MONASH University

FIT3152 Assignment 2

Semester 1 - 2024

Ke Er Ang | 32581343

Task 1: Data Exploration

After a preliminary review, the dataset consists of 2,000 entries and 26 columns, consisting of 25 numerical attributes and one classification variable. It is noteworthy that the attributes vary widely in range, with some between 0 and 1, some between 0 and 100, A12 exceeding 100. Regarding missing values, only A01 and Class attributes are complete, while the others contain numerous missing entries. The Class variable categorises websites into two groups which are phishing and legitimate. Of the total dataset, approximately 32.90% of the sites are classified as phishing, while 67.10% are considered legitimate, resulting in a phishing to legitimate ratio of approximately 0.4902.

For real-valued predictor variables, their descriptive statistics have been found via the summary function in the Skimr library and the table below is the result obtained.

Variables	mean	sd	P0 (min)	p25 (Q1)	p50(median)	p75(Q3)	p100(max)
A01	22.4	15.9	3	5	22	27	48
A02	0.178	1.20	0	0	0	0	32
A03	0.00391	0.0624	0	0	0	0	1
A04	2.78	0.563	2	2	3	3	8
A05	0.0221	0.465	0	0	0	0	15
A06	0.132	0.339	0	0	0	0	1
A07	0.00195	0.0442	0	0	0	0	1
A08	0.843	0.217	0.159	0.667	1	1	1
A09	0.0241	0.153	0	0	0	0	1
A10	0.0352	0.184	0	0	0	0	1
A11	0.0638	0.538	0	0	0	0	10
A12	320.	145.	3	232	232	449	695
A13	0.0254	0.689	0	0	0	0	24
A14	0.127	0.333	0	0	0	0	1
A15	0.121	0.326	0	0	0	0	1
A16	0.0547	0.228	0	0	0	0	1
A17	1.16	0.582	0	1	1	1	5
A18	60.6	126.	4	13	33	89	3738
A19	0.0912	0.288	0	0	0	0	1
A20	0.236	0.425	0	0	0	0	1
A21	0.0345	0.233	0	0	0	0	3
A22	0.0557	0.0108	0.00962	0.0507	0.0580	0.0628	0.0851
A23	75.6	91.6	0	13	100	106	1754

A24	0.262	0.251	0	0.00641	0.0800	0.523	0.523
A25	0.000216	0.00572	0	0	0	0	0.197
Class	0.329	0.470	0	0	0	1	1

From this descriptive statistic table, we can tell that the attributes A03, A07, A25 have low mean and standard deviation, they might not be able to provide valuable information to the mode. Therefore, they are omitted from the datasets.

Besides, a correlation heatmap has been generated (refer to FIgure 1.1). In this heatmap, blue shades indicate the variables have positive correlation, yellow shades indicate the variables have negative correlation, the darker the colour, the stronger the correlation. And white shades indicate there is no correlation between the variables. The heatmap reveals that A13 and A07 are strongly positively correlated with A25. Additionally, A11 and A21 also demonstrate a strong positive correlation. On the other hand, A12 and A24, as well as A08 and A17, both exhibit strong negative correlations.

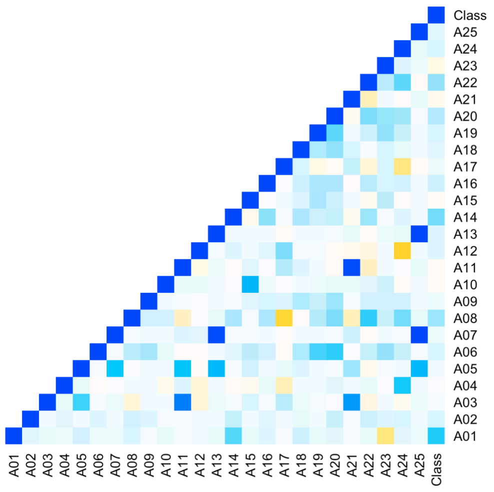


Figure 1.1 Heatmap

Task 2: Data Preprocessing

In order to make the data set suitable for analysis and model fitting, the missing values have been dropped to prevent incorrect and unwanted results from a biased model. Besides, data has been scaled to address the varying ranges issue of the data, this is done to ensure all attributes contribute equally to the models and prevent higher values attributes from predominating the models. Lastly, class attribute is factorised to ensure it is treated as categorical attribute in the analysis.

Task 3, Task 4: Implement Classification Models

After that, the data has been separated into 70% for the training set and 30% for the testing set. And different techniques have been used to implement the classification model, which include the Decision Tree, Naive Bayes, Bagging, Boosting and Random Forest.

Task 5: Confusion Matrix & Accuracy of The Models

A confusion matrix provides visualisation of the results of the classification models by making a tabular arrangement of the various predicted outcomes and actual outcomes. After that, by using the formula (TP + TN) / (TP + TN + FP + FN), the accuracy of the model is calculated.

The tables shown below are the confusion matrix of each classifier and the accuracy of the models.

