FIT3152
Assignment 2
Daniel Nguyen
32471033

Table of Contents

Question 1
The class column determines whether a website is for phishing or is legitimate, the values can only be 1 or 0, where 0 indicates a legitimate website, and 1 indicates a phishing website. The mean is 0.37, which means that 37% of websites in our personal dataset are legitimate, and 64% are phishing sites.

Variable descriptions and standard deviations are contained in the appendix.

For the basic models, I did not omit any values, however I have employed variable selection for some models, and have detailed the process in the relevant question answers.

Question 2

- Convert class column values to factors
- Missing values were handled on a case by case basis, where some models could simply ignore missing values, while others needed them to be omitted

Question 3
I have adapted the given code to create training and testing datasets in the following way:

```
set.seed(32471033)
train.row = sample(1:nrow(PD), 0.7*nrow(PD))
PD.train = PD[train.row,]
PD.test = PD[-train.row,]
```

Question 4
Each classifier model was fit using the default package parameters when possible.

Random sampling using my student number as the seed value, and with replacement, was used for the bagging classifier model.

Question 5
Decision Tree:

	Actual 0	**Actual 1**
Predicted 0	377	136
Predicted 1	22	65

Correct: 442
Incorrect: 158
73.7% Predicted Correctly

Naive Bayes:

	Actual 0	**Actual 1**
Predicted 0	35	4
Predicted 1	364	197

Correct: 232
Incorrect: 368
38.7% Predicted Correctly

Bagging:

	Actual 0	**Actual 1**
Predicted 0	320	102
Predicted 1	79	99

Correct: 419
Incorrect: 181
69.9% Predicted Correctly

Boosting:

	Actual 0	**Actual 1**
Predicted 0	310	82
Predicted 1	89	119

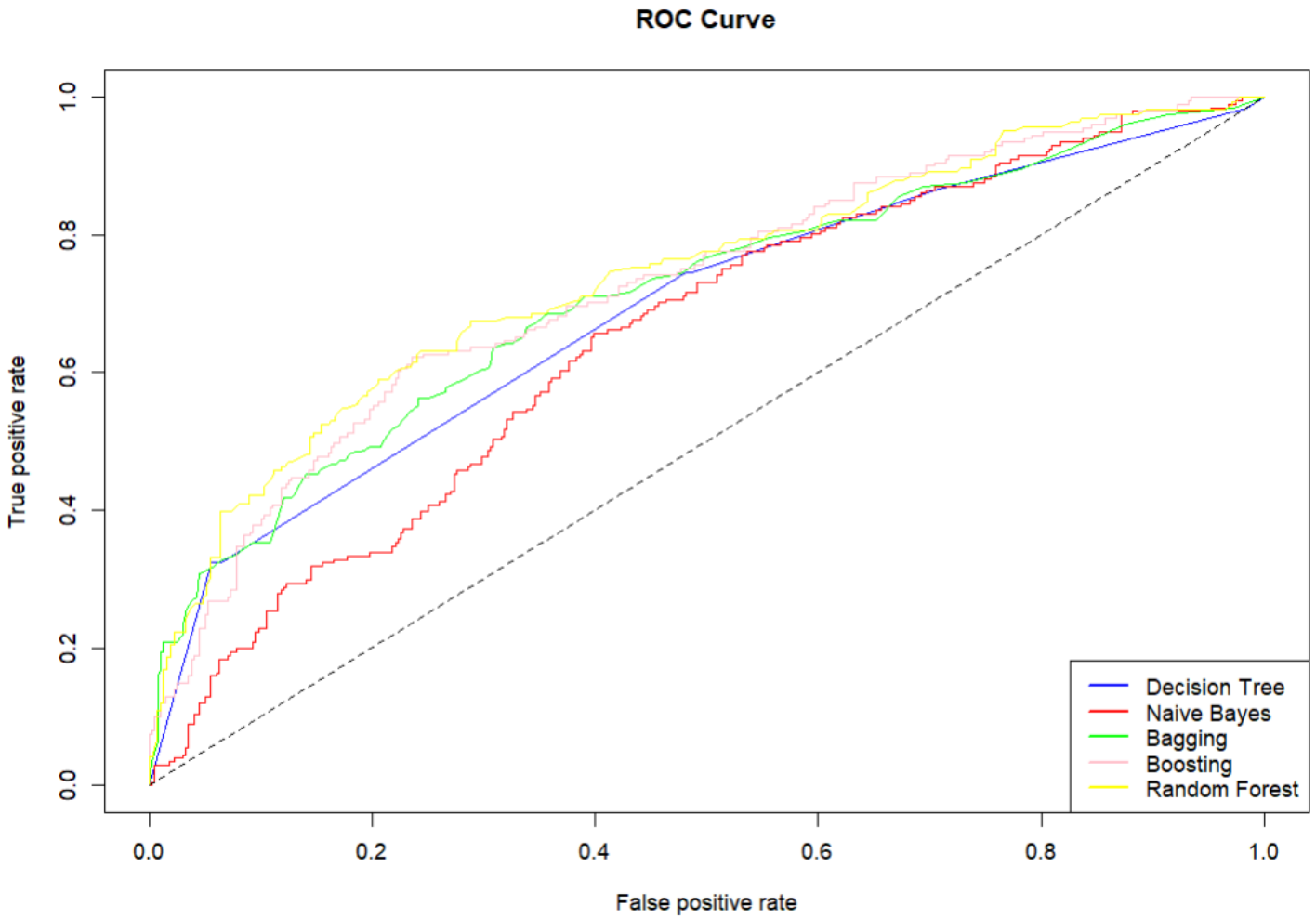
Correct: 429
Incorrect: 171
71.5% Predicted Correctly

Random Forest:

	Actual 0	**Actual 1**
Predicted 0	270	87
Predicted 1	42	79

Correct: 349
Incorrect: 129
73% Predicted Correctly

Question 6



	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Area under Curve	0.6886869	0.6514171	0.7107944	0.7264928	0.7401722

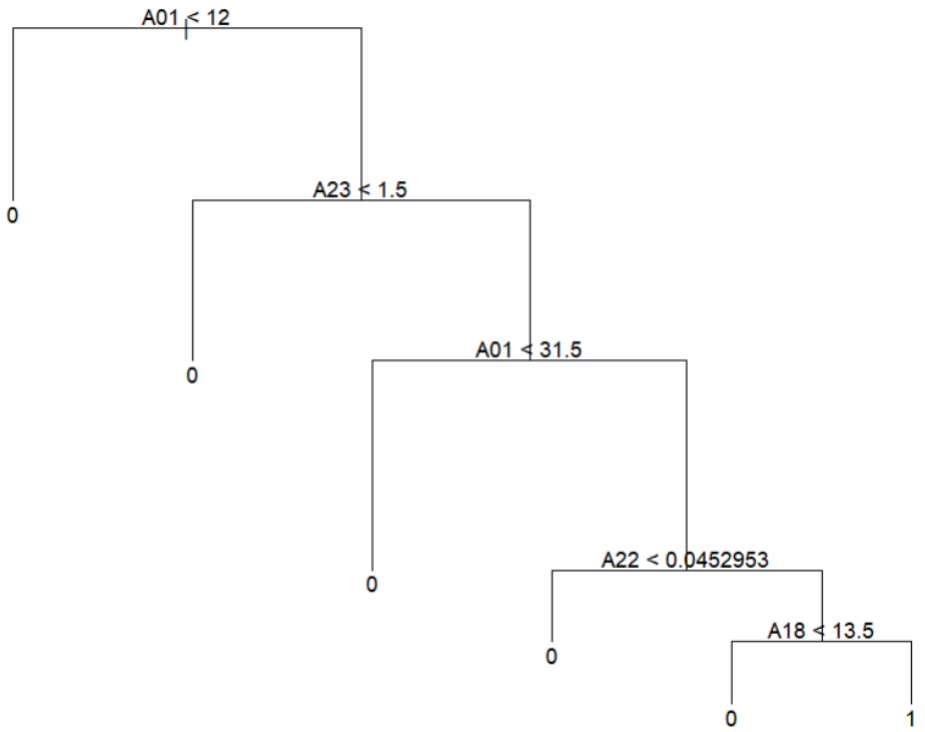
Question 7

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Correct	442	232	419	429	349
Incorrect	158	368	181	171	129
% Accuracy	73.7%	38.7%	69.9%	71.5%	73%
Area under Curve	0.6886869	0.6514171	0.7107944	0.7264928	0.7401722

When using the prediction function for our decision tree, only 9 fp, tp, fn and tn values were returned, which results in a less detailed curve, as seen in our combined ROC curve. This affects our calculations for the area under the curve, which may in fact be higher, based on our correct prediction percentages. The random forest classifier has the highest AUC, while the decision tree classifier has the highest accuracy, so these 2 classifiers are the best, but I cannot make the conclusion that there is any single “best” classifier

Question 8
A table of importance values for our bagging, boosting and random forest models are available in the appendix.

Decision Tree:



We can see from our decision tree plot that the important variables are:

- A01
- A23
- A22
- A18

Naive Bayes:
We cant view the important variables in our Naive Bayes, as the data itself constructs the model, and there is no variable importance in this model.

Bagging:
Looking at the importance values for our bagging model, we can see that some variables are much more important than others in deciding the probabilities of whether a website is phishing or legitimate. Some of the most important variables, with the highest importance values are:

- A01 (25.4)
- A18 (18.5)
- A22 (18.1)
- A23 (19.5)

Boosting:
We can do the same for our boosting model, where the most important variables are:

- A01 (12.1)
- A18 (14.4)
- A22 (25.1)
- A23 (14.0)

Random Forest:
Again, for the Random Forest classifier, the most important variables are:

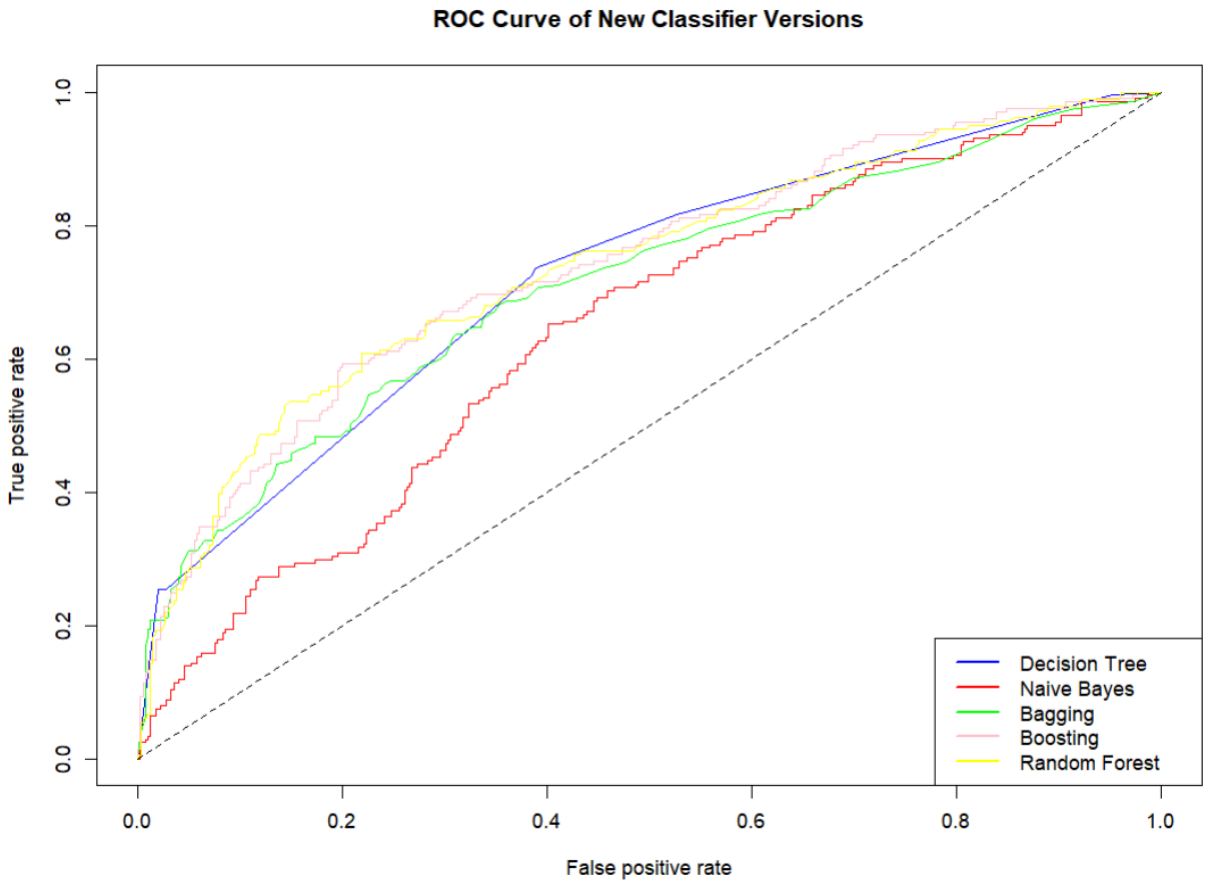
- A01 (75.4)
- A08 (35.2)
- A18 (66.5)
- A22 (74.6)
- A23 (63.8)

The Random Forest produced importance values that were different from the previous 2 models, where it placed higher importance on certain variables that were previously valued very low.

Excluding Variables:
Some of the variables seem to have an extremely low importance in all of our created models where importance can be measured. This suggests that it is possible to remove some of the variables without seriously affecting the performance of the models. I have omitted the variables that have an importance value of below 1 in the bagging and boosting models, and below 5 in the random forest model, and created new instances of each model on the new data. The variables I removed were:

- A03
- A05
- A07
- A09
- A10
- A11
- A13
- A21
- A25

The new ROC curve is:



	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Old AUC	0.6886869	0.6514171	0.7107944	0.7264928	0.7401722
New AUC	0.7222534	0.6427636	0.7102208	0.7393234	0.7385303
Absolute Relative Change (%)	4.9%	1.3%	0.0%	1.8%	0.2%

We can see that after removing some variables according to our rule based on importance values, and creating new models based on this modified data, that overall, the performance of the models is not significantly impacted. The model that changed the most was our decision tree, where performance actually improved.

Question 9

I have chosen to simplify the naive bayes model so it is possible to use it to make a prediction by hand. The e1071 implementation of the naive bayes classifier assumes normal distribution, and uses the probability density function of a normal distribute based on the mean and standard deviations of each variable separated by class. While this is technically possible to do by hand, it is extremely tedious, so I have chosen to make it much simpler, I have separated each variable into 2 categories, separated by the median, to have a higher and lower half, and chosen variables to include in the model that will work with this simplification, in addition to being considered important by the other base models I have explored earlier. These variables are:

- A01
- A18
- A22
- A23

These, according to the importance values of our base random forest model, are the 4 most important variables.

The table of probabilities is below:

A01, Median = 25	Below Median	Above or Equal to Median
Class 0	0.5298329	0.4701671
Class 1	0.2802303	0.7197697

A18, Median = 30	Below Median	Above or Equal to Median
Class 0	0.5704057	0.4295943
Class 1	0.3781190	0.6218810

A22, Median = 0.05804	Below Median	Above or Equal to Median
Class 0	0.575179	0.424821
Class 1	0.378119	0.621881

A23, Median = 100	Below Median	Above or Equal to Median
Class 0	0.4367542	0.5632458
Class 1	0.5508637	0.4491363

Class 0 Count	Class 1 Count
838	521

It is now simple to calculate posterior probabilities and make predictions, provided that the information for these 4 variables is not missing.

When choosing a model to simplify, bagging, boosting and random forest are too difficult to be modelled by hand, and the decision tree is already simple enough to be used to make predictions by hand. This left the naive bayes classifier as the only base model that could be simplified enough to be used to make predictions manually. I chose the method of separating each variable into higher and lower categories to allow for easy calculation of posterior probabilities. The prediction results are below:

Correct	Incorrect	Accuracy	AUC
391	191	67.2%	0.6710792

Our predictions are not as good when compared to the other base models, however this is to be expected due to the simplicity of this model. Additionally, the model actually greatly outperforms our base Naive Bayes model, which I suspect is due to our aggressive variable selection.

