FIT3152 - Data analytics

Assignment 2

Name: Rhyme Bulbul

Student ID: 31865224

AI Statement: Generative AI was not used in this assignment.

We start by utilizing the R skeleton code to import out phishing dataset and set a seed using my student number so the unique data is replicable. Next, we take a random sample of 2000 rows of data, and import libraries further code will rely on.

```r
rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(31865224)
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),]

#install.packages("dplyr")
library(dplyr)
#install.packages("caret")
library(caret)
#install.packages("tree")
library(tree)
#install.packages("e1071")
library(e1071)
#install.packages("adabag", dependencies = TRUE)
library(adabag)
#install.packages("randomForest")
library(randomForest)
#install.packages("ROCR")
library(ROCR)
#install.packages("lightgbm")
library(lightgbm)
#install.packages("kernlab")
library(kernlab)
```

Question 1

There are 729 phishing sites and 1271 legitimate sites in this sample. This brings the proportion of phishing sites to legitimate sites is 0.57, which indicates that a bit over half of the sites included in this sample of the data set are phishing sites. The data is relatively clean with no missing or empty values for this attribute.

```r
phishing <- nrow(subset(PD, subset = Class == 1))
legitimate <- nrow(subset(PD, subset = Class == 0))
phishing
```

```
## [1] 729
```

```r
legitimate
```

```
## [1] 1271
```

```
phishing / legitimate
```

```
## [1] 0.5735641
```

```
nrow(PD[is.na(PD$Class), ])
```

```
## [1] 0
```

Next, we will run a summary of the data set to depict predictor descriptions such mean and standard deviation, which can be found in the appendix. Due to the nature of this data set, we are only familiar with the representation of the `Class` column, as all other columns are numbered from `A01` to `A25`.

It is worth noting a large number of attributes such as `A02`, `A05`, `A06` and so on have very low means, as well as quartiles of 0.00, which indicates they have a very low numerical value as well. In contrast, `A12` has comparatively high mean and quartiles, indicating a higher numerical value in the dataset. In addition, all attributes except `A01` have a small number of missing values.

```
summary(PD)
```

Next, analysing the numerical attributes standard deviation, we observe several columns including `A03`, `A04`, `A06`, `A07` and so on with an exquisitely low standard deviation, indicationg that those columns have been consistent in this sample of the dataset. On the flip side, it worth noting that several columns such as `A12`, `A13` have particularly higher standard deviation, indicating disparity among the values in those columns.

```
apply(PD, 2, sd, na.rm = TRUE)
```

```
##          A01          A02          A03          A04          A05          A06 
## 1.399917e+01 1.425657e+00 3.173403e-02 5.556241e-01 3.415951e+00 3.276963e-01 
##          A07          A08          A09          A10          A11          A12 
## 5.942851e-02 2.202868e-01 1.634812e-01 1.970024e-01 3.695052e+00 1.421402e+02 
##          A13          A14          A15          A16          A17          A18 
## 1.011359e+01 3.538830e-01 3.364326e-01 2.159532e-01 6.278786e-01 1.086829e+02 
##          A19          A20          A21          A22          A23          A24 
## 3.184652e-01 4.196079e-01 1.674980e-01 1.111457e-02 6.562351e+01 2.524273e-01 
##          A25        Class 
## 7.911251e-03 4.814100e-01
```

Considering the missing values and standard deviation, we should not be required to omit any attributes as the data set looks normal.

Question 2

Given none of the columns have any priority we can omit them based off, all columns are included. However, rows with missing values have to be dropped to make the data set suitable to have a model fitted to it.

As such, we take a dataframe with the clean data, and convert the `Class` column into a factor, as it consists of a numerical data type and we are looking to build a classification tree rather than a regression tree. This now gives 1573 rows after dropping the ones with missing values.

```
clean_pd <- PD[complete.cases(PD), ]
clean_pd$Class <- as.factor(clean_pd$Class)

nrow(clean_pd)
```

```
## [1] 1573
```

```
#table(clean_pd$Class)
```

Question 3

Adapting the given skeleton R code to divide the data into a 70% training and 30% test set, we get as follows

```r
train_row <- sample(1:nrow(clean_pd), 0.7 * nrow(clean_pd))
pd_train <- clean_pd[train_row, ]
pd_test <- clean_pd[-train_row, ]
```

Question 4

We implement each of Decision Tree, Naive Baiyes, Bagging, Boosting and Random Forest below with R functions using their default settings.

```r
# Decision tree
pd_tree <- tree(Class ~ ., data = pd_train)

# Naive bayes
pd_bayes <- naiveBayes(Class ~ ., data = pd_train)

# Bagging
pd_bag <- bagging(Class ~ ., data = pd_train)

# Boosting
pd_boost <- boosting(Class ~ ., data = pd_train)

# Random forest
pd_forest <- randomForest(Class ~ ., data = pd_train)
```

Question 5

We start classifying the data into either 1 for phishing sites, or 0 for legitimate sites using the predict functions for each respective models

```r
# decision tree
pd_tree_predict <- predict(pd_tree, pd_test, type = "class")

# naive bayes
pd_bayes_predict <- predict(pd_bayes, pd_test)

# bagging
pd_bag_predict <- predict.bagging(pd_bag, pd_test)

# boosting
pd_boost_predict <- predict.boosting(pd_boost, pd_test)

# random forest
pd_forest_predict <- predict(pd_forest, pd_test)
```

Next, we develop a confusion matrix for each respective model, along with the accuracy of each.

```r
get_accuracy <- function(confusion_matrix) {
    return(sum(diag(confusion_matrix)) / sum(confusion_matrix))
}

# decision tree
pd_tree_confusion_matrix <- table("Predicted Class" = pd_tree_predict,
                      "Actual Class" = pd_test$Class)
pd_tree_accuracy <- get_accuracy(pd_tree_confusion_matrix)
cat("Decision tree (accuracy:", pd_tree_accuracy, ")\n")
```

```
## Decision tree (accuracy: 0.7224576 )
```

```r
pd_tree_confusion_matrix
```

```
##                 Actual Class
## Predicted Class   0   1
##               0 251  86
##               1  45  90
```

```r
# naive bayes
pd_bayes_confusion_matrix <- table("Predicted Class" = pd_bayes_predict,
                       "Actual Class" = pd_test$Class)
pd_bayes_accuracy <- get_accuracy(pd_bayes_confusion_matrix)
cat("Naive bayes classifier (accuracy:", pd_bayes_accuracy, ")\n")
```

```
## Naive bayes classifier (accuracy: 0.3961864 )
```

```r
pd_bayes_confusion_matrix
```

```
##                 Actual Class
## Predicted Class   0   1
##               0  14   3
##               1 282 173
```

```r
# bagging
pd_bag_accuracy <- get_accuracy(pd_bag_predict$confusion)
cat("Bagging (accuracy:", pd_bag_accuracy, ")\n")
```

```
## Bagging (accuracy: 0.7224576 )
```

```r
pd_bag_predict$confusion
```

```
##                 Observed Class
## Predicted Class   0   1
##               0 258  93
##               1  38  83
```

```r
# boosting
pd_boost_accuracy <- get_accuracy(pd_boost_predict$confusion)
cat("Boosting (accuracy:", pd_boost_accuracy, ")\n")
```

```
## Boosting (accuracy: 0.6991525 )
```

```r
pd_boost_predict$confusion
```

```
##                 Observed Class
## Predicted Class   0   1
##               0 244  90
##               1  52  86
```

```r
# random forest
pd_forest_confusion_matrix <- table("Predicted Class" = pd_forest_predict,
                        "Actual Class" = pd_test$Class)
pd_forest_accuracy <- get_accuracy(pd_forest_confusion_matrix)
cat("Random forest (accuracy:", pd_forest_accuracy, ")\n")
```

```
## Random forest (accuracy: 0.7330508 )
```

```
pd_forest_confusion_matrix
```