Decision Tree		Predicted	
		0	1
Actual	0	305	7
	1	96	53
Accuracy: 0.7766			

Naive Bayes		Predicted	
		0	1
Actual	0	28	284
	1	12	137
Accuracy: 0.3579			

Bagging		Predicted	
		0	1
Actual	0	287	79
	1	25	70
Accuracy : 0.7744			

Boosting		Predicted	
		0	1
Actual	0	266	62
	1	46	87
Accuracy: 0.7657			

Random Forest		Predicted	
		0	1
Actual	0	274	38
	1	71	78
Accuracy: 0.7636			

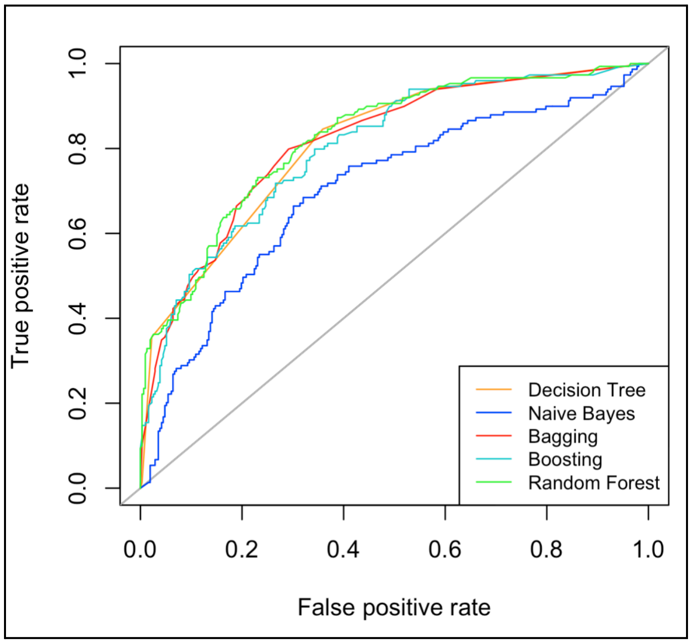
Based on the table above, accuracy wise, the Decision Tree model is the top classifier as it can predict 77.66% of data accurately, which is the highest out of all. On the other hand, Naive Bayes model demonstrates the worst performance in terms of accuracy, because it can only predict 35.79% of data correctly, which is not even half of the data. The Bagging and Boosting models have the similar predictive ability, they can predict 75.44% and 76.57% of data correctly respectively.

Task 6: ROC & AUC of The Models

Next, ROC curves are plotted for all models in a single graph for a deeper analysis of the performance of models. ROC curve tells us the performance of the model by plotting true positive rate against false negative rate at different classification thresholds.

Area under the curve of ROC curve tells us that an overall performance metric across all potential classification thresholds. The value of AUC ranging from 0 to 1, with lower value indicates worse performance of the model, higher value indicates better performance of the model. Figure 6.1 shows the ROC curves of all the models.

Figure 6.1 ROC



Task 7: Is there a single "best" classifier?

Below is a summary of the AUC and accuracy for each models.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Accuracy	0.7766	0.3579	0.7744	0.7657	0.7636
AUC	0.8114	0.7041	0.8194	0.8072	0.8230

The ranking of models (Accuracy): Decision tree, Bagging,Boosting, Random Forest, Naive Bayes
The ranking of models (AUC): Random Forest, Bagging, Decision Tree, Boosting, Naive Bayes

Based on the analysis, Decision Tree has the best accuracy out of all models but its AUC is not the best. This suggests although it does well in terms of accuracy, it is not the best in distinguishing classes. The performance of Naive Bayes is the poorest, it underperforms in both AUC and accuracy. Bagging has the second highest accuracy and AUC among all models. Although Boosting performance is not as good as Decision Tree, Bagging and Random Forest, it is still considered to have a high AUC and accuracy. Last, Random Forest has the best AUC showing that it performs the best in class distinguishing but its accuracy is slightly lower than Decision Tree, Bagging and Boosting.

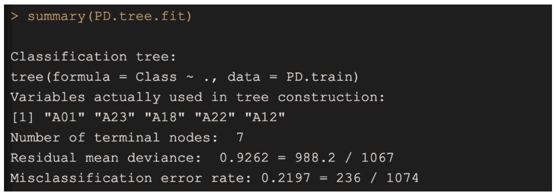
In conclusion, Decision Tree, Random Forest, Bagging and Boosting are all excellent classifiers. Based on performance metrics, there is no best classifier in this case. The best classifier selection might be affected by other factors such as training time and interpretability of the model.

Task 8: The Most Important Variables

After assessing the model performance, the next step is to investigate the most important variables in prediction for each model. However, since Naive Bayes produces probability values for each input attribute, it does not explicitly determine each variable's importance, it will not be examined for the most important variables.

For the Decision Tree model, there are 7 terminal nodes and the most important variables are A01, A23, A18, A22 and A12. The residual mean deviance is 0.926 and misclassification rate is 0.22 (see Figure 8.1)

Figure 8.1 Decision Tree Summary

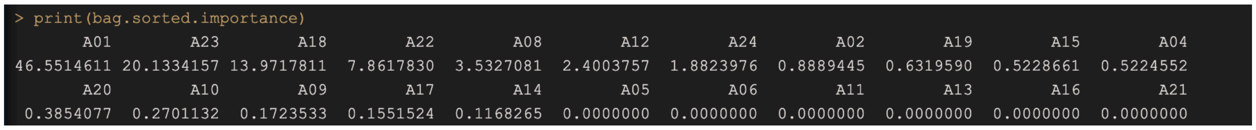


```
> summary(PD.tree.fit)

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A01" "A23" "A18" "A22" "A12"
Number of terminal nodes:  7
Residual mean deviance:  0.9262 = 988.2 / 1067
Misclassification error rate: 0.2197 = 236 / 1074
```

For the Bagging classifier, A01 is the most important variable, as the variable importance measure is significantly higher than the other variables. The second most important variable is A23, followed by the A18, A22 and A08. (see Figure 8.2)

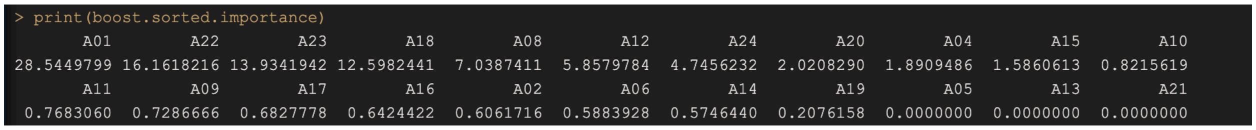
Figure 8.2 Variables Importance of Bagging



```
> print(bag.sorted.importance)
       A01        A23        A18        A22        A08        A12        A24        A02        A19        A15        A04 
46.5514611 20.1334157 13.9717811  7.8617830  3.5327081  2.4003757  1.8823976  0.8889445  0.6319590  0.5228661  0.5224552 
       A20        A10        A09        A17        A14        A05        A06        A11        A13        A16        A21 
 0.3854077  0.2701132  0.1723533  0.1551524  0.1168265  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000 
```

For the Boosting classifier, same as the previous model, A01 is the most important variable. However, in this model, A22 is the second most important, followed by A23, A18 and A08. (see Figure 8.3)