Question 10

To create the best tree based classifier I can, I began with the random forest classifier, as it was the one with the initially highest AUC and second highest accuracy. The steps I took to improve the model are the following:

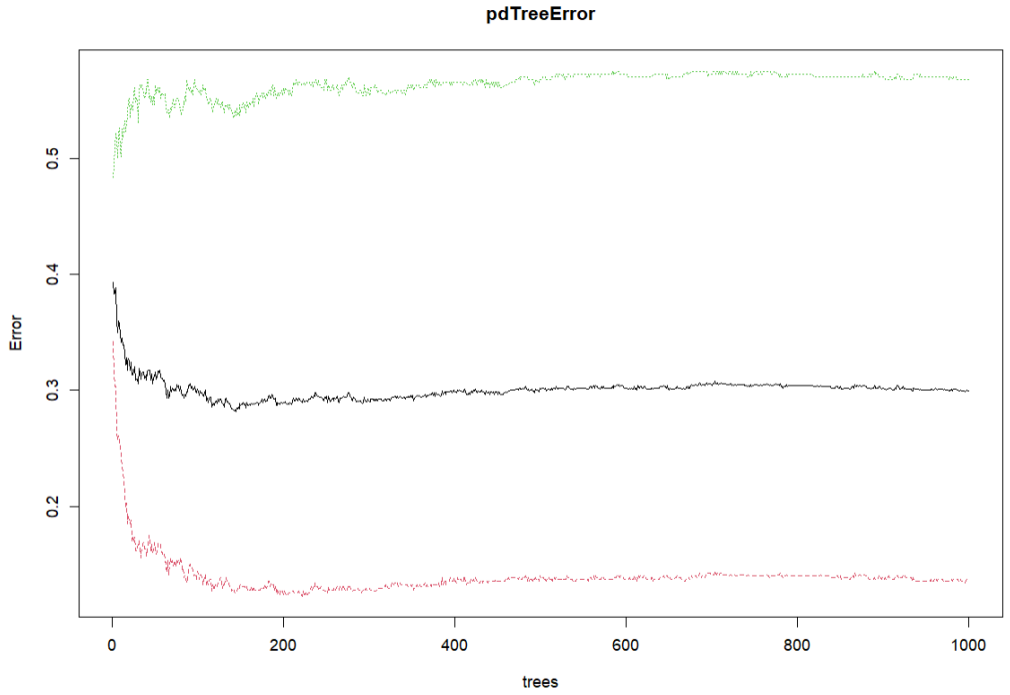
- Remove unimportant variables

I began by looking at the importance values for our random forest with all variables, and taking only the variables with an importance score of more than 10. I then used the function rfcv to find out how many more variables I could remove without negatively affecting our model performance.

```
$error.cv
        9         8         7         6         5         4         3         2         1
0.3040724 0.2950226 0.2914027 0.2977376 0.3058824 0.3257919 0.2859729 0.2841629 0.3628959
```

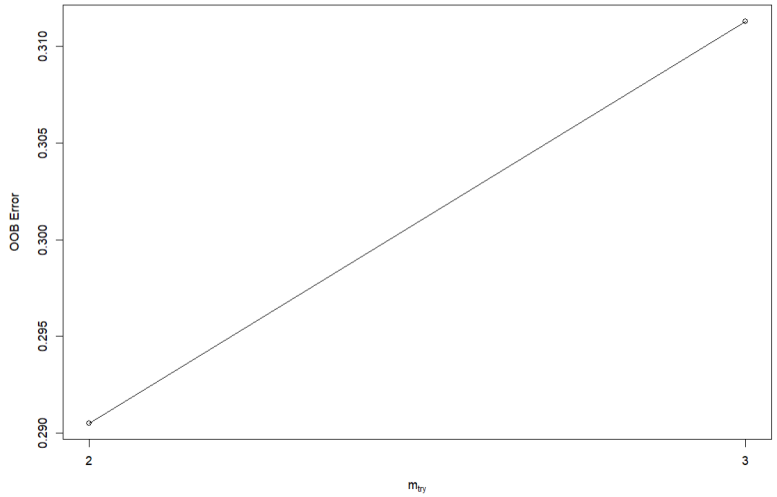
We can see that the lowest error rate is when we only keep 2 variables, however I believe that this oversimplifies the model, so I have decided to remove the 2 least important variables, as it improves our error rate, as seen in the function output.

- Find the best number of trees



We can see that after around 200 trees in the forest, the error values stabilise, so I have chose to include 300 trees in our final model.

- Find optimal number of variables to sample per tree (mTry)



I used the tuneRF function included within randomForest to determine the optimal mTry value, which has shown that the best value to use is 2.

- Implement sampling with replacement

Additionally, I have also made use of the sampsize parameter, choosing a value of 0 = 300, 1 = 160, to better reflect the spread of the class variable in our dataset.

After implementing our optimisations, our results are:

	Actual 0	**Actual 1**
Predicted 0	277	90
Predicted 1	35	76

	Accuracy	AUC
Base Model	73%	0.7401722
Optimised Model	73.8%	0.7387338

In the end, we have improved our accuracy by 1.1%, while our AUC has been reduced by 0.2% compared to our base random forest model. This result has our optimised model as the most accurate, and with the second highest AUC by only a very small margin.

Question 11
When creating my neural network, there are 2 main steps that need to be taken before constructing the model. These are attribute selection and preprocessing.

