FIT 3152

Data Analytics

Assignment 2 Report *By Mohib Ali Khan 33370311*

Questions 1 - 7

Question 1

In a sample of 2000 rows, **phishing sites constitute 32.8%**, while **legitimate sites make up 67.2%** (figure A), indicating a higher prevalence of legitimate sites in the dataset. The descriptive statistics reveal that certain attributes exhibit high variability. For example, A12 has a mean of 315 with a standard deviation of 138.99, A18 has a mean of 60.7 with a standard deviation of 105.75, and A23 has a mean of 71.7 with a standard deviation of 61.88. These attributes suggest substantial differences across the observations. On the other hand, attributes such as A03, A07, and A25 have very low means and standard deviations, indicating that these features are mostly zero or near-zero. Additionally, the data shows significant amounts of missing values in certain attributes: A02, A08, A21, A22, and A24, with missing entries ranging from 20 to 31. These missing values could impact the analysis and need to be addressed. The overall variability and missing data patterns highlight potential areas of interest for further exploration and might influence the outcomes of any predictive models built using this data.

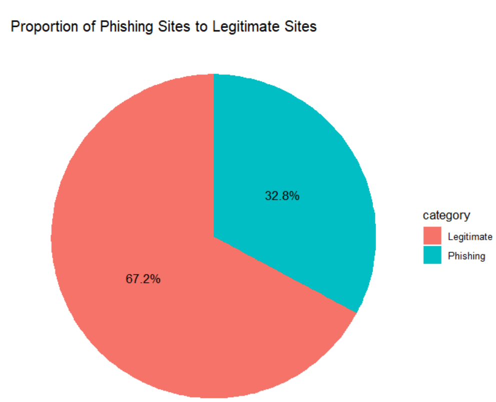

figure A

Question 2

To prepare the dataset for model fitting, addressing the missing values is crucial. Several attributes, including **A02, A08, A21, A22, and A24, have significant missing values.** We can handle these missing values through imputation methods such as replacing them with the mean, median, or mode of the respective attribute. Alternatively, if the proportion of missing values is very high and the attribute is not critical, we can consider removing those columns. For rows with missing values, if they are few in number, removing them might be a viable option.

Question 3 and Question 4

implemented with code (refer to appendix)