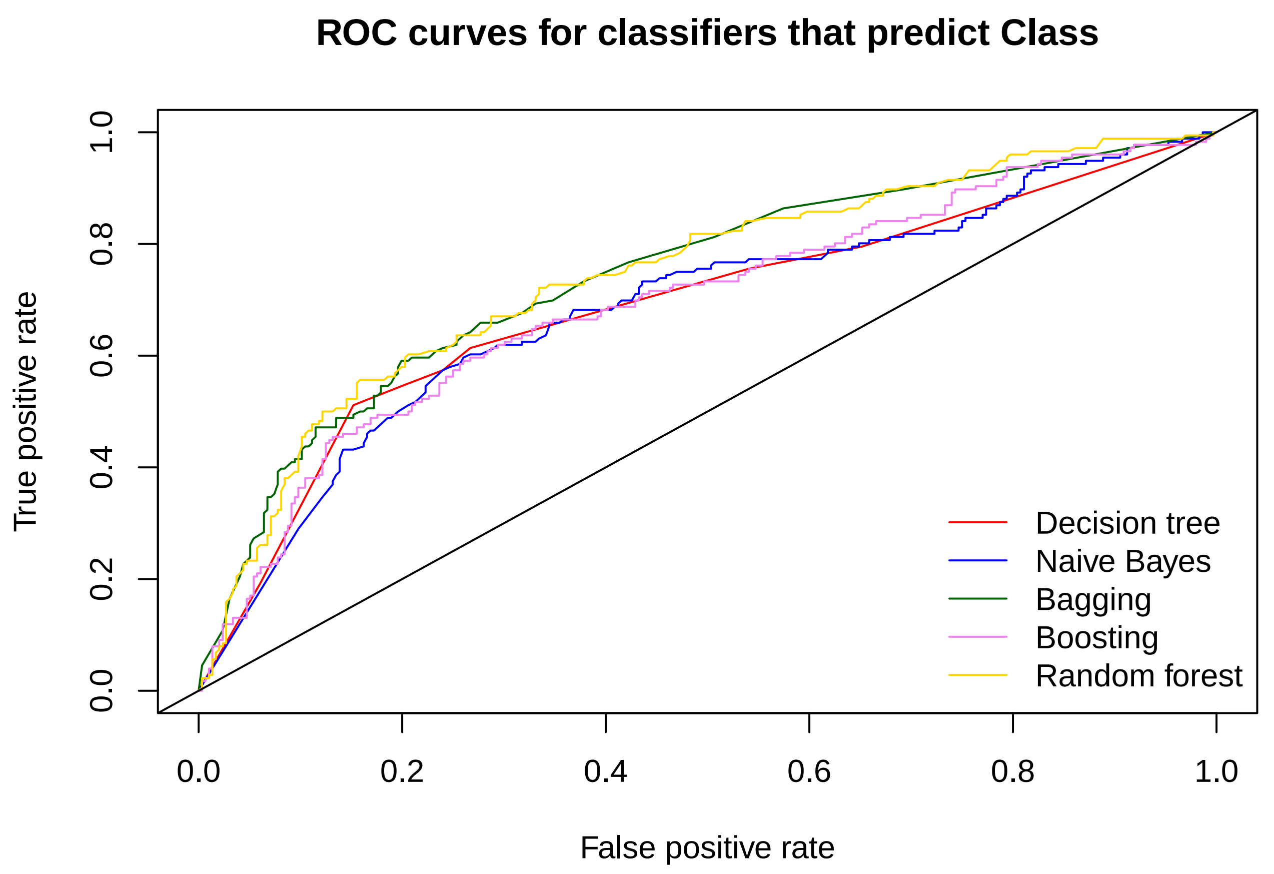
```
##                  Actual Class
## Predicted Class   0   1
##                 0 260  90
##                 1  36  86
```

Overall, we find most models have accuracy of 0.7 or higher, while Naive Bayes classifier has the least of 0.39 which is on par with random guesses.

Question 6

We calculate the confidence of predicting `phishing` for each case by utilizing the parameter `type` from the function predict already used in each model. Next, we are able to construct a ROC curve for each curve using the prediction and performance functions to plot them on the same axis using a different colour for each classifier.

```r
# decision tree
pd_tree_predict_prob <- predict(pd_tree, pd_test, type = "vector")
pd_tree_pred <- prediction(pd_tree_predict_prob[, 2], pd_test$Class)
pd_tree_perf <- performance(pd_tree_pred, "tpr", "fpr")
plot(pd_tree_perf, col = "red")

# naive bayes
pd_bayes_predict_prob <- predict(pd_bayes, pd_test, type = "raw")
pd_bayes_pred <- prediction(pd_bayes_predict_prob[, 2], pd_test$Class)
pd_bayes_perf <- performance(pd_bayes_pred, "tpr", "fpr")
plot(pd_bayes_perf, col = "blue", add = TRUE)

# bagging
pd_bag_predict_prob <- predict.bagging(pd_bag, pd_test, type = "prob")
pd_bag_pred <- prediction(pd_bag_predict_prob$prob[, 2], pd_test$Class)
pd_bag_perf <- performance(pd_bag_pred, "tpr", "fpr")
plot(pd_bag_perf, col = "darkgreen", add = TRUE)

# boosting
pd_boost_predict_prob <- predict.boosting(pd_boost, pd_test, type = "prob")
pd_boost_pred <- prediction(pd_boost_predict_prob$prob[, 2], pd_test$Class)
pd_boost_perf <- performance(pd_boost_pred, "tpr", "fpr")
plot(pd_boost_perf, col = "violet", add = TRUE)

# random forest
pd_forest_predict_prob <- predict(pd_forest, pd_test, type = "prob")
pd_forest_pred <- prediction(pd_forest_predict_prob[, 2], pd_test$Class)
pd_forest_perf <- performance(pd_forest_pred, "tpr", "fpr")
plot(pd_forest_perf, col = "gold", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest"),
       col = c("red", "blue", "darkgreen", "violet", "gold"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict Class")
```