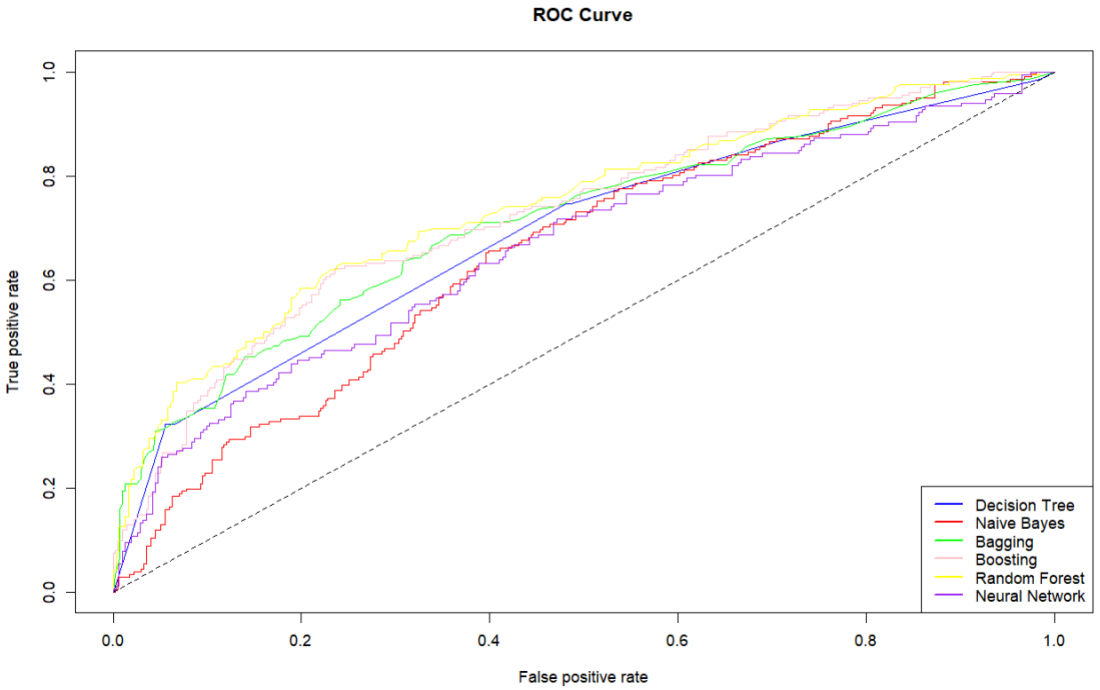
When selecting attributes, I used my initial selection of variables that was made for the optimised random forest classifier, which were:

- A01
- A08
- A12
- A14
- A17
- A18
- A22
- A23
- A24

When preprocessing data, we have to normalise our data, and account for missing values. To do this I omitted NA values, which removed approximately 20% of our data overall. To normalise data, I used the scale function in R, and imported the class column without normalisation, as it is our output variable. This ensures that different columns with different scales are treated equally by the neural network to prevent bias.

Finally, I took the mean of the sum of the number of input nodes and output nodes (11) to arrive at a node count of 6. Perhaps more layers would allow the neural network to gain more detailed insight into the relationships between the variables, however in this case, I am limited by computing power.

The results are shown below:



Neural Network Results	**Actual 0**	**Actual 1**
Predicted 0	260	100
Predicted 1	52	66

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	Neural Network
Correct	442	232	419	429	349	326
Incorrect	158	368	181	171	129	152
% Accuracy	73.7%	38.7%	69.9%	71.5%	73%	68.2%
Area under Curve	0.6886869	0.6514171	0.7107944	0.7264928	0.7401722	0.6619748

The AUC and accuracy are lower than that of the other models, however, it is still similar, and not much worse.

Neural networks excel at improving their performance when given large sets of data compared to the tree based classifiers, and perhaps the data set was too small after removing all missing values to show the advantage of using a neural network. Additionally, a deeper network, with more hidden layers and hidden nodes may increase the performance of the model, however I am limited in how complex I can make the model due to the time it takes to train such a complex system with my available computing power.

Question 12
I have chosen to implement support vector classification as my new model. To do this I used the svm (support vector machine) functionality of the e1071 package.

Package Link:
https://www.rdocumentation.org/packages/e1071/versions/1.7-14

Function Documentation:
https://www.rdocumentation.org/packages/e1071/versions/1.7-14/topics/svm

When looking at data, we can plot it on a plane, in the case of our phishing website data, the plane will be on a higher dimension scaling with the number of input variables. An SVM calculates a “hyperplane” to separate the training data into the separate classes.

Calculating these higher dimensions gets exponentially more computationally expensive the more input variables we have, but we can bypass this by using what is called a “kernel trick,” which is equivalent to mapping out higher dimensions via dot product calculations.

The calculated hyperplane aims to maximise the distance between it and the data points of the different classes, which maximises the amount of variable variance that can be tolerated.

It follows that NA values are not tolerated in the model, so imputation or omission of missing values must be done. I have chosen to omit the missing values.

As the SVM deals with points that are inside/outside the different regions of the plane, it is not a probabilistic model, and thus we cannot use a metric like AUC to calculate the performance of the model. The accuracy details of our SVC are below:

Classifier Results	**Actual 0**	**Actual 1**
Predicted 0	287	116
Predicted 1	25	50

Correct: 337
Incorrect: 141
70.5% Predicted Correctly

Looking at the accuracy of the model without any optimisation, it is comparable to our other models. Optimisation of this model would involve variable selection, and the selection of the “kernel trick”, as each kernel trick method will transform the data differently, leading to differences in results.