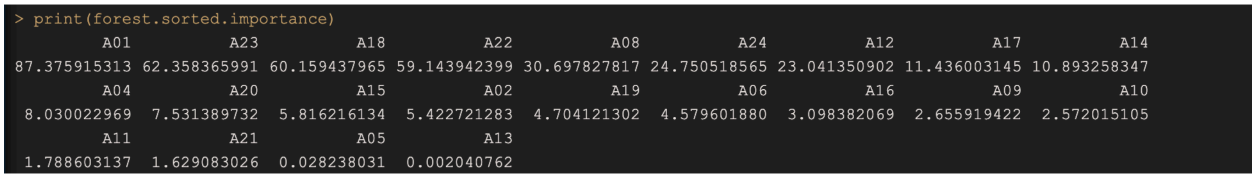
Figure 8.3 Variables Importance of Boosting



```
> print(boost.sorted.importance)
       A01        A22        A23        A18        A08        A12        A24        A20        A04        A15        A10 
28.5449799 16.1618216 13.9341942 12.5982441  7.0387411  5.8579784  4.7456232  2.0208290  1.8909486  1.5860613  0.8215619 
       A11        A09        A17        A16        A02        A06        A14        A19        A05        A13        A21 
 0.7683060  0.7286666  0.6827778  0.6424422  0.6061716  0.5883928  0.5746440  0.2076158  0.0000000  0.0000000  0.0000000 
```

For the Random Forest classifier, same as the previous model, the most important variable is also A01. The second most important variable is A23, followed by A18, A22 and A08. (see Figure 8.4)

Figure 8.4 Variables Importance of Random Forest



```
> print(forest.sorted.importance)
         A01          A23          A18          A22          A08          A24          A12          A17          A14 
87.375915313 62.358365991 60.159437965 59.143942399 30.697827817 24.750518565 23.041350902 11.436003145 10.893258347 
         A04          A20          A15          A02          A19          A06          A16          A09          A10 
 8.030022969  7.531389732  5.816216134  5.422721283  4.704121302  4.579601880  3.098382069  2.655919422  2.572015105 
         A11          A21          A05          A13 
 1.788603137  1.629083026  0.028238031  0.002040762 
```

From the analysis above, we can see that A01, A23, A18, A22, A12, A08 and A24 are common most important variables. The variables that can be omitted from the data include A21, A13, A05. First, they contributed nothing to the Bagging and Boosting model, excluding them will not change the models' ability in doing the classification. They are also the least important variables in the Random Forest model, which means their contribution to the predictive model is negligible. As they are not used in the Decision Tree model, they do not provide any value for the model to do classification. Thus, this suggests that the models' accuracy in distinguishing between phishing and legitimate websites will not be affected by the exclusion of these variables.

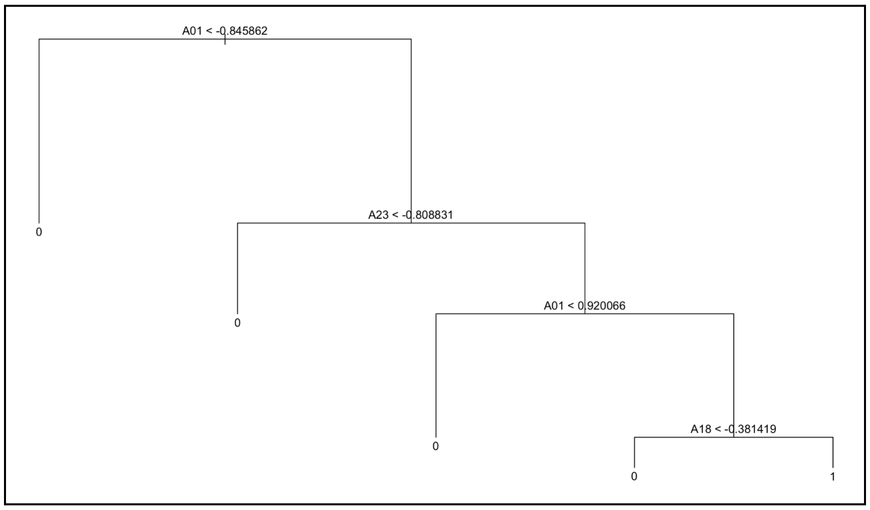
Below is a summary of the most important variables for each models.

	Decision Tree	Bagging	Boosting	Random Forest
Top 1	A01	A01	A01	A01
Top 2	A23	A23	A22	A23
Top 3	A18	A18	A23	A18
Top 4	A22	A22	A18	A22
Top 5	A12	A08	A08	A08
Top 6	-	A12	A12	A24
Top 7	-	A24	A24	A12
Top 8	-	A02	A20	A17

<u>Task 9: Simple Classifier for Manual Classification</u>

The Decision Tree classifier is chosen for this question. One of the factors that were taken into consideration when choosing this classifier is the Interpretability of the model. As the goal is to let a person easily use it for manual classification, the Decision Tree is chosen as it is direct and intuitive. The second factor considered is the attributes of the model. The attributes that have been included are the three variables that are considered the most important in the original Decision Tree classifier which is A01, A23, A18, as these attributes demonstrate a strong predictive ability. Figure 9.1 shows the simple Decision Tree created.

Figure 9.1 Simple Decision Tree



The simple Decision Tree model is 2 nodes less than the original model, it consists of only 5 terminal nodes. The root node is split with the attribute A01, it decides whether the attribute is less or more than -0.8459. If it is less than -0.8459, then it is classified as a legitimate site. The next split is based on A23, it checks whether it is larger or smaller than -0.8088. If it is larger than -0.8088, then it will lead to the next split which involves re-evaluating the attribute A01. Then, this process is continued until the last split, which also consists of the leaf node. The website will be labelled as a phishing site if every condition on the right side of the branches is satisfied. Otherwise, it is classified as a legitimate site.

After the simple tree is created, it is fitted on the test data. The accuracy of the model stays the same as 0.7766 and area under the curve has slightly improved to 0.8118. These two metrics indicate that this simple model still demonstrates good prediction and class distinguishing ability; the simpler tree does not reduce the performance. Compared to the other models created earlier, the Decision Tree still has the best accuracy and AUC is still considered high. Although the Bagging and Random Forest are still better than it, this tree is easy enough for a person to classify the sites by hand.

