conf = predict(rf_fit, PD.test, type = "prob")
rf_pred_obj <- prediction(rf_cv_conf[,2], PD.test$Class)
auc = performance(rf_pred_obj, "auc")
as.numeric(auc@y.values)


# Question 11

# performing some preprocessing
PD_ANN = PD[complete.cases(PD),]
PD$Class = as.numeric(PD$Class) # changing Class to numeric data type

set.seed(32845650)
PD_ANN[, sapply(PD_ANN, is.numeric)] <- scale(PD_ANN[, sapply(PD_ANN, is.numeric)])

set.seed(32845650)
train.row = sample(1:nrow(PD_ANN), 0.7*nrow(PD_ANN))

# training and testing datasets for the NN model
PD_ANN.train = PD_ANN[train.row,]
PD_ANN.test = PD_ANN[-train.row,]
```

```
# indicators for the training dataset
Class = PD_ANN.train[,26]
PD_ANN.train = PD_ANN.train[,1:25]
dummy <- dummyVars(" ~ .", data = PD_ANN.train)
newPD_ANN.train <- data.frame(predict(dummy, newdata = PD_ANN.train))
PD_ANN.train = cbind(newPD_ANN.train, Class)

# indicators for the testing set
Class = PD_ANN.test[,26]
PD_ANN.test = PD_ANN.test[,1:25]
dummy <- dummyVars(" ~ .", data = PD_ANN.test)
newPD_ANN.test <- data.frame(predict(dummy, newdata = PD_ANN.test))
PD_ANN.test = cbind(newPD_ANN.test, Class)

# fitting the neural network model with all the predictors (A01 to A25)
set.seed(32845650)
PD.nn = neuralnet(Class ~ A01 + A02 + A03 + A04 + A05 + A06 + A07 + A08 + A09+ A10 +
                          A11 + A12 + A13 + A14 + A15 + A16 + A17 + A18 + A19 + A20 +
                          A21 + A22 + A23 + A24 + A25,
                          data=PD_ANN.train, hidden=3, linear.output=TRUE)

# plotting the neural network
plot(PD.nn, rep="best")

# setting a threshold value of 0.5
PD.pred = compute(PD.nn, PD_ANN.test[,1:25])
PD.prob <- PD.pred$net.result
pred <- ifelse(PD.prob>0.5, 1, 0)

# calculating the accuracy
nnConf <- table(observed = PD_ANN.test$Class, predicted=pred[,2])
nnConf
calculate_accuracy(nnConf)

# calculating the AUC
auc <- roc(PD_ANN.test$Class, pred[,2])$auc
auc

# Question 12

# fitting a support vector machine
set.seed(32845650)
svm_fit <- svm(Class ~., data = PD.train, kernel="linear", probability = TRUE)

svm_pred <- predict(svm_fit, newdata = PD.test)
svm_conf <- confusionMatrix(data=svm_pred, reference=PD.test$Class)

# calculating the accuracy
svm_accuracy <- svm_conf$overall["Accuracy"]
svm_accuracy

# calculating the AUC
svm_conf <- predict(svm_fit, PD.test, probability = TRUE)
svm_conf <- attr(svm_conf, "probabilities")
svm_pred <- ROCR::prediction(svm_conf[, 1], PD.test$Class)
auc.svm <- performance(svm_pred, "auc")@y.values[[1]]
auc.svm
```