Question 5

Decision Tree Accuracy

Accuracy = 411+86 ⁄ 411+92 +11+86 = 497 ⁄ 600 ≈ 0.828

Naive Bayes Accuracy

Accuracy= 176 + 20/ 20+2+402+176 = 196/600 ≈ 0.327

Bagging Accuracy

Accuracy = 402+96 / 402+82+20+96 = 498/600 ≈ 0.830

Boosting Accuracy

Accuracy = 383 + 113 / 383 + 113 + 65 + 39 = 496/600 ≈ 0.827

Random Forest Accuracy

Accuracy = 305 + 93 / 305 + 93 + 48 + 24 = 398/470 ≈ 0.847

Question 6

Decision Tree AUC

AUC = 0.741

Naive Bayes AUC

AUC = 0.741

Bagging AUC

AUC = 0.782

Boosting AUC

AUC =0.838

Random Forest AUC

AUC = 0.848

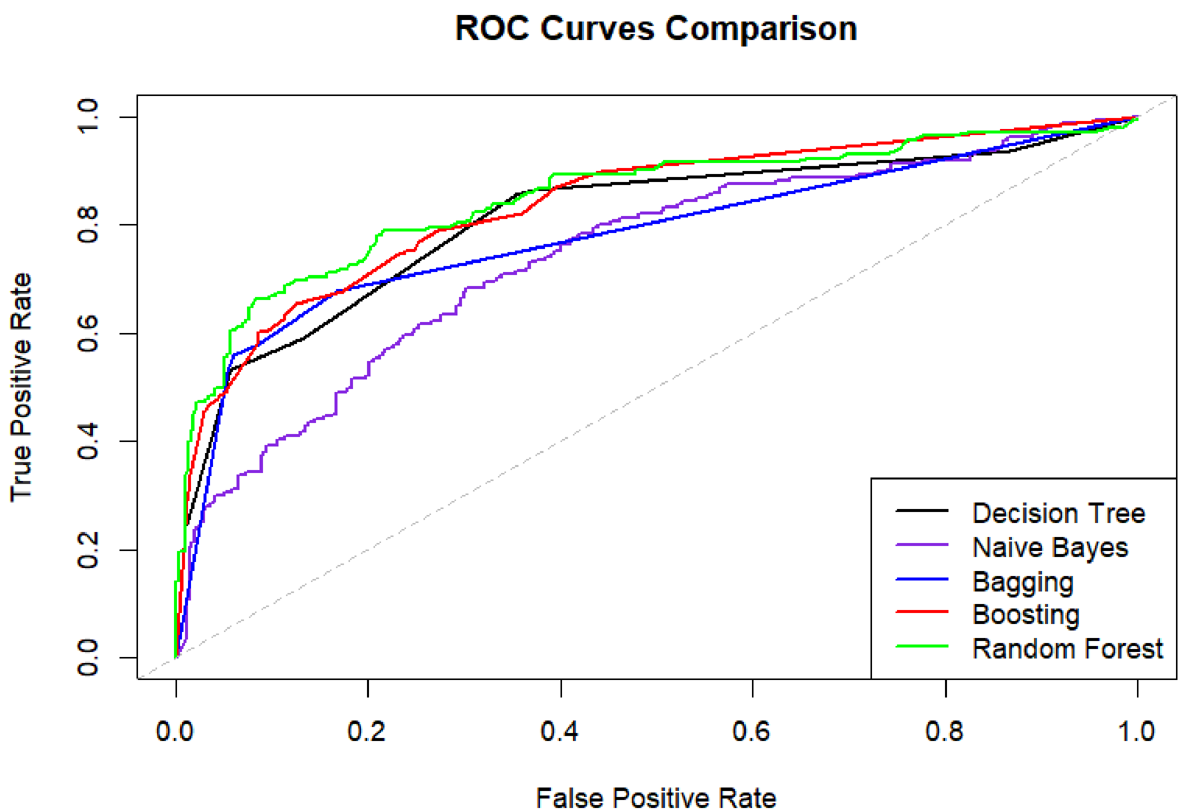
ROC Curves Comparison
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Decision Tree
Naive Bayes
Bagging
Boosting
Random Forest

figure B

Question 7

Model	***Accuracy***	***AUC***
Decision Tree	0.828	0.754
Naive Bayes	0.327	0.754
Bagging	0.830	0.836
Boosting	0.827	0.861
Random Forest	0.847	0.85

Based on the comparison of classifiers, **Random Forest** stands out as the best model with the highest accuracy (0.847) and AUC (0.85), indicating superior performance in both correct classification and discriminative power. Bagging and Boosting also show strong performance with high accuracy (0.830 and 0.827 respectively) and AUC values (0.836 and 0.861 respectively), but they are slightly lower than Random Forest. The Decision Tree has decent accuracy (0.828) but a lower AUC (0.754), while Naive Bayes performs poorly in accuracy (0.327) but shares the same AUC as the Decision Tree. Overall, Random Forest is the most effective classifier among the options considered

Investigative Tasks

Question 8

Decision Tree model

The Decision Tree model constructed to differentiate phishing from legitimate websites relies significantly on four primary variables: A01, A23, A18, and A22. These variables are pivotal in the tree's branching decisions, indicating their strong discriminative power in the context of the classification task. The tree culminates in 8 terminal nodes, showing a balanced complexity that neither underfits nor overfits, considering the total sample size. The residual mean deviance of 0.836 (904 residuals on 1080 degrees of freedom) reflects a good fit, suggesting the model captures a significant portion of the variance within the data without being overly complex. Moreover, a misclassification error rate of 0.178 illustrates that while the model performs well, there is still room for improvement, possibly by tuning hyperparameters, incorporating more data, or including interaction effects between the variables if not already considered. The reliance on these four variables and the performance metrics underscore their relevance in detecting phishing websites, suggesting that any reduction in the dataset should preserve these features to maintain the effectiveness of the model.

Bagging model

In the bagging model, the most important features are A01 (47.386), A23 (34.998), A22 (7.502), and A18 (5.645), which are critical for predicting phishing websites, aligning closely with the boosting model's key predictors. The boosting model also highlights A01 (52.101), A23 (16.929), A22 (13.885), and A18 (9.658) as the most significant variables. Both models suggest that features such as A03, A05, and A07 have negligible importance, implying that their removal would have minimal impact on the models' performance. This consistency across the boosting and bagging models underscores the robustness and relevance of the primary predictors, while also indicating the potential for dataset simplification by omitting the least important features.