	Simple Decision Tree	Decision Tree (original)	Naive Bayes	Bagging	Boosting	Random Forest
Accuracy	0.7766	0.7766	0.3579	0.7744	0.7657	0.7636
AUC	0.8118	0.8114	0.7041	0.8194	0.8072	0.8230

Task 10: The Best Tree-based Classifier

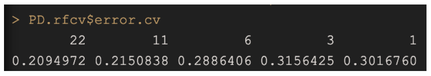
The **Bagging classifier** with specific parameters has been chosen as the best classifier after performing an evaluation procedure using cross-validation and parameter tuning.

To develop the best tree-based classifier, the accuracy and AUC of the models from Questions 4, 5, and 6 were taken into consideration. Initially, the Decision Tree has the highest accuracy whereas the Random Forest has the largest AUC ;Bagging and Boosting both have competitive measures. Thus, cross-validation has been employed to evaluate performance of all the classifiers and by tuning the hyperparameters to build the best model. Besides, the selection of attributes included was based on their predictive performance and significance in prior models. Only the attributes that have importance score more than 1 are included in the model creation, this is to ensure that only the most important attributes are contributed to the model.

Random Forest

For Random Forest, the cross-validation procedure began by using the "rfcv" function to optimise the number of predictors "mtry", in the model the results are recorded in the "error.cv" column and a smaller number of errors indicate better results. The performance of the model is optimised by adjusting the number of trees used (mtry). Figure 10.1 shows the result reported the Random Forest model has the least error when mtry=22, with this parameter, the accuracy of the model has improved to 0.7505. Interestingly, when mtry=12, the accuracy of the model improves to 0.7636 and AUC improves to 0.8231. This issue will be caused by the model overfitting of data, as mtry=22 indicates that almost all variables are considered in the splits, hence, the pattern is less likely to be captured by each tree.

Figure 10.1 error.cv



```
> PD.rfcv$error.cv
       22        11         6         3         1 
0.2094972 0.2150838 0.2886406 0.3156425 0.3016760 
```

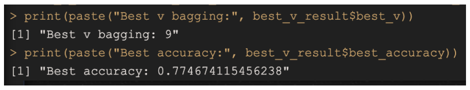
Decision Tree

For the Decision Tree, the function, “cv.tree” has been applied to find optimised tree size. The result shows that size 5 yielded the lowest number of deviance. Although the tuned tree has less variables used, the misclassification rate has slightly gone up to 0.2263. Interestingly, the accuracy of the tuned model stays the same and the area under the curve has gone up to 0.8118.

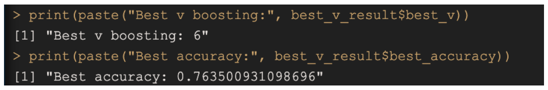
Bagging & Boosting

Besides, cross validation has also been applied on Bagging and Boosting. The models are evaluated with folds ranging from 5 to 10. Boosting has highest accuracy of 0.7747 when the fold is 6 and Bagging has highest accuracy of 0.7635 when fold is 9 (see Figure 10.3). Subsequently, the parameter ‘mfinal’ (number of trees) of both models are adjusted to achieve optimal model performance. After tuning, the bagging model achieved the highest accuracy and AUC out of all tuned classifiers , with an accuracy of 0.7896 and AUC of 0.8339. (see Figure 10.4)

Figure 10.3 Best fold of Bagging & Boosting

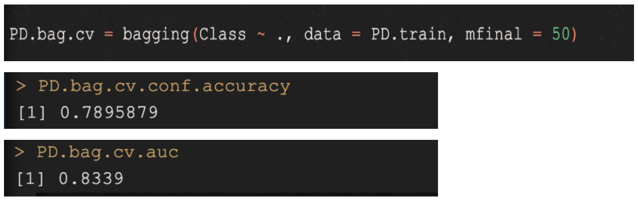


```
> print(paste("Best v bagging:", best_v_result$best_v))
[1] "Best v bagging: 9"
> print(paste("Best accuracy:", best_v_result$best_accuracy))
[1] "Best accuracy: 0.774674115456238"
```



```
> print(paste("Best v boosting:", best_v_result$best_v))
[1] "Best v boosting: 6"
> print(paste("Best accuracy:", best_v_result$best_accuracy))
[1] "Best accuracy: 0.763500931098696"
```

Figure 10.4 Code snippet of tuned Bagging model and its accuracy and AUC



```
PD.bag.cv = bagging(Class ~ ., data = PD.train, mfinal = 50)
```

```
> PD.bag.cv.conf.accuracy
[1] 0.7895879
```

```
> PD.bag.cv.auc
[1] 0.8339
```

Table below shows the summary of the model performances before and after tuning:

	Accuracy (before)	Accuracy (after)	AUC (before)	AUC (after)
Decision Tree	0.7766	0.7766	0.8114	0.8118
Bagging	0.7744	0.7896	0.8194	0.8339
Boosting	0.7657	0.7570	0.8072	0.7867
Random Forest	0.7636	0.7636	0.8230	0.8231

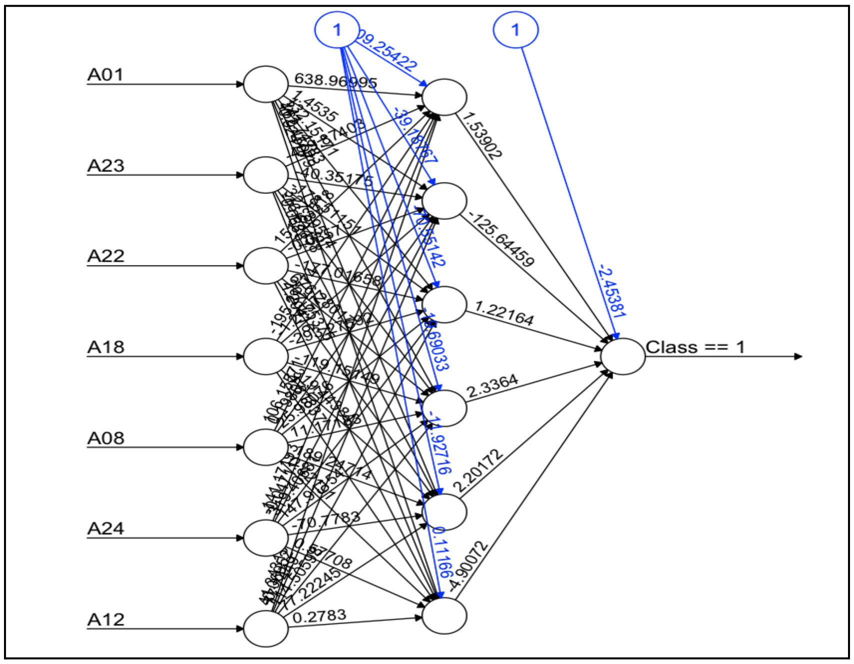
After tuning, the accuracy of Bagging has been significantly improved from 0.7744 to 0.7896, and its AUC has increased to 0.8339. The accuracy of Boosting has dropped from 0.7657 to 0.7570 and the AUC has dropped to 0.7867. The accuracy and AUC of Random Forest has not changed much, with accuracy staying 0.7636 and AUC slightly increased to 0.8231. The accuracy of the Decision Tree has not been improved as not improved as well, but its AUC slightly increases to 0.8118.