Appendix

Variable Descriptions:

```
      A01                A02                A03                A04                A05                A06                A07                A08                A09
 Min.   : 7.00   Min.   : 0.0000   Min.   :0.000000   Min.   :2.000   Min.   : 0.00000   Min.   :0.0000   Min.   :0.00000   Min.   :0.1711   Min.   :0.00000
 1st Qu.:13.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.6802   1st Qu.:0.00000
 Median :25.00   Median : 0.0000   Median :0.000000   Median :3.000   Median : 0.00000   Median :0.0000   Median :0.00000   Median :1.0000   Median :0.00000
 Mean   :25.37   Mean   : 0.2293   Mean   :0.000504   Mean   :2.745   Mean   : 0.04087   Mean   :0.1087   Mean   :0.00455   Mean   :0.8445   Mean   :0.02228
 3rd Qu.:33.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.00000
 Max.   :49.00   Max.   :84.0000   Max.   :1.000000   Max.   :7.000   Max.   :15.00000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000   Max.   :1.00000
                 NA's   :20        NA's   :16         NA's   :21      NA's   :18         NA's   :12        NA's   :22        NA's   :13        NA's   :25
      A10                A11                A12                A13                A14                A15                A16                A17                A18
 Min.   :0.00000   Min.   : 0.00000   Min.   :115.0   Min.   : 0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :   5.00
 1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:232.0   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:  12.75
 Median :0.00000   Median : 0.00000   Median :232.0   Median : 0.00000   Median :0.0000   Median :0.0000   Median :0.00000   Median :1.000   Median :  31.00
 Mean   :0.05121   Mean   : 0.05449   Mean   :320.4   Mean   : 0.04883   Mean   :0.1318   Mean   :0.1479   Mean   :0.04644   Mean   :1.152   Mean   :  58.78
 3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:449.0   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:1.000   3rd Qu.:  89.25
 Max.   :1.00000   Max.   :15.00000   Max.   :692.0   Max.   :24.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :5.000   Max.   :1690.00
 NA's   :8         NA's   :18         NA's   :23      NA's   :34         NA's   :20       NA's   :12       NA's   :19        NA's   :29      NA's   :24
      A19                A20                A21                A22                A23                A24                A25                Class
 Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.01486   Min.   :  0.00   Min.   :0.000000   Min.   :0.000000   Min.   :0.00
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.05097   1st Qu.:  8.00   1st Qu.:0.005977   1st Qu.:0.000000   1st Qu.:0.00
 Median :0.0000   Median :0.0000   Median :0.00000   Median :0.05806   Median :100.00   Median :0.079963   Median :0.000000   Median :0.00
 Mean   :0.1018   Mean   :0.2294   Mean   :0.02418   Mean   :0.05608   Mean   : 68.28   Mean   :0.263846   Mean   :0.000346   Mean   :0.37
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.06307   3rd Qu.:104.00   3rd Qu.:0.522907   3rd Qu.:0.000000   3rd Qu.:1.00
 Max.   :1.0000   Max.   :1.0000   Max.   :2.00000   Max.   :0.07980   Max.   :767.00   Max.   :0.522907   Max.   :0.197000   Max.   :1.00
 NA's   :15       NA's   :17       NA's   :15        NA's   :11        NA's   :25       NA's   :25         NA's   :31
```

Variable Standard Deviations:

```
         A01          A02          A03          A04          A05          A06          A07          A08          A09          A10          A11          A12
1.324227e+01 2.615488e+00 2.245066e-02 5.480614e-01 6.192641e-01 3.112803e-01 6.731745e-02 2.206599e-01 1.476251e-01 2.204706e-01 5.675541e-01 1.432231e+02
         A13          A14          A15          A16          A17          A18          A19          A20          A21          A22          A23          A24
8.913815e-01 3.383785e-01 3.550776e-01 2.104917e-01 6.035604e-01 9.212139e+01 3.024129e-01 4.205854e-01 1.662551e-01 1.021448e-02 6.281743e+01 2.517752e-01
         A25        Class
6.680783e-03 4.829251e-01
```

Variable Importance Values:

	Bagging	Boosting	Random Forest
A01	25.39641469	12.14747925	75.4083612
A02	0.33745144	1.51440439	6.66793686
A03	0.00000000	0.00000000	0.01750303
A04	0.32979534	1.49720813	8.93042577
A05	0.00000000	0.08646732	0.71198085
A06	0.75122025	1.04306548	6.37441389
A07	0.00000000	0.00000000	0.19129568
A08	6.02003950	9.13366637	35.17458089
A09	0.04371191	0.40325445	2.23078554
A10	0.08121892	0.88678923	3.16281009
A11	0.00000000	0.53488268	2.08372635
A12	3.47227097	6.02479339	27.51368237
A13	0.00000000	0.00000000	0.36035577
A14	1.43270004	0.68337115	10.56409344
A15	0.46145034	0.88250458	6.81251906
A16	0.91706223	0.88711507	5.15513218
A17	0.33002577	2.34771329	11.14689124
A18	18.49056649	14.38148171	66.4723592
A19	0.66240595	0.71687519	6.18692091
A20	0.57306164	1.14307696	8.89073866
A21	0.00000000	0.71065722	1.68702151
A22	18.10154776	25.10234290	74.55375041
A23	19.51257247	14.02799255	63.75497098
A24	3.08648430	5.84485869	29.03274886
A25	0.00000000	0.00000000	0.32285311

R Packages Used

tree
e1071
adabag
rpart
randomForest
ROCR
dplyr
neuralnet

All R packages used have been used as demonstrated in unit tutorials and applied sessions, with the addition of support vector machine functionality from package e1071, and randomForest parameter tuning functionality from package randomForest.