<u>Boosting model</u>
In the boosting model, the most important features are A01 (52.101), A23 (16.929), A22 (13.885), and A18 (9.658), indicating these variables are crucial for predicting phishing websites. Less important variables include A12 (2.154), A14 (1.146), and A15 (0.889), with many others having negligible importance. Similarly, the bagging model, analyzed using the randomForest package, identifies A01, A23, A22, and A18 as highly influential, consistent with the boosting model's findings. A combined feature importance graph would show these variables with significantly higher importance scores, while features like A03, A05, and A07 would have near-zero impact, suggesting their removal would have minimal effect on model performance. This consistency across models highlights the robustness of the key predictors and underscores the potential for simplifying the dataset by omitting the least important features.

<u>Random Forest model</u>
The Random Forest model identifies A01 (101.0), A23 (72.0), A22 (60.7), and A18 (56.4) as the most important features for predicting phishing sites, indicating these variables are crucial for the model’s performance. Features like A08 (24.2), A14 (23.7), and A24 (23.2) also contribute significantly. In contrast, variables A05 (0.027), A13 (0.016), A07 (0.007), A03 (0.004), and A25 (0.003) have minimal impact, suggesting their removal would not significantly affect model accuracy. Simplifying the model by removing these least important features can reduce computational costs and improve interpretability without sacrificing predictive performance, as confirmed by validation tests comparing model accuracy before and after feature removal.

Question 9

Refer below to Figure C for diagram

The simplified decision tree for phishing detection, constructed using only two attributes, A01 and A23, strikes an effective balance between simplicity and performance. This tree, with nine terminal nodes, demonstrated a residual mean deviance of 0.677 and a misclassification error rate of 13.4%, calculated from the 470 test instances. The confusion matrix revealed that the model correctly identified 673 legitimate sites (true negatives) and 171 phishing sites (true positives), while incorrectly classifying 51 legitimate sites as phishing (false positives) and 195 phishing sites as legitimate (false negatives). This resulted in an overall accuracy of approximately 83.3%. Additionally, the ROC curve, which plots the true positive rate against the false positive rate, yielded an AUC (Area Under the Curve) of 0.809, signifying a good level of model performance in distinguishing between phishing and legitimate sites. Despite a higher number of false negatives, indicating some missed phishing sites, the model's simplicity makes it highly interpretable and practical for manual classification. The attributes A01 and A23 were selected due to their significant impact on the classification process, with A01 providing a primary split and A23 further refining the decision criteria.At the root of the tree, the initial decision is made by evaluating whether A01 is less than 11; if true, the site is classified as legitimate. If A01 is 11 or greater, the decision-making process proceeds to consider A23. If A23 is 1.5 or less, the site is again classified as legitimate. For values of A23 greater than 1.5, the tree further splits based on whether A01 is less than 38, classifying sites as legitimate in this case as well. If A01 is 38 or more, additional splits based on A23 are made to refine the classification. For instance, if A23 is between 118.5 and 132.5, the site is classified as phishing, whereas other ranges result in classifications as either legitimate or phishing depending on the specific value of A23. This decision tree, with its clear and methodical splits, allows for easy manual classification by following a simple series of yes/no questions based on the attributes. The attributes A01 and A23 were selected due to their significant contribution to distinguishing between phishing and legitimate sites, providing a balance between simplicity and accuracy in the model.

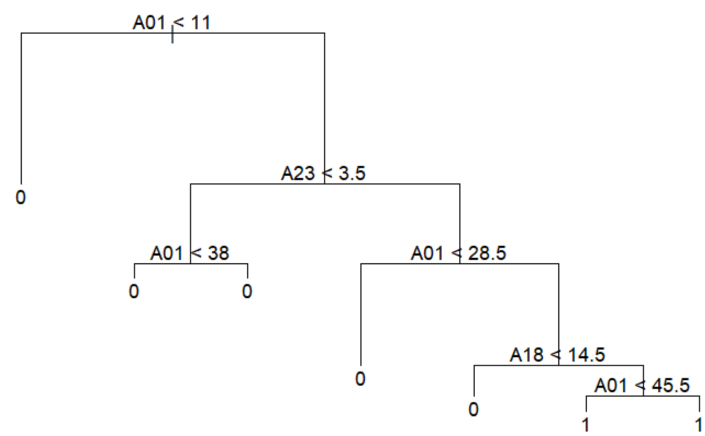

figure C

Question 10

In our comprehensive efforts to enhance the performance of our random forest model, we explored a variety of advanced techniques. Initially, we tackled the challenge of NA values by omitting them from both the training and test datasets, resulting in an accuracy of 0.85 and an AUC of 0.85. Proceeding to Question 10, we implemented cross-validation with SMOTE to address class imbalance, tuned hyperparameters extensively, and experimented with feature engineering and increasing the number of trees in the model. Despite these advanced efforts, the highest accuracy we achieved was 0.829, with an AUC of 0.818.