Based on the analysis, the tuned Bagging model outperformed all models, as evidenced by its greatest AUC and accuracy after tuning. Also it has a high accuracy of 0.7747 with 9-folds cross validation. Hence, Bagging model, particularly with parameter “mfinal=50” has been chosen as the best classifier. This analysis shows how effective and robust the Bagging model is for this particular datasets. The Bagging model aggregates predictions by taking majority votes from all trained trees of different data subsets, which lowers variance and increases generalisation. Besides, it also has the ability to manage dataset complexity without significant overfitting and has a balanced performance across several metrics.

Task 11: Artificial Neural Network

Before fitting the ANN model, the data is preprocessed by scaling the data to a similar scale. This is done to ensure that no feature with higher values is dominating the model. Next, the data is split into 70% training and 30% testing dataset, with all the attributes confirmed as numeric beforehand.
The ANN model created consists of 3 layers which are input layer, hidden layer and output layer. The input layer created by the top 7 most important attributes of all models identified in Question 8; the hidden layer consists of 6 neurons to capture intricate patterns in the dataset without overfitting while considering the interpretability of the model and appropriate processing load (see Figure 11.1). The parameter "linear.output = 1" makes the output of the model a number between 0 and 1. Figure 11.1 shows the ANN model implemented.

Figure 11.1 Artificial Neural Network



The accuracy of the ANN model is 0.7396 and the AUC is 0.6727. While the AUC that is larger than 0.5 indicates using the model is better than random guessing, it is not considered to have strong performance. The accuracy of the ANN model is one the lowest out of all models, it is only doing better than Naive Bayes. AUC wise, the performance of this ANN model lags behind the other tuned tree-based classifiers. The reasons for the poor performance might be due to the ANN model underfitting the data, the chosen architecture, such as the number of hidden layers or the other chosen parameters might not be optimal. In order to obtain a better ANN model, further tuning the hyperparameters or changing the architecture might be able to help.

Table below shows the summary of the model performances:

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	ANN
Accuracy	0.7766	0.3579	0.7896	0.7570	0.7636	0.7396
AUC	0.8118	0.7041	0.8339	0.7867	0.8231	0.6727

Task 12: New Classifier - XGBoost

Package link:

https://cran.r-project.org/web/packages/xgboost/xgboost.pdf

https://xgboost.readthedocs.io/en/stable/R-package/xgboostPresentation.html

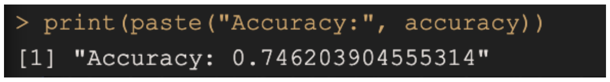
The new classifier chosen is Extreme Gradient Boosting, which is also known as XGBoost Classifier. XGBoost is an algorithm that falls under the group of ensemble learning. Due to its good performance for computational efficiency, feature importance analysis, and capability of handling missing value, making it as the popular choice for regression, classification, and ranking.

The way the algorithm works is by building an initial tree. Then, continue to build the decision trees one at a time. The tree built uses gradient boosting technique to modify the error of the previous tree and therefore minimise prediction error in the next tree. The final prediction is made by combining all the predictions made from all the trees, providing an exceptional accuracy.

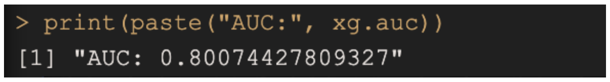
To begin the model creation, the data is preprocessed by dividing it into 70% training and 30% testing sets. After that, each of these sets is converted into an 'xgb.DMatrix' object adhering to the XGBoost specifications. For both datasets, a label set is also generated which is used to do calculation for performance metrics for the model. To make sure the labels begin at 0, the Class attribute is transformed to a numeric type, and 1 is then subtracted. The "binary:logistic" in parameter indicates that the model does logistic regression for binary classification and output is probabilities.

After fitting the model on the test data, the accuracy of the model is 0.7462, meaning the predictions made by the model are about 75% correct, shows that the model is fairly effective in doing predictions and has a decent level of predictive power. The AUC of this model is 0.8007, which shows the model has a strong performance in class distinguishing. The high AUC and accuracy indicates that the ANN model has good prediction ability and outstanding class distinguishing capability. (see Figure 12.1)

Figure 12.1 Accuracy and AUC of XGBoost



```
> print(paste("Accuracy:", accuracy))
[1] "Accuracy: 0.746203904555314"
```



```
> print(paste("AUC:", xg.auc))
[1] "AUC: 0.80074427809327"
```

With an accuracy of 0.7462 and an AUC of 0.8007, the XGBoost model demonstrates strong classification ability, however it still falls short of the Random Forest and Bagging models. Better performance metrics are demonstrated by the Random Forest and Bagging models, in particular, with Bagging and Random Forest having the highest and second highest AUC respectively. This might be explained by the size of data or XGBoost's sensitivity to hyperparameter settings, which will require more precise tuning for the best result.