Next, we calculate the `AUC` for each classifier utilizing the performance function with `auc` as the parameter. `AUC` is computed to be above .6 for all models, as well as above .7 for Bagging and Random Forest.

```
pd_tree_auc <- performance(pd_tree_pred, "auc")@y.values[[1]]
pd_bayes_auc <- performance(pd_bayes_pred, "auc")@y.values[[1]]
pd_bag_auc <- performance(pd_bag_pred, "auc")@y.values[[1]]
pd_boost_auc <- performance(pd_boost_pred, "auc")@y.values[[1]]
pd_forest_auc <- performance(pd_forest_pred, "auc")@y.values[[1]]

cat("Decision tree AUC", pd_tree_auc, "\n")
```

```
## Decision tree AUC 0.6879415
```

```
cat("Naive Bayes classifier AUC", pd_bayes_auc, "\n")
```

```
## Naive Bayes classifier AUC 0.6842464
```

```
cat("Bagging AUC", pd_bag_auc, "\n")
```

```
## Bagging AUC 0.7431089
```

```
cat("Boosting tree AUC", pd_boost_auc, "\n")
```

```
## Boosting tree AUC 0.69418
```

```
cat("Random forest AUC", pd_forest_auc, "\n")
```

```
## Random forest AUC 0.7450956
```

Question 7

Creating a table comparing the AUC and accuracy of each model throughout questions 5 and 6, Random Forest wins the highest values, with both well above 0.7. However, it is not well ahead of Bagging, also both above 0.7, as well as Decision Tree and Boosting which are both above 0.6. Only Naives Bayes Classifier performs poorly as observed previously. However classifiers should be evaluated with caution at this stage, lest the model be over fit. As such, there is no single best classifier, only the right classifier for the right data set.

```
pd_accuracy_auc <- data.frame(
  model = c("Decision tree", "Naive Bayes classifier", "Bagging", "Boosting",
            "Random forest"),
  accuracy = c(pd_tree_accuracy, pd_bayes_accuracy, pd_bag_accuracy, pd_boost_accuracy,
               pd_forest_accuracy),
  auc = c(pd_tree_auc, pd_bayes_auc, pd_bag_auc, pd_boost_auc, pd_forest_auc)
)

pd_accuracy_auc
```

```
##                    model  accuracy       auc
## 1          Decision tree 0.7224576 0.6879415
## 2 Naive Bayes classifier 0.3961864 0.6842464
## 3                Bagging 0.7224576 0.7431089
## 4               Boosting 0.6991525 0.6941800
## 5          Random forest 0.7330508 0.7450956
```

Question 8

We start by peeking through the summary of the decision tree for phishing data, with `A14 22 01 18 and 23` being the most important predictors of whether a website is a phishing one or is legitimate. Naives Bayes Classifiers, however assume each predictor to be independent and thus equal or no importance each.

```
summary(pd_tree)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = pd_train)
## Variables actually used in tree construction:
## [1] "A14" "A22" "A01" "A18" "A23"
## Number of terminal nodes:  8
## Residual mean deviance:  1.025 = 1121 / 1093
## Misclassification error rate: 0.2243 = 247 / 1101
```

Sorting the variables in order of ascending importance for the Bagging model, we can see that `A01 22 18 23 14` are significantly higher in importance than the rest, with `A03 05 07 10 11 13 21 and 25` having no importance at all.

```
sort(pd_bag$importance)
```

```
##         A03         A05         A07         A10         A11         A13
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##         A21         A25         A15         A09         A16         A06
##  0.00000000  0.00000000  0.08421883  0.08507971  0.09154959  0.10775908
##         A20         A17         A02         A04         A19         A12
##  0.15789867  0.15842305  0.19297981  0.22158064  0.25151788  1.08418424
##         A08         A24         A14         A23         A18         A22
##  2.61147890  2.63016704 14.71401180 15.65736761 17.56398502 20.10531997
##         A01
## 24.28247816
```

Sorting again in order of importance for the Boosting model, we get similar results, with `A22 18 23 01 08 24 and 12` being significantly higher than others, again with `A03 07 13 and 25` with no importance at all.

```
sort(pd_boost$importance)
```

```
##         A03         A07         A13         A25         A21         A05
##  0.00000000  0.00000000  0.00000000  0.00000000  0.03746415  0.08331263
##         A11         A10         A20         A16         A09         A06
##  0.23208759  0.30281897  0.52569912  0.59823633  0.61034459  0.83480589
##         A02         A04         A19         A17         A15         A14
##  1.04382151  1.13492515  1.18835954  1.40413021  1.81567691  2.75081887
##         A12         A24         A08         A01         A23         A18
##  5.46336669  6.79557212  9.02186109  9.59616805 14.31621058 16.15025773
##         A22
## 26.09406228
```

Finally, we also sort in order of importance for the Random Forest Model, this time however `A01 08 12 14 18 22 23 24` have significantly high values higher than 20, compared to the rest with much lower values.

```
pd_forest$importance
```

```
##     MeanDecreaseGini
## A01     68.974150020
## A02      6.494051400
## A03      0.009230137
## A04      9.473005188
## A05      0.235441730
## A06      6.018464222
## A07      0.080347428
## A08     28.651833759
## A09      4.120971735
## A10      1.703212390
## A11      1.485091096
## A12     25.792098813
## A13      0.072644779
## A14     22.473853389
## A15      5.562874829
## A16      4.198210111
## A17     10.709656481
## A18     67.612614914
## A19      5.702748868
## A20      8.706318119
## A21      1.385302732
## A22     77.490128054
## A23     66.727041715
## A24     26.956009476
## A25      0.100086973
```

To conclude, we find `A01 18 22 23` to have consistently high importance throughout all the models in predicting `Class`, while, `A03 07 13 25` have the least importance. Given the models in question have sufficiently high accuracy, these 4 least important variables could potentially be omitted from the data with little to no effect on performance. However, it worth noting that with trees based classifiers being unstable, minor differences in the input sampled can result in the tree generated varying significantly. As such, it not possible to predict the performance gain by omitting such variables, and is recommended to analyse the results after testing and training and make decisions based on that information.

Question 9

Starting off by developing a basic classifier based on the decision tree generated in Question 4, that can be used to manually classify objects. The strategy is to trim the tree to the ideal size, resulting in a more straightforward decision tree that is readily employed for categorization.

Using `cv.tree()`, we perform cross-validation on the original decision tree to determine what size to prune the tree to. The original tree was trained on all qualities, therefore all of its attributes were kept. Even if the tree may be over fit, it is still the most complete tree that needs to be pruned. Put another way, the original tree will undergo **post-pruning** in order to produce the simple tree.