Interestingly, simpler techniques learned during the course initially improved our accuracy to 0.85, but this came at the cost of a significant drop in AUC to 0.799. This disparity highlighted the importance of not just accuracy but also the AUC metric, especially given the imbalanced nature of our dataset, which contains a higher proportion of legitimate sites compared to phishing sites.
A high AUC is critical in this context because it measures the model's ability to distinguish between legitimate and phishing sites across all threshold levels. A model with a high AUC is better at identifying true positives and minimizing false positives and negatives. Therefore, while our advanced techniques marginally improved accuracy, they failed to enhance the AUC sufficiently, reinforcing that our original, simpler approach yielded the best overall performance for this dataset. Consequently, to answer the question of choosing the single best classifier I choose the Random Forest from question where we have just removed NA values from both the train and the test dataset passed into it and it resulted in good accuracy and auc values.

Question 11

Initially, we used a simple neural network model to differentiate between phishing and legitimate websites. The model was trained using a limited set of features and without extensive preprocessing or resampling, resulting in an unsatisfactory accuracy of around 15%, indicating significant underperformance. To improve the model's performance, we adopted a more robust methodology. We ensured that the datasets were free of NA values and combined them to create dummy variables for the top 10 important features identified from previous analyses: A01, A08, A23, A18, A10, A12, A17, A14, and A22. After splitting the data back into training and test sets, we resampled the training data with replacement to create a larger and more balanced training set. Using this resampled data, we trained a neural network with a more complex architecture, consisting of two hidden layers with 5 and 3 neurons, respectively.
The enhanced ANN model utilized the selected top features, which were identified as highly predictive across various other models such as Decision Tree, Bagging, Boosting, and Random Forest. The resulting confusion matrix showed that out of the observed legitimate websites (class 0), 329 were correctly identified, while 141 were misclassified as phishing (class 1). The overall accuracy achieved was 70%, a substantial improvement from the initial 15%. Despite thorough exploration and optimization of the attributes, this was the maximum accuracy attained. This demonstrates that the revised approach, which included better feature selection, data preprocessing, and model architecture, substantially enhanced the model's classification ability, yet further improvements may require additional data or alternative modeling techniques.

Comparatively, the Decision Tree model, which also relied on the key variables A01, A23, A18, and A22, achieved an accuracy of 82.8% and an AUC of 0.754. The Bagging model, which identified A01, A23, A22, and A18 as the most important features, had an accuracy of 83% and an AUC of 0.836. The Boosting model, highlighting the same key predictors, achieved an accuracy of 82.7% and an AUC of 0.861. Finally, the Random Forest model, which also underscored the significance of A01, A23, A22, and A18, attained the highest accuracy of 84.7% and an AUC of 0.88.

Despite achieving a 70% accuracy, the ANN model did not perform as well as the Decision Tree, Bagging, Boosting, and Random Forest models. This comparison highlights the importance of the chosen features and the effectiveness of different modeling techniques. The results underscore the necessity for robust preprocessing, feature selection, and model architecture to achieve significant predictive performance. While the revised ANN model shows promise, further improvements might require to experiment more with the attributes since reaching from 20% accuracy to 70% just by adjusting attributes took a lot of time.

Question 12

In our analysis, we employed the XGBoost classifier, utilizing the xgboost package in R (https://xgboost.readthedocs.io/) , to enhance our phishing detection model. XGBoost, or Extreme Gradient Boosting, is a robust and efficient implementation of the gradient boosting framework, which is particularly well-suited for classification and regression tasks. This approach builds an ensemble of weak learners, typically decision trees, where each subsequent model attempts to correct the errors made by the previous models, thereby incrementally improving the overall model performance.

In preparing the data, we converted the 'Class' variable into a binary factor and then into a numeric format suitable for XGBoost. The training and test datasets were transformed into matrix format. We defined the model parameters, setting the objective to binary:logistic, evaluation metric to auc (Area Under the Curve), maximum tree depth to 6, learning rate (eta) to 0.1, and utilized 2 threads (nthread) for parallel processing.