Table below shows the summary of the model performances:

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	ANN	XGBoost
Accuracy	0.7766	0.3579	0.7896	0.7570	0.7636	0.7396	0.7462
AUC	0.8118	0.7041	0.8339	0.7867	0.8231	0.6727	0.8007

```
# ------------------------------------------------
# Assignment 2
# Name: Ke Er Ang
# Student ID: 32581343
#------------------------------------------------
library(dplyr)
library(skimr)
library(tree)
library(e1071)
library(adabag)
library(randomForest)
library(ROCR)
library(ggplot2)
library(xgboost)
library(ipred)
library(pROC)
library(neuralnet)

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32581343) # 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

View(PD)

'--------------------- Question 1 ----------------------'
# Check occurrences of NA values in all columns
colSums(is.na(PD))
# Drop NA
PD = na.omit(PD) # left 1535 rows

View(PD)
str(PD)
attach(PD)

# What is the proportion of phishing sites to legitimate sites? legitimate =0 , phishing = 1
props1 = sum(PD$Class == 1) # phishing
print(props1) # 505
propotion.props1 = props1/nrow(PD)*100
print(propotion.props1)# 32.89902

props0 = sum(PD$Class == 0) # legitimate
print(props0) # 1030
propotion.props0 = props0/nrow(PD)*100
print(propotion.props0) # 67.10098

proportion = props1/props0 # 0.490291262135922
print(paste("Proportion:",proportion)) # 0.49

# correlation
cor = cor(PD)
my_palette = colorRampPalette(c("#FFD700","#FFE455", "#FFF1AA",
"#FFFFFF","#AAE9FF","#54D4FF","#00BFFF","#007FFF","#003FFF","#0000FF"))(n=299)
cor[upper.tri(cor)] =  NA
heatmap(cor,Rowv = NA, Colv = NA, symm = TRUE, col = my_palette)

# descriptive statistic
skim(PD) # before scaled

'---------------------Question 2---------------------'
# scale the data
PD[,1:25] = scale(PD[,1:25])
# factorize class attribute
PD$Class = factor(PD$Class)
# drop attributes
PD$A03 = NULL
PD$A07 = NULL
PD$A25 = NULL

'---------------------Question 3---------------------'
# Divide your data into a 70% training and 30% test set
```

```r
set.seed(32581343) #Student ID as random seed
train.row = sample(1:nrow(PD), 0.7*nrow(PD))
PD.train = PD[train.row,]
PD.test = PD[-train.row,]
'---------------------Question 4----------------------'
# Decision tree
DTree.fit = tree(Class~., data = PD.train)
# summary(DTree.fit) # Misclassification error rate: 0.2197 = 236 / 1074
DTree.predict = predict(DTree.fit,PD.test, type = "class")
plot(DTree.fit)
text(DTree.fit)
DTree.conf = table(actual = PD.test$Class, predicted = DTree.predict)
DTree.conf

# Naive Bayes
set.seed(32581343)
NB.fit = naiveBayes(Class ~. ,data = PD.train)
NB.predict = predict(NB.fit, PD.test, type = "class")
NB.conf = table(actual = PD.test$Class, predicted =NB.predict)
NB.conf

# Bagging
set.seed(32581343)
Bagging.fit = bagging(Class ~. ,data = PD.train, mfinal = 15)
Bagging.predict = predict.bagging(Bagging.fit, PD.test, type = "class")
Bagging.conf = Bagging.predict$confusion
Bagging.conf

# Boosting
set.seed(32581343)
Boosting.fit = boosting(Class ~. ,data = PD.train, mfinal = 10)
Boosting.predict = predict.boosting(Boosting.fit, newdata = PD.test, type = "class")
Boosting.conf = Boosting.predict$confusion
Boosting.conf

# Random Forest
set.seed(32581343)
RForest.fit = randomForest(Class ~. ,data = PD.train,na.action = na.exclude)
RForest.predict = predict(RForest.fit, PD.test, type = "class")
RForest.conf = table(actual = PD.test$Class, predicted = RForest.predict)
RForest.conf

'---------------------Question 5----------------------'
DTree.accuracy = (sum(diag(DTree.conf))/sum(DTree.conf))
DTree.accuracy

NB.accuracy = (sum(diag(NB.conf))/sum(NB.conf))
NB.accuracy

Bagging.accuracy = (sum(diag(Bagging.conf))/sum(Bagging.conf))
Bagging.accuracy

Boosting.accuracy = (sum(diag(Boosting.conf))/sum(Boosting.conf))
Boosting.accuracy

RForest.accuracy = (sum(diag(RForest.conf))/sum(RForest.conf))
RForest.accuracy

'---------------------Question 6----------------------'
# Create ROC for all models
# Tree
DTree.conf = predict(DTree.fit,PD.test, type = "vector")
DTree.conf.pred = ROCR::prediction(DTree.conf[,2],PD.test$Class)
DTree.conf.pred.perf = ROCR::performance(DTree.conf.pred, "tpr", "fpr")
plot(DTree.conf.pred.perf, col = "orange", loc = "bottomleft")
abline(0,1, col = "grey")

# Naive Bayes
NB.conf = predict(NB.fit, PD.test, type = "raw")
NB.conf.pred = ROCR::prediction(NB.conf[,2],PD.test$Class)
NB.conf.pred.perf = ROCR::performance(NB.conf.pred, "tpr", "fpr")
plot(NB.conf.pred.perf, col = "blue", add= TRUE)

# Bagging
```

```r
Bagging.conf.pred = ROCR::prediction(Bagging.predict$prob[,2],PD.test$Class)
Bagging.conf.pred.perf = ROCR::performance(Bagging.conf.pred, "tpr", "fpr")
plot(Bagging.conf.pred.perf, col = "red", add= TRUE)

# Boosting
Boosting.conf.pred = ROCR::prediction(Boosting.predict$prob[,2],PD.test$Class)
Boosting.conf.pred.perf = ROCR::performance(Boosting.conf.pred, "tpr", "fpr")
plot(Boosting.conf.pred.perf, col = "darkturquoise", add= TRUE)

# Random Forest
RForest.conf = predict(RForest.fit, PD.test, type="prob")
RForest.conf.pred = ROCR::prediction(RForest.conf[,2],PD.test$Class)
RForest.conf.pred.perf = ROCR::performance(RForest.conf.pred, "tpr", "fpr")
plot(RForest.conf.pred.perf, col = "green", add= TRUE)