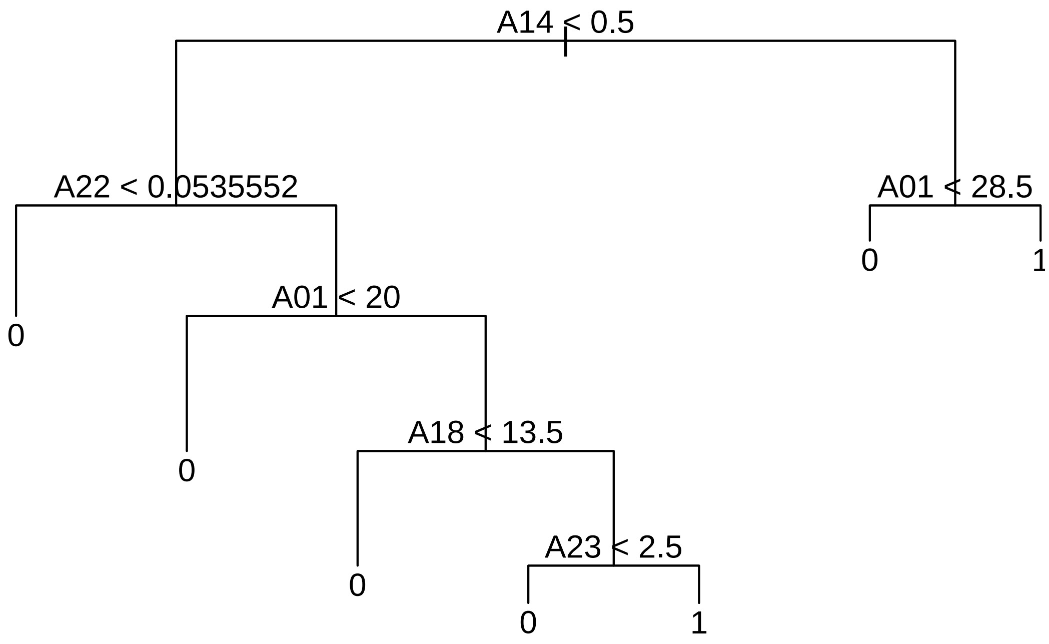
```r
pd_tree_cv <- cv.tree(pd_tree)
pd_tree_cv
```

```
## $size
## [1] 8 7 6 5 4 2 1
##
## $dev
## [1] 1204.974 1215.596 1215.596 1220.983 1321.215 1329.058 1423.100
##
## $k
## [1]     -Inf 16.50566 17.39028 18.47400 56.67066 60.71972 81.40772
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The output indicates that a tree size of `8` produces the least deviation. On the other hand, a tree with only one terminal node is illogical. The next best tree size is `7`, which is obviously an overly simplistic and under fitted tree. Trees of these sizes are anticipated to perform badly on unknown data, despite wielding the lowest deviance. The low deviations are most likely the result of over fitting on the training set and cross-validation identifying simple trees that function effectively. Therefore, choosing the best tree size cannot be based just on the lowest deviation.

`6` is the next-smallest deviant tree size. This tree size was selected because it produces a reasonably low deviation and is a suitable size—it is neither too big nor too little. To prune the tree, we use `prune.tree()`.

```r
pd_tree_pruned <- prune.tree(pd_tree, best = 7)
plot(pd_tree_pruned)
text(pd_tree_pruned, pretty = 0)
```



The decision tree that results is clear-cut and straightforward to understand. Compared to the original decision tree, which had 22 terminal nodes and split on nine qualities, this decision tree has just seven terminal nodes and splits on five attributes.

To assess how well this pruned decision tree performs on the test data, generate a confusion matrix, and report its accuracy, similar code from Question 5 is utilized.

```
pd_tree_pruned_accuracy
```

```
## [1] 0.7224576
```

```
pd_tree_pruned_confusion_matrix
```

```
##                Actual Class
## Predicted Class   0   1
##               0 251  86
##               1  45  90
```

This decision tree's pruned accuracy, of 0.7224576, is exactly the same as the original tree's accuracy of 0.7224576.

Next, the plot of ROC curves is updated, and the AUC value is calculated using code similar to that from Question 6 (see Appendix).

```
pd_accuracy_auc <- rbind(pd_accuracy_auc, data.frame(model = "Pruned decision tree",
                                                   accuracy = pd_tree_pruned_accuracy,
                                                   auc = pd_tree_pruned_auc))
pd_accuracy_auc
```

```
##                      model  accuracy       auc
## 1           Decision tree 0.7224576 0.6879415
## 2 Naive Bayes classifier 0.3961864 0.6842464
## 3                 Bagging 0.7224576 0.7431089
## 4                Boosting 0.6991525 0.6941800
## 5           Random forest 0.7330508 0.7450956
## 6   Pruned decision tree 0.7224576 0.6875576
```

Moreover, the AUC value from 0.6879415 is unchanged for the pruned tree.

Question 10

I have experimented with scaling features, removing predictors, modifying parameters, and cross-validation on the classifiers developed in Question 4 in an effort to develop the best tree-based classifier. It is unfortunate that none of the final classifiers could outperform the default-parameter boosting and bagging classifiers included in Question 4. Rather, a gradient-boosted trees model is the most powerful tree-based classifier that can be developed.

Using an ensemble approach called gradient boosting, decision trees are repeatedly trained to reduce the mistakes produced by the group of previously trained trees. Using the lightgbm package, it is possible to create a gradient-boosted trees model.

The only input features and targets that gradient boosting takes are numerical. Therefore before testing and training, the factor variables are converted into numerical values.

```
pd_train_lgbm <- pd_train
pd_test_lgbm <- pd_test

for (col in colnames(pd_train_lgbm)[1:21]) {
  if (is.factor(pd_train_lgbm[, col])) {
    pd_train_lgbm[, col] <- as.numeric(pd_train_lgbm[, col])
    pd_test_lgbm[, col] <- as.numeric(pd_test_lgbm[, col])
  }
}

pd_train_lgbm$Class <- as.numeric(pd_train_lgbm$Class) - 1
pd_test_lgbm$Class <- as.numeric(pd_test_lgbm$Class) - 1
```

We use lgb.train() to fit the model, which only accepts input data of type lgb.Dataset. As such we transform our phishing dataset as matrices into lgb.Dataset type.