R Code

```
#Note, I have also included the r code as a .r file in the submission

#package handling
install.packages("tree")
install.packages("e1071")
install.packages("adabag")
install.packages("randomForest")
install.packages("ROCR")
install.packages("neuralnet")
library(tree)
library(e1071)
library(adabag)
library(rpart)
library(randomForest)
library(ROCR)
library(dplyr)
library(neuralnet)

#data setup
rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32471033) # Your Student ID is the random seed
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

#q1
#get summaries
summary(PD)

#get standard deviations
sapply(PD, sd, na.rm = TRUE)

#q2
#convert Class to factors
PD$Class <- factor(PD$Class, levels = c(0, 1), labels = c(0, 1))

#q3
```

```
set.seed(32471033)
train.row = sample(1:nrow(PD), 0.7*nrow(PD))
PD.train = PD[train.row,]
PD.test = PD[-train.row,]

#q4
#decision tree
pdDT <- tree(Class~., data = PD.train)

plot(pdDT)
text(pdDT)

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

#bagging
#take sample with replacement
#see https://www.ibm.com/topics/bagging
set.seed(32471033)
sub <-c(sample(1:1400, 700, replace = TRUE))

#fit model
pdBag <- bagging(Class~., data = PD.train[sub,])

#boosting
pdBoost <- boosting(Class~., data = PD.train)

#random forest
pdRF <- randomForest(Class~., data = PD.train, na.action = na.exclude)

#q5
#decision tree
pdDTPredict <- predict(pdDT, PD.test, type = "class")

#confusion matrix
table(predicted = pdDTPredict, actual = PD.test$Class)

#naive bayes
pdNBPredict <- predict(pdNB, PD.test)

#confusion matrix
table(predicted = pdNBPredict, actual = PD.test$Class)

#bagging
pdBagPredict <- predict(pdBag, PD.test)

#confusion matrix
table(predicted = pdBagPredict$class, actual = PD.test$Class)

#boosting
pdBoostPredict = predict(pdBoost, PD.test)

#confusion matrix
table(predicted = pdBoostPredict$class, actual = PD.test$Class)
```

```
#random forest
pdRFPredict <- predict(pdRF, PD.test)

#confusion matrix
table(predicted = pdRFPredict, actual = PD.test$Class)

#q6
#get raw probability data
#decision tree
pdDTPredict <- predict(pdDT, PD.test)[,2] %>%
  ROCR::prediction(., PD.test$Class)
pdDTPerf <- performance(pdDTPredict, "tpr", "fpr")

#naive bayes
pdNBPredict <- predict(pdNB, PD.test, type = 'raw')[,2] %>%
  ROCR::prediction(., PD.test$Class)
pdNBPerf <- performance(pdNBPredict, "tpr", "fpr")

#bagging
pdBagPredict <- predict(pdBag, PD.test)$prob[,2] %>%
  ROCR::prediction(., PD.test$Class)
pdBagPerf <- performance(pdBagPredict, "tpr", "fpr")

#boosting
pdBoostPredict <- predict(pdBoost, PD.test)$prob[,2] %>%
  ROCR::prediction(., PD.test$Class)
pdBoostPerf <- performance(pdBoostPredict, "tpr", "fpr")

#random forest
pdRFPredict <- na.omit(predict(pdRF, PD.test, type = 'prob'))[,2] %>%
  ROCR::prediction(., na.omit(PD.test)$Class)
pdRFPerf <- performance(pdRFPredict, "tpr", "fpr")

#plot the graph
plot(pdDTPerf, main = "ROC Curve", col = "blue")
plot(pdNBPerf, col = "red", add = TRUE)
plot(pdBagPerf, col = "green", add = TRUE)
plot(pdBoostPerf, col = "pink", add = TRUE)
plot(pdRFPerf, col = "yellow", add = TRUE)
lines(c(0, 1), c(0, 1), col = "black", lty = 2)
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"), col
= c("blue", "red", "green", "pink", "yellow"), lwd = 2)

#get areas under the curve
performance(pdDTPredict, "auc")@y.values
performance(pdNBPredict, "auc")@y.values
performance(pdBagPredict, "auc")@y.values
performance(pdBoostPredict, "auc")@y.values
performance(pdRFPredict, "auc")@y.values

#q8
#decision tree variable importance
#plot the decision tree and look at the variables
```

```
plot(pdDT)
text(pdDT, pretty = 0)

#bagging
pdBag$importance

#boosting
pdBoost$importance

#randomForest
pdRF$importance

#create new training and testing datasets after removing variables
excludedVars <- c("A03", "A05", "A07", "A09", "A10", "A11", "A13", "A21", "A25")

PD.train2 <- PD.train[, -which(names(PD.train) %in% excludedVars)]
PD.test2 <- PD.test[, -which(names(PD.test) %in% excludedVars)]

#create models
pdDT2 <- tree(Class~., data = PD.train2)

pdNB2 <- naiveBayes(Class~., data = PD.train2)

set.seed(32471033)
sub <-c(sample(1:1400, 700, replace = TRUE))
pdBag2 <- bagging(Class~., data = PD.train2[sub,])

pdBoost2 <- boosting(Class~., data = PD.train2)

pdRF2 <- randomForest(Class~., data = PD.train2, na.action = na.exclude)

#get new performance data
pdDTPredict2 <- predict(pdDT2, PD.test2)[,2] %>%
  ROCR::prediction(., PD.test2$Class)
pdDTPerf2 <- performance(pdDTPredict2, "tpr", "fpr")

pdNBPredict2 <- predict(pdNB2, PD.test2, type = 'raw')[,2] %>%
  ROCR::prediction(., PD.test2$Class)
pdNBPerf2 <- performance(pdNBPredict2, "tpr", "fpr")

pdBagPredict2 <- predict(pdBag2, PD.test2)$prob[,2] %>%
  ROCR::prediction(., PD.test2$Class)
pdBagPerf2 <- performance(pdBagPredict2, "tpr", "fpr")

pdBoostPredict2 <- predict(pdBoost2, PD.test2)$prob[,2] %>%
  ROCR::prediction(., PD.test2$Class)
pdBoostPerf2 <- performance(pdBoostPredict2, "tpr", "fpr")