# Add a legend
legend("bottomright",
       legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
       col = c("orange", "blue", "red", "darkturquoise", "green"),
       lty = 1,  # Line type
       cex = 0.8)


# Create AUC for all models
# Decision Tree
DTree.auc.perf = performance(DTree.conf.pred, "auc")
DTree.auc = round(as.numeric(DTree.auc.perf@y.values), 4)
DTree.auc

# Naive Bayes
NB.auc.perf = performance(NB.conf.pred, "auc")
NB.auc = round(as.numeric(NB.auc.perf@y.values), 4)
NB.auc

# Bagging
Bagging.auc.perf = performance(Bagging.conf.pred, "auc")
Bagging.auc = round(as.numeric(Bagging.auc.perf@y.values), 4)
Bagging.auc

# Boosting
Boosting.auc.perf = performance(Boosting.conf.pred, "auc")
Boosting.auc = round(as.numeric(Boosting.auc.perf@y.values), 4)
Boosting.auc

# Random Forest
RForest.auc.perf = performance(RForest.conf.pred, "auc")
RForest.auc = round(as.numeric(RForest.auc.perf@y.values), 4)
RForest.auc

'---------------------Question 8----------------------'
# Find the most important attributes
# Tree
summary(DTree.fit)

# Bagging
bag.sorted.importance = sort(Bagging.fit$importance,decreasing = T)
print(bag.sorted.importance)

# Boosting
boost.sorted.importance = sort(Boosting.fit$importance,decreasing = T)
print(boost.sorted.importance)

# Random Forest
forest.sorted.importance = RForest.fit$importance[order(-RForest.fit$importance),]
print(forest.sorted.importance)

'-------------------- Question 9  ------------------------'
# Create a smaller tree with lesser attributes
set.seed(32581343)
View(PD)
# Extract only the used variables in original Decision Tree
DTree.small = PD[,c('A01','A23','A18','A22','A12','Class')]

# Fit the model and plot the tree
```

```
DTree.small.fit = tree(Class~A01+A23+A18, data = PD.train)
plot(DTree.small.fit)
text(DTree.small.fit, pretty = 0)
summary(DTree.small.fit)

# Do prediction with test data
DTree.small.fit.predict = predict(DTree.small.fit, PD.test, type = 'class')

# Calculate accuracy
DTree.small.conf = table(actual = PD.test$Class, predicted = DTree.small.fit.predict)
DTree.small.accuracy = (sum(diag(DTree.small.conf ))/sum(DTree.small.conf ))
DTree.small.accuracy

# Calculate AUC
DTree.small.conf = predict(DTree.small.fit, PD.test, type = 'vector')
DTree.small.conf.pred = ROCR::prediction(DTree.small.conf[,2],PD.test$Class)
DTree.small.auc.perf = ROCR::performance(DTree.small.conf.pred, "auc")
DTree.small.auc = round(as.numeric(DTree.small.auc.perf@y.values), 4)
DTree.small.auc

'-------------------- Question 10 -----------------------'
# Find best tree by doing cross validation

#### CV tree
set.seed(32581343)
DTree.cv = cv.tree(DTree.fit, FUN= prune.misclass)
DTree.cv
# prune using size 5
DTree.cv.prune = prune.misclass(DTree.fit, best = 5)
summary(DTree.cv.prune)
plot(DTree.cv.prune)
text(DTree.cv.prune, pretty =0)

# check accuracy using the pruned decision tree
DTree.cv.prune.predict = predict(DTree.cv.prune, PD.test, type = 'class')
DTree.cv.prune.conf = table(actual = PD.test$Class, predicted = DTree.cv.prune.predict)
DTree.cv.prune.accuracy = sum(diag(DTree.cv.prune.conf)/sum(DTree.cv.prune.conf))
DTree.cv.prune.accuracy

# CV tree AUC
set.seed(32581343)
DTree.cv.conf = predict(DTree.cv.prune,PD.test, type = "vector")
DTree.cv.conf.pred = ROCR::prediction(DTree.cv.conf[,2],PD.test$Class)
DTree.cv.auc.perf = ROCR::performance(DTree.cv.conf.pred, "auc")
DTree.cve.auc = round(as.numeric(DTree.cv.auc.perf@y.values), 4)
DTree.cve.auc

#### CV forest
set.seed(32581343)
# cross validation for
PD.rfcv = rfcv(trainx=PD.train[,-c(23)], # training attributes (all columns except Class)
               trainy=PD.train[,c(23)],  # target attributes (Class)
               cv.fold=10,step=0.5,      # 10-folds cross validation, number of features reduced
at step size of 0.5
               mtry=function(p) max(1,floor(sqrt(p))), # number of features to try at each split,
min is 1
               recursive=TRUE) # remove the least important features at each steps and build the
model again
PD.rfcv$error.cv


# check accuracy using the pruned Random Forest
set.seed(32581343)
RForest.cv = randomForest(Class~., data = PD.train, mtry = 12)
RForest.cv.pred = predict(RForest.cv, PD.test)
RForest.cv.conf = table(actual = PD.test$Class, predicted = RForest.cv.pred )
RForest.cv.accuracy = sum(diag(RForest.cv.conf)/sum(RForest.cv.conf))
RForest.cv.accuracy

# CV forest AUC
set.seed(32581343)
RForest.cv.conf = predict(RForest.cv, PD.test, type="prob")
RForest.cv.conf.pred = ROCR::prediction(RForest.cv.conf[,2],PD.test$Class)
RForest.cv.auc.perf = ROCR::performance(RForest.cv.conf.pred, "auc")
```

```r
RForest.cv.auc = round(as.numeric(RForest.cv.auc.perf@y.values), 4)
RForest.cv.auc

#### CV Bagging & Boosting
# Retrieve attributes' name which have importance score > 0
bag.important.attribute = names(bag.sorted.importance[bag.sorted.importance> 0])
boost.important.attribute = names(boost.sorted.importance[boost.sorted.importance> 0])

# Find best number of folds for Bagging and Boosting
find.best.v = function(data, model.type, important.attributes ) {
  best.v = 0
  best.accuracy = 0

  for (v in 5:10) {
    set.seed(32581343)
    formula.string = paste(" Class ~", paste(important.attributes , collapse = " + "))