```
pd_train_lgbmd <- lgb.Dataset(data = as.matrix(pd_train_lgbm[1:21]),
                              label = as.matrix(pd_train_lgbm$Class))
```

The best-performing gradient-boosted trees model was obtained using the following parameters, which are initially allocated to a variable that will be supplied to ‘lgb.train()}. After trying several settings repeatedly to get the best-performing model, the parameter values were chosen.

- `learning_rate = 0.05` sets the shrinkage rate (which regulates how much each tree in the ensemble contributes to the final prediction) to 0.05. - `objective = "binary"` sets `lgb.train()` to train a binary classification model. The parameters `feature_fraction = 0.7` and `bagging_fraction = 0.8` instruct `lgb.train()` to choose 70% of features and 80% of data, respectively, at random for bagging; `bagging_freq = 10` instructs `lgb.train()` to carry out bagging every 10 iterations.

```
params <- list(objective = "binary", learning_rate = 0.05, feature_fraction = 0.7,
               bagging_fraction = 0.8, bagging_freq = 10)
```

Next, 100 rounds of lgb.train() are executed with these settings.

```
pd_lgbm <- lgb.train(params = params, data = pd_train_lgbmd, nrounds = 100)
```

Finally, we develop a confusion matrix and measure the accuracy of the prediction

```
pd_lgbm_accuracy
```

```
## [1] 0.7394068
```

```
pd_lgbm_cm
```

```
##                  Actual Class
## Predicted Class   0   1
##               0 257  84
##               1  39  92
```

The gradient-boosted trees model exceeds all of our previous tree-based classifiers with an accuracy of 0.7394068. The ROC curve for this model is added to the display, just like in earlier questions, and the AUC value is calculated as referred to in the Appendix.

```
pd_accuracy_auc
```

```
##                     model  accuracy       auc
## 1           Decision tree 0.7224576 0.6879415
## 2 Naive Bayes classifier 0.3961864 0.6842464
## 3                 Bagging 0.7224576 0.7431089
## 4                Boosting 0.6991525 0.6941800
## 5           Random forest 0.7330508 0.7450956
## 6    Pruned decision tree 0.7224576 0.6875576
## 7 Gradient-boosted trees 0.7394068 0.7395961
```

The AUC score, 0.7395961, is likewise greater than the average of all previous tree-based classifiers, being exceed only by Bagging. The ROC curve indicates that at medium-high thresholds, this model outperforms boosting and bagging, but it performs better at all other thresholds. These metrics demonstrate that, when it comes to Class predictions, this classifier is superior to the others.

Given that gradient boosting is a variation of boosting, the most potent classifier in Question 4, it was selected when no other tree-based classifiers could perform any better. An improvement in performance is anticipated by implementing a potentially superior variant of it that is good at capturing relationships between predictors and has a lower tendency to over fit. Additionally, gradient boosting is known to perform well in applications involving binary classification in general. Since incorporating all of them yielded the best models for bagging and boosting, all of the original attributes of the Phishing data were kept. Naturally, the target variable and factor predictors must first be converted into numerical values.

Question 11

To implement an artificial neural network (ANN) classifier, the `neuralnet` package is used.

```
#rm(list = ls())
#install.packages("neuralnet")
library(neuralnet)
```

In order to build an Artificial Neural Network, we need to preprocess the data, dropping any factor attributes such as strings, which our data set has none. Next, we also drop the columns with the same values for all rows, such as 0 in columns `A3`, `A5` and so on, as they will have no influence on the neural network at this stage, given we will not be able to scale them. Finally, we normalize the data by scaling with the inbuilt R function as we have only numerica data.

```
pd_train_nn <- na.omit(pd_train[, c(1, 2, 4, 6, 9, 12, 14, 15, 16, 17, 18,
                                    19, 21, 22, 23, 24, 26)])
pd_test_nn <- na.omit(pd_test[, c(1, 2, 4, 6, 9, 12, 14, 15, 16, 17, 18,
                                  19, 21, 22, 23, 24, 26)])

pd_train_nn$Class <- as.numeric(pd_train_nn$Class) - 1
pd_test_nn$Class <- as.numeric(pd_test_nn$Class) - 1

pd_train_nn[1:17] <- scale(pd_train_nn[1:17])
pd_test_nn[1:17] <- scale(pd_test_nn[1:17])
```

Next, we build an Artificial Neural Net using the preprocessed data, setting 3 neurones, so that there is only 1 hidden layer. We can predict making use of the test data, and in turn create a confusion matrix to determine the accuracy of our ANN.

```
pd_nn_accuracy
```

```
## [1] 0.6271186
```

```
pd_nn_cm
```

```
##                Actual Class
## Predicted Class -0.770282325049726 1.29547481940181
##               0                296              176
```

At 0.6271186, Artificial Neural Networks only seem to do particularly better than Naive Bayes classifiers, which isn't a very high bar, yet ANN for our data set still sits up there with the other highly accurate classifiers.

Next, let's build ROC curves for all our classifiers so far, and compute their AUC.

```
pd_accuracy_auc
```

```
##                        model  accuracy       auc
## 1              Decision tree 0.7224576 0.6879415
## 2     Naive Bayes classifier 0.3961864 0.6842464
## 3                    Bagging 0.7224576 0.7431089
## 4                   Boosting 0.6991525 0.6941800
## 5              Random forest 0.7330508 0.7450956
## 6        Pruned decision tree 0.7224576 0.6875576
## 7     Gradient-boosted trees 0.7394068 0.7395961
## 8 Artificial neural network 0.6271186 0.5740364
```

Interestingly with the AUC being 0.5740364, ANNs seem to be a significantly strong classifier with high probability, according to the ROC curve, but gradient-boosted trees eventually outperform it as it moves up the curve. The confusion matrix indicates that the ANN predicts positives `Class = 1` well but not negatives, which could be the explanation for the greatest AUC value among all classifiers but not a particularly high accuracy. The ANN's greater AUC is partly due to its high true positive rate, but its total accuracy is only moderate because of its poor performance on the negative class.

Question 12

The R package `kernlab`, a support vector machine (SVM) is used to create another classifier.

Kernel-based machine learning methods for classification, regression, clustering, novelty detection, quantile regression and dimensional reduction. Among other methods `kernlab` includes Support Vector Machines, Spectral Clustering, Kernel PCA, Gaussian Processes and a QP solver.

A supervised learning model that can be used for both regression and classification, an SVM is a good fit for the data in this assignment. In binary classification, each instance is assigned to one of two classes by the SVM based on its search for a hyper plane, or decision boundary, that divides the training data. The examples from each class that are closest to one another in the feature space are known as support vectors, and they are used to build the hyper plane during training. In order to create as clear of a separation as possible, the goal is to maximize the distance between these points and the hyper plane. A kernel function, which converts the input into a higher-dimensional space, can also be implemented by SVMs. Thus, it is possible to find a hyper plane with a higher dimension that would be more effective in dividing the instances, (Lin, 2014).

As binary classification problems are much easier to solve many techniques exist to decompose multi-class classification problems into many binary classification problems (voting, error codes, etc.). Pairwise coupling (one against one) constructs a rule for discriminating between every pair of classes and then selecting the class with the most winning two-class decisions. By using Platt's probabilities output for SVM one can get a class probability for each of the k(k - 1)/2 models created in the pairwise classification. The couple method implements various techniques to combine these probabilities

`ksvm()` is the `kernlab` technique that fits the SVM model. The argument `vanilladot` represents the linear kernel, which is the kernel that is used. This was selected because there are many predictors, which reduces the data points' capacity to be distinguished from one another in a higher-dimensional space.