pdRFPredict2 <- na.omit(predict(pdRF2, PD.test2, type = 'prob'))[,2] %>%
  ROCR::prediction(., na.omit(PD.test2)$Class)
pdRFPerf2 <- performance(pdRFPredict2, "tpr", "fpr")

#plot new ROC graph
plot(pdDTPerf2, main = "ROC Curve of New Classifier Versions", col = "blue")
```

```
plot(pdNBPerf2, col = "red", add = TRUE)
plot(pdBagPerf2, col = "green", add = TRUE)
plot(pdBoostPerf2, col = "pink", add = TRUE)
plot(pdRFPerf2, col = "yellow", add = TRUE)
lines(c(0, 1), c(0, 1), col = "black", lty = 2)
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"), col
= c("blue", "red", "green", "pink", "yellow"), lwd = 2)

#get areas under the curve
performance(pdDTPredict2, "auc")@y.values
performance(pdNBPredict2, "auc")@y.values
performance(pdBagPredict2, "auc")@y.values
performance(pdBoostPredict2, "auc")@y.values
performance(pdRFPredict2, "auc")@y.values

#q9
#create train and test datasets
nbTrain <- PD.train[, c(1, 18, 22, 23, 26)]
nbTrain <- na.omit(nbTrain)
nbTest <- PD.test[, c(1, 18, 22, 23, 26)]
nbTest <- na.omit(nbTest)

#summary table to view median values
summary(nbTrain)

#splitting and factoring function
medianFactor <- function(x) {
  medianVal <- median(x)
  newCol <- ifelse(x < medianVal, 0, 1)
  newCol <- factor(newCol, levels = c(0, 1), labels = c("Below", "Above or Equal"))
  return(newCol)
}

#apply function to datasets
nbTrain[, c(1:4)] <- lapply(nbTrain[, c(1:4)], medianFactor)
nbTest[, c(1:4)] <- lapply(nbTest[, c(1:4)], medianFactor)

#fit model
pdNBSimple <- naiveBayes(Class~., data = nbTrain)

#model probability tables
pdNBSimple$tables

#model class distribution
pdNBSimple$apriori

#make predictions
pdNBSimplePredict <- predict(pdNBSimple, nbTest)

#confusion matrix
table(predicted = pdNBSimplePredict, actual = nbTest$Class)

#get AUC
pdNBSimplePrediction <- predict(pdNBSimple, nbTest, type = 'raw')[,2] %>%
```

```
  ROCR::prediction(., nbTest$Class)
performance(pdNBSimplePrediction, "auc")@y.values

#q10
#remove NA values
PD.trainOmit <- na.omit(PD.train)
PD.testOmit <- na.omit(PD.test)

PD.trainOmit <- PD.trainOmit[, c(1, 8, 12, 14, 17, 18, 22, 23, 24, 26)]
PD.testOmit <- PD.testOmit[, c(1, 8, 12, 14, 17, 18, 22, 23, 24, 26)]

#see if variables can be removed with cross validation
rfcv(PD.trainOmit[, -10], PD.trainOmit$Class, cv.fold=3, step=0.9)

#remove 2 least important variables
PD.trainOmit <- PD.trainOmit[, -c(4, 5)]
PD.testOmit <- PD.testOmit[, -c(4, 5)]

#get error by numtrees plot
set.seed(32471033)
pdTreeError <- randomForest(Class~., data = PD.trainOmit, ntree = 1000)

plot(pdTreeError)

#get best mtry val
set.seed(32471033)
tuneRF(PD.trainOmit[, -8], PD.trainOmit$Class, mtryStart = 2, ntreeTry = 300, stepFactor = 1.5, plot = TRUE,
trace = TRUE, improve = 0.01)

#create model
set.seed(32471033)
pdBest <- randomForest(Class~., data = PD.trainOmit, ntree = 300, mtry = 2, sampsize = c('0' = 300, '1' =
160), replace = TRUE)

#get AUC
pdBestPredict <- predict(pdBest, PD.testOmit, type = 'prob')[,2] %>%
  ROCR::prediction(., PD.testOmit$Class)
pdBestPerf <- performance(pdBestPredict, "tpr", "fpr")
performance(pdBestPredict, "auc")@y.values

#plot ROC curve
plot(pdBestPerf, main = "ROC Curve of Best Classifier", col = "blue")

#confusion table
pdBestPredict <- predict(pdBest, PD.testOmit)
table(predicted = pdBestPredict, actual = PD.testOmit$Class)

#q11
#remove missing vals
nn.train <- na.omit(PD.train)
nn.test <- na.omit(PD.test)

#scale data while keeping real Class vals
nn.train <- as.data.frame(scale(nn.train[1:25]))
```

```
nn.train$Class <- (na.omit(PD.train))$Class

nn.test <- as.data.frame(scale(nn.test[1:25]))
nn.test$Class <- (na.omit(PD.test))$Class

#create neural network
set.seed(32471033)
pd.nn <- neuralnet(Class~A01 + A08 + A12 + A14 + A17 + A18 + A22 + A23 + A24, nn.train, hidden = c(6),
stepmax = 1e+6, lifesign = "full")

#plot neural network
plot(pd.nn)

#prediction + prediction processing
nn.pred <- compute(pd.nn, nn.test)
nn.predr <- round(nn.pred$net.result[, 2], 0)
nn.predrdf <- as.data.frame(as.table(nn.predr))

#ROC processing
nnPredict <- nn.pred$net.result[, 2] %>%
  ROCR::prediction(., nn.test$Class)
nnPerf <- performance(nnPredict, "tpr", "fpr")

#plot ROC curve
plot(pdDTPerf, main = "ROC Curve", col = "blue")
plot(pdNBPerf, col = "red", add = TRUE)
plot(pdBagPerf, col = "green", add = TRUE)
plot(pdBoostPerf, col = "pink", add = TRUE)
plot(pdRFPerf, col = "yellow", add = TRUE)
plot(nnPerf, col = "purple", add = TRUE)
lines(c(0, 1), c(0, 1), col = "black", lty = 2)
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest",
"Neural Network"), col = c("blue", "red", "green", "pink", "yellow", "purple"), lwd = 2)

#get AUC
performance(nnPredict, "auc")@y.values

#confusion matrix
table(predicted = nn.predrdf$Freq, actual = nn.test$Class)

#q12
#support vector classification
#train model
svc <- svm(Class ~., data = PD.train, na.action = na.omit)

#get predictions
svcPred <- predict(svc, PD.test)

#confusion table
table(predicted = svcPred, actual = na.omit(PD.test)$Class)
```