    if (model.type == "bagging") {# CV for bagging
      cv.model = bagging.cv(as.formula(formula.string), v = v, data = data)
    } else if (model.type == "boosting") { # CV for boosting
      cv.model = boosting.cv(as.formula(formula.string), v = v, data = data)
    }

    # Find accuracy of each created model
    cv.conf = cv.model$confusion
    cv.accuracy = sum(diag(cv.conf)) / sum(cv.conf)

    # If the model's accuracy is better than best accuracy, then it will become the best accuracy
    if (cv.accuracy > best.accuracy) {
      best.accuracy = cv.accuracy
      best.v = v
    }
  }
  return(list(best.v = best.v, best.accuracy = best.accuracy))
}

PD.best.v.bag = find.best.v(PD.train,"bagging", bag.important.attribute)
print(paste("Best v bagging:",PD.best.v.bag$best.v))
print(paste("Best accuracy:", PD.best.v.bag$best.accuracy))

PD.best.v.boost = find.best.v(PD.train, "boosting", boost.important.attribute)
print(paste("Best v boosting:", PD.best.v.boost$best.v))
print(paste("Best accuracy:", PD.best.v.boost$best.accuracy))

# Fit the tuned model for bagging
set.seed(32581343)
Bagging.cv = bagging(Class ~ ., data = PD.train, mfinal = 50)

# Do prediction on PD.test and find accuracy
Bagging.cv.predict = predict.bagging(Bagging.cv, PD.test, type = "class")
Bagging.cv.conf = Bagging.cv.predict$confusion
Bagging.cv.conf.accuracy = sum(diag(Bagging.cv.conf )) / sum(Bagging.cv.conf )
Bagging.cv.conf.accuracy

# Find AUC for bagging cv
Bagging.cv.conf.pred = ROCR::prediction(Bagging.cv.predict$prob[,2],PD.test$Class)
Bagging.cv.auc.perf = ROCR::performance(Bagging.cv.conf.pred, "auc")
Bagging.cv.auc = round(as.numeric(Bagging.cv.auc.perf@y.values), 4)
Bagging.cv.auc

# Fit the tuned model for boosting
set.seed(32581343)
Boosting.cv = boosting(Class ~ ., data = PD.train, mfinal=150)
# Do prediction on PD.test and find accuracy
Boosting.cv.predict = predict.boosting(Boosting.cv, PD.test, type = "class")
Boosting.cv.conf = Boosting.cv.predict $confusion
Boosting.cv.conf.accuracy = sum(diag(Boosting.cv.conf )) / sum(Boosting.cv.conf)
Boosting.cv.conf.accuracy

# Find AUC for boosting cv
Boosting.cv.conf.pred = ROCR::prediction(Boosting.cv.predict$prob[,2],PD.test$Class)
Boosting.cv.auc.perf = ROCR::performance(Boosting.cv.conf.pred, "auc")
Boosting.cv.auc = round(as.numeric(Boosting.cv.auc.perf@y.values), 4)
Boosting.cv.auc
```

```
'-------------------- Question 11 ------------------------'
set.seed(32581343)
# Fit ANN model, choose only top 8 most important attributes

ANN.fit = neuralnet(Class == 1 ~ A01 + A23 + A22 + A18 + A08 + A24 + A12, PD.train,
hidden=6,linear.output = FALSE)

ANN.pred = neuralnet::compute(ANN.fit, PD.test[,c("A01", "A23", "A22", "A18", "A08", "A24",
"A12")])
ANN.pred = as.data.frame(round(ANN.pred$net.result,0))

# Calculate ANN accuracy
ANN.conf = table(observed = PD.test$Class, predicted = ANN.pred$V1)
ANN.conf.accuracy = sum(diag(ANN.conf)/sum(ANN.conf))
ANN.conf.accuracy

# Calculate ANN AUC
ANN.conf.pred = ROCR::prediction(ANN.pred[,c(1)],PD.test$Class)
ANN.conf.pred.perf = ROCR::performance(ANN.conf.pred, "auc")
ANN.conf.pred.auc = round(as.numeric(ANN.conf.pred.perf@y.values), 4)
ANN.conf.pred.auc

# plot ANN diagram
plot(ANN.fit)

'-------------------- Question 12 ------------------------'
PD.xg = PD
set.seed(32581343)

# Transform Class attribute to numeric
label = PD.xg$Class
label = as.numeric(PD.xg$Class) -1
PD.xg$Class = NULL # drop Class attributes

# Split data to training & testing
xg.train.row = sample(1:nrow(PD.xg), 0.7*nrow(PD.xg))
PD.xg.train = PD.xg[xg.train.row,]
PD.xg.test = PD.xg[-xg.train.row,]

PD.train.data = as.matrix(PD.xg.train)
PD.train.label = label[xg.train.row] # create label for performance evaluation

PD.test.data = as.matrix(PD.xg.test)
PD.test.label = label[-xg.train.row]# create label for performance evaluation

# Transform train and test data into xgb.Matrix
xg.train = xgb.DMatrix(data = PD.train.data, label = PD.train.label)
xg.test = xgb.DMatrix(data = PD.test.data, label = PD.test.label)

# Define parameters
params = list(
  booster = "gbtree", # tree based model
  objective = "binary:logistic", # use logistic regression to do binary classification
  eval_metric = "auc"
)
? xgb.train
# Train the model, stop if no improvement after 10 rounds
PD.xg.fit = xgb.train(params = params, data = xg.train, nrounds = 100, watchlist = list(eval =
xg.test, train = xg.train), early_stopping_rounds = 10)
# Do prediction
PD.xg.pred = predict(PD.xg.fit,PD.test.data)

# Convert probabilities to binary class labels
PD.xg.pred.labels = ifelse(PD.xg.pred > 0.5, 1, 0)

# Create a confusion matrix
PD.xg.conf = table(Predicted = PD.xg.pred.labels, Actual = PD.test.label)
print(PD.xg.conf)

# Calculate accuracy
accuracy = sum(diag(PD.xg.conf)) / sum(PD.xg.conf)
print(paste("Accuracy:", accuracy))
```

```
# Calculate AUC
xg.pred = ROCR::prediction(PD.xg.pred,PD.test.label)
xg.perf = ROCR::performance(xg.pred, "auc")
xg.auc = xg.perf@y.values[[1]]
print(paste("AUC:", xg.auc))

# Plot one of the tree
install.packages("DiagrammeR")
model_trees = xgb.model.dt.tree(model = PD.xg.fit)
length(model_trees)
xgb.plot.tree(model = PD.xg.fit, trees = 0)
```