```
pd_svm <- ksvm(Class ~ ., data = pd_train, kernel = "vanilladot")
```

Similar to other classifiers, we use the test data to predict the model. Next, we build a confusion matrix so the accuracy can be estimated

```
pd_svm_acc
```

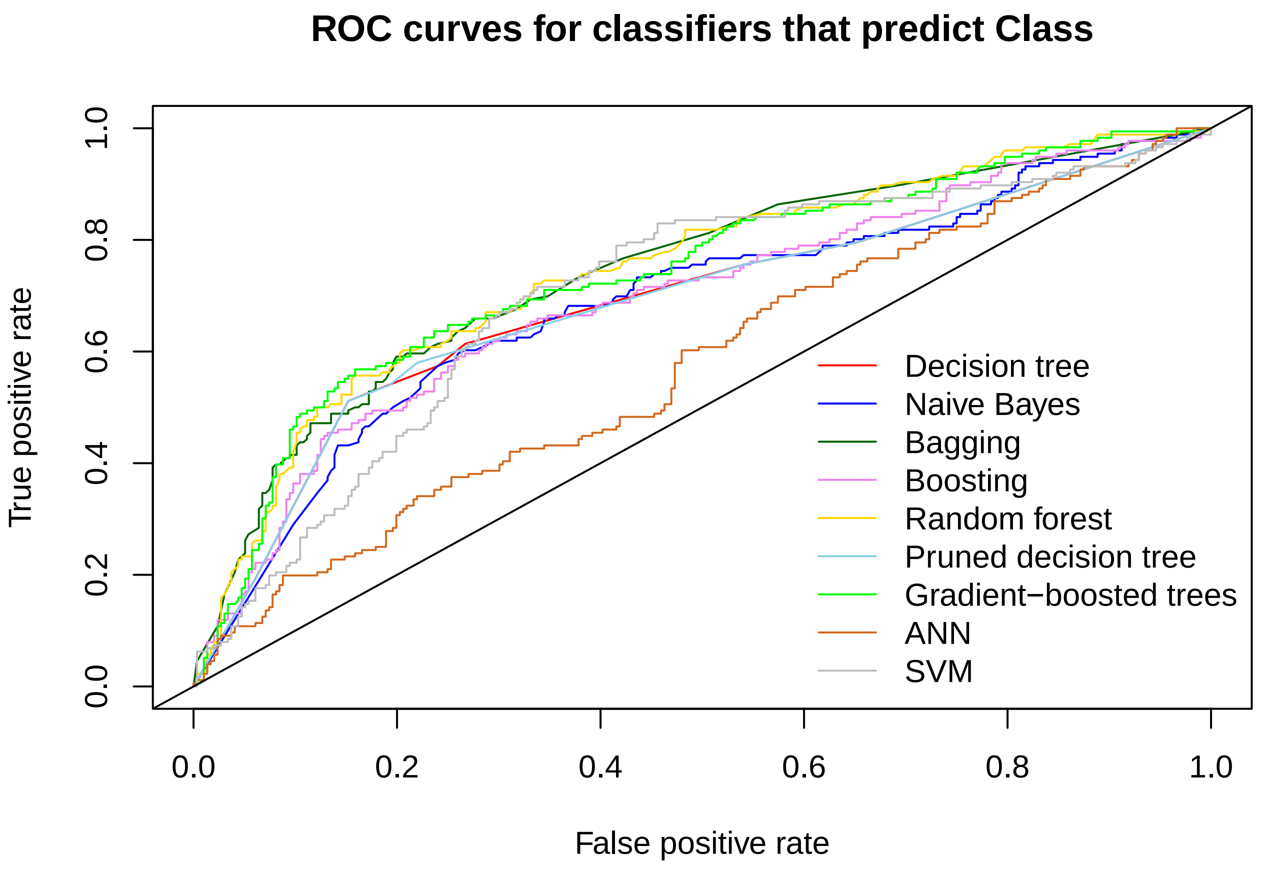
```
## [1] 0.6461864
```

```
pd_svm_cm
```

```
##                Actual Class
## Predicted Class   0   1
##               0 266 137
##               1  30  39
```

At 0.6461864, Support Vector Machines only seem to do particularly better than Naive Bayes classifiers, and slightly more than ANN for our data set. However, it still sits up there with the other highly accurate classifiers.

For a more in depth comparision, we again plot the ROC curve and caluculate the AUC for all classifiers, as well as portray the values in a table.