The model was trained over 100 rounds (iterations). Predictions on the test data were then evaluated, and a confusion matrix was generated. The model correctly classified 299 legitimate sites (true negatives) and 91 phishing sites (true positives), while incorrectly classifying 30 legitimate sites as phishing (false positives) and 50 phishing sites as legitimate (false negatives). This yielded an overall accuracy of approximately 82.98%. The model's performance was further validated using the ROC curve, which plots the true positive rate against the false positive rate, and the AUC (Area Under the Curve) was calculated to be 0.888, indicating a high level of model performance. The ROC curve was plotted to visualize the trade-off between the true positive rate and the false positive rate, demonstrating the effectiveness of the XGBoost classifier in distinguishing between phishing and legitimate sites. This classifier's ability to efficiently handle large datasets and deliver high accuracy makes it a powerful tool for phishing detection.

When comparing the performance of the XGBoost classifier to other models from question 4 such as Decision Tree, Bagging, Boosting, and Random Forest, several observations can be made. The Decision Tree model, with an accuracy of 0.828 and an AUC of 0.754, showed reliance on four key variables (A01, A23, A18, and A22) but had a slightly higher misclassification error rate. The Bagging model, which also highlighted the importance of the same variables, achieved a slightly better accuracy of 0.830 and an AUC of 0.836, indicating its ability to reduce variance and improve robustness. The Boosting model, with an accuracy of 0.827 and an AUC of 0.861, further emphasized these variables' significance, demonstrating improved AUC over the Bagging model by focusing on correcting the errors of weaker classifiers. The Random Forest model, with an accuracy of 0.847 and an AUC of 0.85, outperformed the other models in both metrics, likely due to its ensemble approach that reduces overfitting while capturing a wide range of features. In comparison, the XGBoost model achieved an accuracy of 0.830 and a superior AUC of 0.888, indicating its effectiveness in maximizing both sensitivity and specificity. While the Random Forest had a marginally higher accuracy, XGBoost's higher AUC suggests a better overall performance in distinguishing between phishing and legitimate sites, making it a strong candidate for this classification task.