```
pd_accuracy_auc
```

```
##                        model  accuracy       auc
## 1              Decision tree 0.7224576 0.6879415
## 2     Naive Bayes classifier 0.3961864 0.6842464
## 3                    Bagging 0.7224576 0.7431089
## 4                   Boosting 0.6991525 0.6941800
## 5              Random forest 0.7330508 0.7450956
## 6       Pruned decision tree 0.7224576 0.6875576
## 7     Gradient-boosted trees 0.7394068 0.7395961
## 8 Artificial neural network 0.6271186 0.5740364
## 9     Support vector machine 0.6461864 0.7019733
```

Finally, we have AUC of 0.7019733, which outperforms quite a few classifiers at lower thresholds, but falling short of Bagging, Random Forest and Gradient-boosted trees. As such, we would pick Support Vector Machines to predict whether a website is a phishing site if there is a higher tolerance for false positives.

References

Lin, C.-J. (2014, June 14). Probability estimates for multi-class classification by Pairwise Coupling. The Journal of Machine Learning Research. https://proceedings.neurips.cc/paper_files/paper/2003/file/03e7ef47cee6fa4ae7567394b99912b7-Paper.pdf

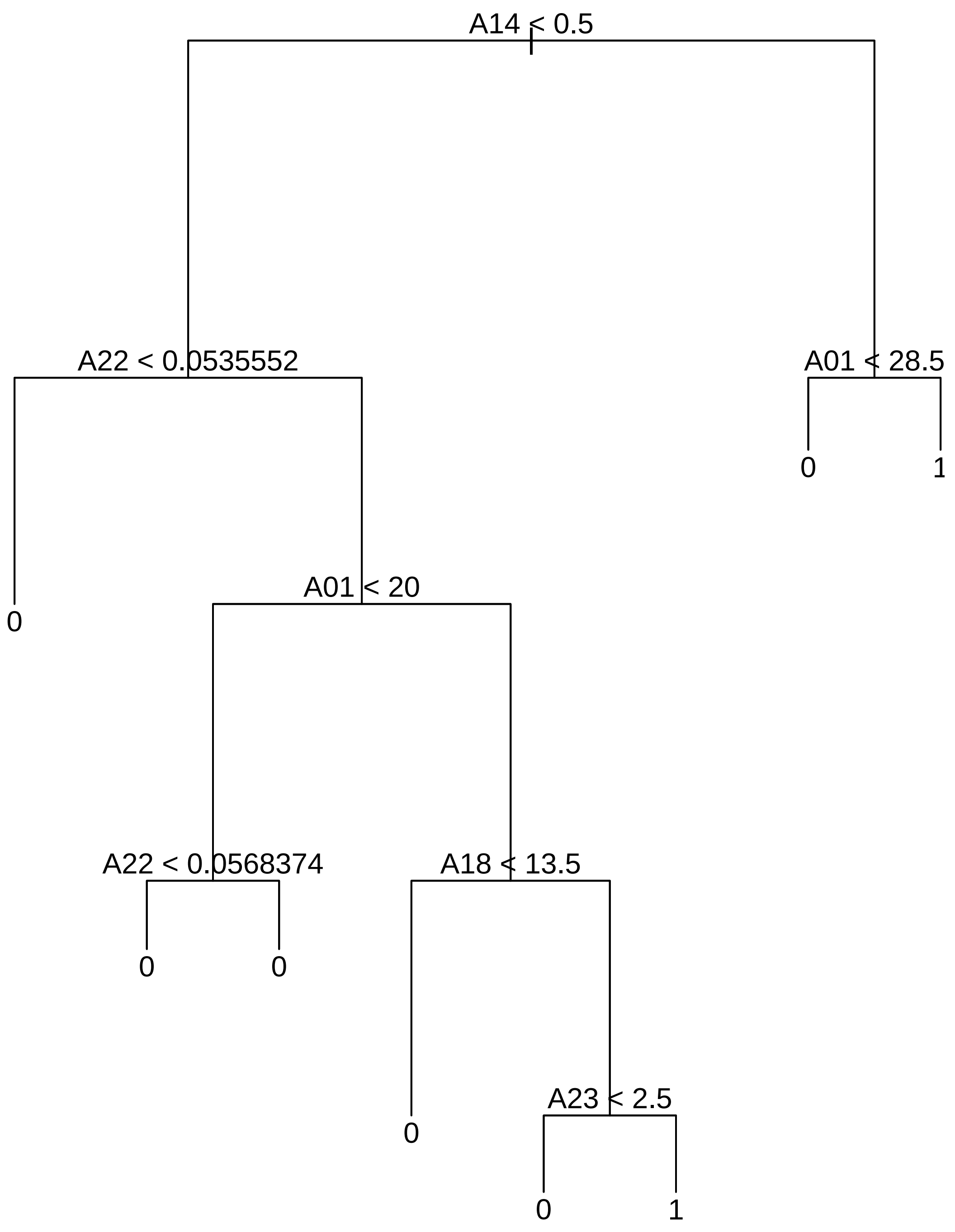
Appendix

Output of `summary(pd)` in Question 1.

```
##       A01             A02              A03                 A04
##  Min.   : 6.00   Min.   : 0.000   Min.   :0.000000   Min.   :2.000
##  1st Qu.:14.00   1st Qu.: 0.000   1st Qu.:0.000000   1st Qu.:2.000
##  Median :22.00   Median : 0.000   Median :0.000000   Median :3.000
##  Mean   :26.51   Mean   : 0.171   Mean   :0.001008   Mean   :2.767
##  3rd Qu.:39.00   3rd Qu.: 0.000   3rd Qu.:0.000000   3rd Qu.:3.000
##  Max.   :47.00   Max.   :39.000   Max.   :1.000000   Max.   :7.000
##                  NA's   :23       NA's   :15         NA's   :13
##       A05               A06              A07                 A08
##  Min.   :  0.0000   Min.   :0.0000   Min.   :0.000000   Min.   :0.1818
##  1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6667
##  Median :  0.0000   Median :0.0000   Median :0.000000   Median :1.0000
##  Mean   :  0.1166   Mean   :0.1223   Mean   :0.003543   Mean   :0.8414
##  3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :149.0000   Max.   :1.0000   Max.   :1.000000   Max.   :1.0000
##  NA's   :19         NA's   :21       NA's   :24         NA's   :18
##       A09              A10               A11                 A12
##  Min.   :0.00000   Min.   :0.00000   Min.   :  0.0000   Min.   : 48.0
##  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   :0.02747   Mean   :0.04042   Mean   :  0.1346   Mean   :317.2
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:  0.0000   3rd Qu.:418.0
##  Max.   :1.00000   Max.   :1.00000   Max.   :163.0000   Max.   :692.0
##  NA's   :34        NA's   :21        NA's   :17         NA's   :20
##       A13               A14              A15            A16
##  Min.   :  0.0000   Min.   :0.0000   Min.   :0.00   Min.   :0.00000
##  1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:0.00000
##  Median :  0.0000   Median :0.0000   Median :0.00   Median :0.00000
##  Mean   :  0.2618   Mean   :0.1467   Mean   :0.13   Mean   :0.04902
##  3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.00   3rd Qu.:0.00000
##  Max.   :447.0000   Max.   :1.0000   Max.   :1.00   Max.   :1.00000
##  NA's   :29         NA's   :23       NA's   :16     NA's   :21
##       A17              A18                A19               A20
##  Min.   :0.000   Min.   :   5.00   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:1.000   1st Qu.:  14.00   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.000   Median :  31.00   Median :0.0000   Median :0.0000
##  Mean   :1.182   Mean   :  57.02   Mean   :0.1145   Mean   :0.2279
##  3rd Qu.:1.000   3rd Qu.:  88.50   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :5.000   Max.   :3738.00   Max.   :1.0000   Max.   :1.0000
##  NA's   :19      NA's   :21        NA's   :17       NA's   :17
##       A21                A22                A23                A24
##  Min.   :0.00000   Min.   :0.01407   Min.   :   0.00   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.05034   1st Qu.:   7.00   1st Qu.:0.00697
##  Median :0.00000   Median :0.05778   Median :  90.00   Median :0.07996
##  Mean   :0.02568   Mean   :0.05553   Mean   :  66.28   Mean   :0.27074
##  3rd Qu.:0.00000   3rd Qu.:0.06305   3rd Qu.: 104.00   3rd Qu.:0.52291
##  Max.   :3.00000   Max.   :0.08164   Max.   :1074.00   Max.   :0.52291
##  NA's   :14        NA's   :16        NA's   :13        NA's   :16
##       A25                Class
##  Min.   :0.000000   Min.   :0.0000
##  1st Qu.:0.000000   1st Qu.:0.0000
##  Median :0.000000   Median :0.0000
##  Mean   :0.000322   Mean   :0.3645
##  3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :0.320000   Max.   :1.0000
##  NA's   :18
```

Diagram of initial decision tree plotted in Question 4.

```r
plot(pd_tree)
text(pd_tree, pretty = 0)
```

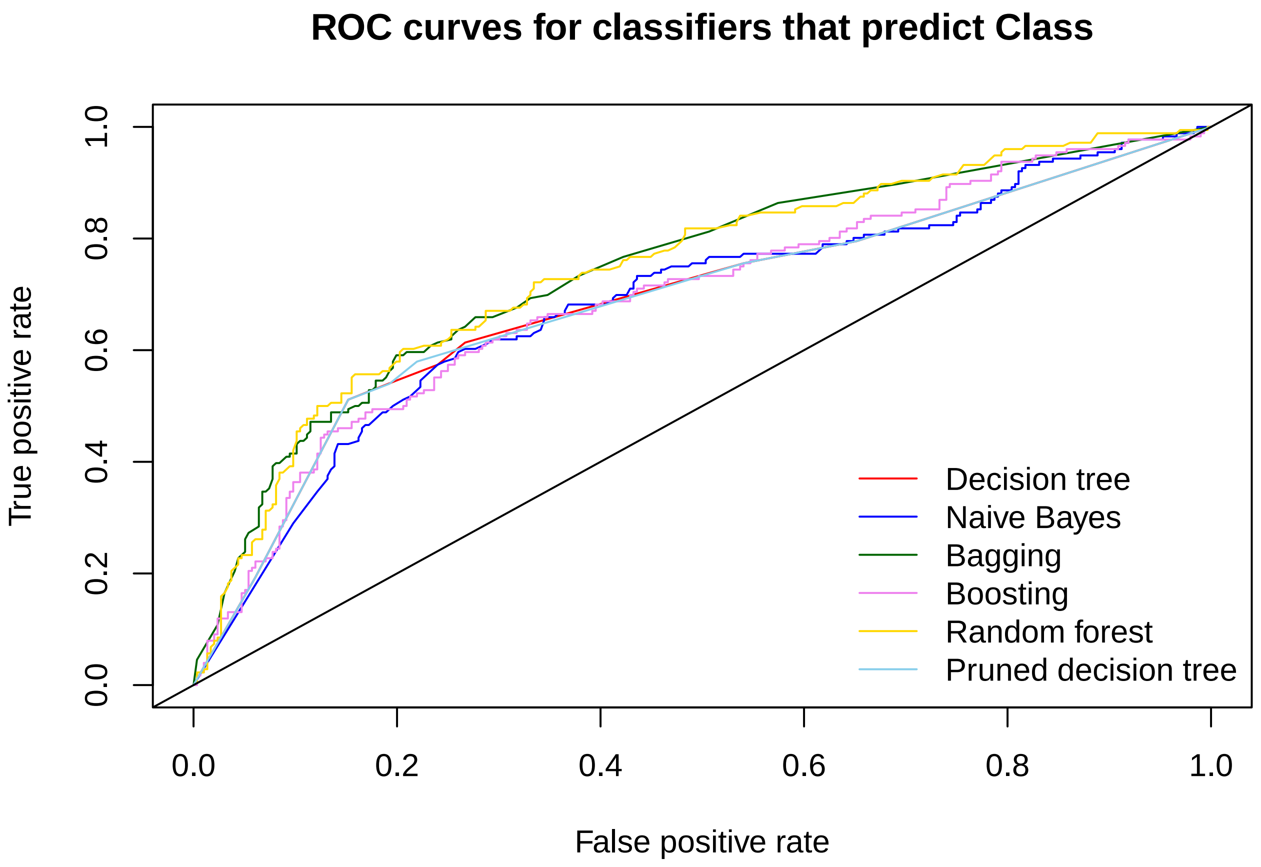


Code to make predictions, create confusion matrix, report accuracy, construct ROC curve and compute AUC value for the simple classifier (pruned decision tree) in Question 9.

```
pd_tree_pruned_predict <- predict(pd_tree_pruned, pd_test, type = "class")
pd_tree_pruned_cm <- table("Predicted Class" = pd_tree_pruned_predict,
                           "Actual Class" = pd_test$Class)
pd_tree_pruned_acc <- get_accuracy(pd_tree_pruned_cm)

pd_tree_pruned_predict_prob <- predict(pd_tree_pruned, pd_test, type = "vector")
pd_tree_pruned_pred <- prediction(pd_tree_pruned_predict_prob[, 2],  pd_test$Class)
pd_tree_pruned_perf <- performance(pd_tree_pruned_pred, "tpr", "fpr")
```

```
plot(pd_tree_perf, col = "red")
plot(pd_bayes_perf, col = "blue", add = TRUE)
plot(pd_bag_perf, col = "darkgreen", add = TRUE)
plot(pd_boost_perf, col = "violet", add = TRUE)
plot(pd_forest_perf, col = "gold", add = TRUE)
plot(pd_tree_pruned_perf, col = "skyblue", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
         "Pruned decision tree"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict Class")

pd_tree_pruned_auc <- performance(pd_tree_pruned_pred, "auc")@y.values[[1]]
```

ROC curves at Queestion 9.



Code to make predictions, create confusion matrix, report accuracy, construct ROC curve, compute AUC value and update classifier comparison table for the best tree-based classifier (gradient-boosted trees) in Question 10.

```
pd_lgbm_predict <- predict(pd_lgbm, as.matrix(pd_test_lgbm[1:21]),
                           type = "response")
pd_lgbm_cm <- table("Predicted Class" = ifelse(pd_lgbm_predict > 0.5, 1, 0),
                    "Actual Class" = pd_test_lgbm$Class)
```

```r
pd_lgbm_acc <- get_accuracy(pd_lgbm_cm)
pd_lgbm_pred <- prediction(pd_lgbm_predict, pd_test_lgbm$Class)
pd_lgbm_perf <- performance(pd_lgbm_pred, "tpr", "fpr")

plot(pd_tree_perf, col = "red")
plot(pd_bayes_perf, col = "blue", add = TRUE)
plot(pd_bag_perf, col = "darkgreen", add = TRUE)
plot(pd_boost_perf, col = "violet", add = TRUE)
plot(pd_forest_perf, col = "gold", add = TRUE)
plot(pd_tree_pruned_perf, col = "skyblue", add = TRUE)
plot(pd_lgbm_perf, col = "green", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
         "Pruned decision tree", "Gradient-boosted trees"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue", "green"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict Class")

pd_lgbm_auc <- performance(pd_lgbm_pred, "auc")@y.values[[1]]
pd_accuracy_auc <- rbind(pd_acc_auc,
                         data.frame(model = "Gradient-boosted trees",
                                    accuracy = pd_lgbm_acc, auc = pd_lgbm_auc))
```