Appendix (R Code)

```
> rm(list = ls())
> # Load necessary libraries
> library(tree)
> library(e1071)
> library(ROCR)
> library(randomForest)
> library(adabag)
> library(rpart)
> library(ggplot2)
> library(caret)
> library(ROSE)
> library(neuralnet)
> library(xgboost)
> library(pROC)
> options(digits = 3)
> # Set random seed using Student ID
> set.seed(33370311)
> # Load and sample the data
> Phish <- read.csv("C:/Users/Home/OneDrive/Desktop/3152/PhishingData.csv")
> L <- as.data.frame(c(1:50))
> L <- L[sample(nrow(L), 10, replace = FALSE),]
> Phish <- Phish[(Phish$A01 %in% L),]
> PD <- Phish[sample(nrow(Phish), 2000, replace = FALSE),] # Sample of 2000 rows
> # Question 1: Calculate and plot the proportion of phishing sites to legitimate sites
> phishing_proportion <- mean(PD$Class == 1)
> legitimate_proportion <- mean(PD$Class == 0)
> # Create a data frame for plotting
> data <- data.frame(
+   category = c("Phishing", "Legitimate"),
+   proportion = c(phishing_proportion, legitimate_proportion)
+ )
> # Plot the pie chart
> ggplot(data, aes(x = "", y = proportion, fill = category)) +
+   geom_bar(stat = "identity", width = 1) +
+   coord_polar(theta = "y") +
+   labs(title = "Proportion of Phishing Sites to Legitimate Sites") +
+   theme_void() +
+   geom_text(aes(label = scales::percent(proportion, accuracy = 0.1)),
+             position = position_stack(vjust = 0.5))
> # Question 2: Count NA values and summarize data
> na_count <- colSums(is.na(PD))
```

```
> print(na_count)
 A01   A02   A03   A04   A05   A06   A07   A08   A09   A10   A11   A12   A13   A14   A15   A16
A17   A18   A19
   0    27    21    13    21    24    16    29    21    14    23    21    18    16    20    21    18    13    18
 A20   A21   A22   A23   A24   A25 Class
  14    29    31    17    20    19     0
> summary(PD)
     A01            A02            A03           A04            A05             A06            A07
 Min.   : 1.0   Min.   : 0.0   Min.   :0    Min.   :2.00   Min.   : 0.00   Min.   :0.00   Min.   :0
 1st Qu.: 5.0   1st Qu.: 0.0   1st Qu.:0    1st Qu.:2.00   1st Qu.: 0.00   1st Qu.:0.00   1st Qu.:0
 Median :18.0   Median : 0.0   Median :0    Median :3.00   Median : 0.00   Median :0.00   Median
:0
 Mean   :23.3   Mean   : 0.1   Mean   :0    Mean   :2.77   Mean   : 0.02   Mean   :0.12   Mean   :0
 3rd Qu.:42.0   3rd Qu.: 0.0   3rd Qu.:0    3rd Qu.:3.00   3rd Qu.: 0.00   3rd Qu.:0.00   3rd Qu.:0
 Max.   :50.0   Max.   :32.0   Max.   :1    Max.   :8.00   Max.   :14.00   Max.   :1.00   Max.   :1
                NA's   :27     NA's   :21   NA's   :13     NA's   :21      NA's   :24     NA's   :16
     A08            A09            A10            A11             A12            A13            A14
 Min.   :0.15   Min.   :0.00   Min.   :0.00   Min.   : 0.00   Min.   : 48   Min.   :0.00   Min.   :0.00
 1st Qu.:0.70   1st Qu.:0.00   1st Qu.:0.00   1st Qu.: 0.00   1st Qu.:232   1st Qu.:0.00   1st
Qu.:0.00
 Median :1.00   Median :0.00   Median :0.00   Median : 0.00   Median :232   Median :0.00
Median :0.00
 Mean   :0.85   Mean   :0.02   Mean   :0.03   Mean   : 0.07   Mean   :315   Mean   :0.01   Mean
:0.12
 3rd Qu.:1.00   3rd Qu.:0.00   3rd Qu.:0.00   3rd Qu.: 0.00   3rd Qu.:388   3rd Qu.:0.00   3rd
Qu.:0.00
 Max.   :1.00   Max.   :1.00   Max.   :1.00   Max.   :31.00   Max.   :692   Max.   :9.00   Max.   :1.00
 NA's   :29     NA's   :21     NA's   :14     NA's   :23      NA's   :21    NA's   :18     NA's   :16
     A15            A16            A17            A18             A19            A20            A21
 Min.   :0.00   Min.   :0.00   Min.   :0.00   Min.   :   5   Min.   :0.0   Min.   :0.00   Min.   :0.00
 1st Qu.:0.00   1st Qu.:0.00   1st Qu.:1.00   1st Qu.:  13   1st Qu.:0.0   1st Qu.:0.00   1st
Qu.:0.00
 Median :0.00   Median :0.00   Median :1.00   Median :  32   Median :0.0   Median :0.00   Median
:0.00
 Mean   :0.12   Mean   :0.05   Mean   :1.14   Mean   :  61   Mean   :0.1   Mean   :0.23   Mean
:0.03
 3rd Qu.:0.00   3rd Qu.:0.00   3rd Qu.:1.00   3rd Qu.:  89   3rd Qu.:0.0   3rd Qu.:0.00   3rd
Qu.:0.00
 Max.   :1.00   Max.   :1.00   Max.   :5.00   Max.   :1814   Max.   :1.0   Max.   :1.00   Max.   :3.00
 NA's   :20     NA's   :21     NA's   :18     NA's   :13     NA's   :18    NA's   :14     NA's   :29
     A22            A23           A24            A25          Class
 Min.   :0.01   Min.   :  0   Min.   :0.00   Min.   :0.0   Min.   :0.000
 1st Qu.:0.05   1st Qu.: 13   1st Qu.:0.01   1st Qu.:0.0   1st Qu.:0.000
 Median :0.06   Median :100   Median :0.08   Median :0.0   Median :0.000
```

```
 Mean   :0.06   Mean   : 72   Mean   :0.27   Mean   :0.0   Mean   :0.328
 3rd Qu.:0.06   3rd Qu.:106   3rd Qu.:0.52   3rd Qu.:0.0   3rd Qu.:1.000
 Max.   :0.08   Max.   :901   Max.   :0.52   Max.   :0.1   Max.   :1.000
 NA's   :31     NA's   :17    NA's   :20     NA's   :19
> # Question 3: Calculate mean, standard deviation, and variance for website attributes
> website_attributes <- PD[, grepl("^A", names(PD))]
> mean_values <- apply(website_attributes, 2, mean, na.rm = TRUE)
> sd_values <- apply(website_attributes, 2, sd, na.rm = TRUE)
> var_values <- apply(website_attributes, 2, var, na.rm = TRUE)
> summary_stats <- data.frame(
+   Attribute = names(mean_values),
+   Mean = mean_values,
+   SD = sd_values,
+   Variance = var_values
+ )
> print(summary_stats)
    Attribute     Mean       SD Variance
A01       A01 2.33e+01  17.9234 3.21e+02
A02       A02 1.36e-01   0.9994 9.99e-01
A03       A03 2.02e-03   0.0449 2.02e-03
A04       A04 2.77e+00   0.5457 2.98e-01
A05       A05 1.62e-02   0.4006 1.61e-01
A06       A06 1.17e-01   0.3220 1.04e-01
A07       A07 1.01e-03   0.0317 1.01e-03
A08       A08 8.49e-01   0.2168 4.70e-02
A09       A09 2.32e-02   0.1507 2.27e-02
A10       A10 3.32e-02   0.1793 3.21e-02
A11       A11 6.93e-02   0.8505 7.23e-01
A12       A12 3.15e+02 138.9938 1.93e+04
A13       A13 7.57e-03   0.2429 5.90e-02
A14       A14 1.25e-01   0.3308 1.09e-01
A15       A15 1.18e-01   0.3223 1.04e-01
A16       A16 5.00e-02   0.2181 4.75e-02
A17       A17 1.14e+00   0.5708 3.26e-01
A18       A18 6.07e+01 105.7457 1.12e+04
A19       A19 9.59e-02   0.2945 8.67e-02
A20       A20 2.29e-01   0.4200 1.76e-01
A21       A21 2.99e-02   0.1928 3.72e-02
A22       A22 5.60e-02   0.0105 1.11e-04
A23       A23 7.17e+01  61.8825 3.83e+03
A24       A24 2.71e-01   0.2512 6.31e-02
A25       A25 9.19e-05   0.0029 8.39e-06
> # Split the data into training and test sets
> train.row = sample(1:nrow(PD), 0.7 * nrow(PD))
```

```
> PD.train = PD[train.row,]
> PD.test = PD[-train.row,]
> PD.train$Class <- factor(PD.train$Class)
> PD.test$Class <- factor(PD.test$Class)
> # Decision Tree
> PDtree = tree(Class ~ ., data = PD.train)
> print(summary(PDtree))

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A01" "A23" "A18"
Number of terminal nodes:  7
Residual mean deviance:  0.79 = 865 / 1100
Misclassification error rate: 0.164 = 181 / 1102
> plot(PDtree)
> text(PDtree, pretty = 0)
> # Naive Bayes
> model_nb <- naiveBayes(Class ~ ., data = PD.train)
> # Bagging
> PDbag <- bagging(Class ~ ., data = PD.train, mfinal = 5)
> # Boosting
> PDboost <- boosting(Class ~ ., data = PD.train, mfinal = 5)
> # Random Forest
> PDrf.train <- na.omit(PD.train) # Omit missing values
> PD.rf <- randomForest(Class ~ ., data = PDrf.train)
> # Decision Tree Confusion Matrix
> PD.predtree = predict(PDtree, PD.test, type = "class")
> t1 = table(Predicted_Class = PD.predtree, Actual_Class = PD.test$Class)
> cat("\n# Decision Tree Confusion\n")

# Decision Tree Confusion
> print(t1)
               Actual_Class
Predicted_Class   0   1
              0 391  89
              1  23  97
> # Naive Bayes Confusion Matrix
> PD.predbayes = predict(model_nb, PD.test)
> t2 = table(Predicted_Class = PD.predbayes, Actual_Class = PD.test$Class)
> cat("\n# Naive Bayes Confusion\n")

# Naive Bayes Confusion
> print(t2)
```

```
           Actual_Class
Predicted_Class   0   1
              0  17   1
              1 397 185
> # Bagging Confusion Matrix
> PDpred.bag <- predict(PDbag, newdata = PD.test, type = "class")
> predicted <- ifelse(PDpred.bag$prob[, 2] > 0.5, 1, 0)
> confusionMatrix <- table(Predicted = predicted, Actual = PD.test$Class)
> cat("\n# Bagging Confusion\n")

# Bagging Confusion
> print(confusionMatrix)
         Actual
Predicted   0   1
        0 389  82
        1  25 104
> # Boosting Confusion Matrix
> PDpred.boost <- predict(PDboost, newdata = PD.test, type = "class")
> predicted <- PDpred.boost$class
> confusionMatrix <- table(Predicted = predicted, Actual = PD.test$Class)
> cat("\n# Boosting Confusion\n")

# Boosting Confusion
> print(confusionMatrix)
         Actual
Predicted   0   1
        0 378  74
        1  36 112
> # Random Forest Confusion Matrix
> PDpred.rf <- predict(PD.rf, newdata = PD.test)
> confusionMatrix <- table(Predicted = PDpred.rf, Actual = PD.test$Class)
> cat("\n# Random Forest Confusion\n")

# Random Forest Confusion
> print(confusionMatrix)
         Actual
Predicted   0   1
        0 294  55
        1  22  87
> # Setup an initial plot area
> plot(0, type = "n", xlim = c(0, 1), ylim = c(0, 1), xlab = "False Positive Rate", ylab = "True
Positive Rate", main = "ROC Curves Comparison")
> # Decision Tree
> PD.pred.tree = predict(PDtree, PD.test, type = "vector")
```

```
> PD.pred.tree = as.data.frame(PD.pred.tree)
> PDpred <- ROCR::prediction(PD.pred.tree[, 2], PD.test$Class)
> PDperf <- performance(PDpred, "tpr", "fpr")
> plot(PDperf, col = "black", add = TRUE, lwd = 2)
> # Naive Bayes
> PDpred.bayes = predict(model_nb, PD.test, type = 'raw')
> PDpred <- ROCR::prediction(PDpred.bayes[, 2], PD.test$Class)
> PDperf <- performance(PDpred, "tpr", "fpr")
> plot(PDperf, col = "blueviolet", add = TRUE, lwd = 2)
> # Bagging
> PDpred.bag <- predict.bagging(PDbag, PD.test, type = "prob")
> PDBagpred <- ROCR::prediction(PDpred.bag$prob[, 2], PD.test$Class)
> PDBagperf <- performance(PDBagpred, "tpr", "fpr")
> plot(PDBagperf, col = "blue", add = TRUE, lwd = 2)
> # Boosting
> PDpred.boost <- predict.boosting(PDboost, newdata = PD.test)
> PDBoostpred <- ROCR::prediction(PDpred.boost$prob[, 2], PD.test$Class)
> PDBoostperf <- performance(PDBoostpred, "tpr", "fpr")
> plot(PDBoostperf, col = "red", add = TRUE, lwd = 2)
> # Random Forest
> PDrf.test <- na.omit(PD.test)
> PDpred.rf <- predict(PD.rf, PDrf.test, type = "prob")
> PDrfpred <- ROCR::prediction(PDpred.rf[, 2], PDrf.test$Class)
> PDrfperf <- performance(PDrfpred, "tpr", "fpr")
> plot(PDrfperf, col = "green", add = TRUE, lwd = 2)
> # Add a diagonal line
> abline(0, 1, lty = 2, col = "gray")
> # Add legend
> legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest"),
+       col = c("black", "blueviolet", "blue", "red", "green"), lwd = 2)
> # Calculate and print AUC for each model
> PDauc <- performance(PDpred, "auc")
> cat("Decision Tree AUC:", as.numeric(PDauc@y.values), "\n")
Decision Tree AUC: 0.741
> PDauc <- performance(PDpred, "auc")
> cat("Naive Bayes AUC:", as.numeric(PDauc@y.values), "\n")
Naive Bayes AUC: 0.741
> PDBagAuc <- performance(PDBagpred, "auc")
> cat("Bagging AUC:", as.numeric(PDBagAuc@y.values), "\n")
Bagging AUC: 0.782
> PDBoostAuc <- performance(PDBoostpred, "auc")
> cat("Boosting AUC:", as.numeric(PDBoostAuc@y.values), "\n")
Boosting AUC: 0.838
```

```
> PDrfAuc <- performance(PDrfpred, "auc")
> cat("Random Forest AUC:", as.numeric(PDrfAuc@y.values), "\n")
Random Forest AUC: 0.848
> # Question 8: Attribute Importance
> cat("\n# Decision Tree Attribute Importance\n")

# Decision Tree Attribute Importance
> print(summary(PDtree))

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A01" "A23" "A18"
Number of terminal nodes:  7
Residual mean deviance:  0.79 = 865 / 1100
Misclassification error rate: 0.164 = 181 / 1102
> # Bagging
> importance <- PDbag$importance
> print(importance)
  A01    A02    A03    A04    A05    A06    A07    A08    A09    A10    A11    A12    A13    A14
A15    A16
56.674  1.358  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.355  0.000
0.000  0.403  0.000
  A17    A18    A19    A20    A21    A22    A23    A24    A25
 0.509 10.454  0.000  0.527  0.000  2.233 26.487  0.000  0.000
> # Boosting
> importance <- PDboost$importance
> print(importance)
  A01    A02    A03    A04    A05    A06    A07    A08    A09    A10    A11    A12    A13    A14
A15    A16
55.854  1.179  0.000  0.000  0.000  0.504  0.000  1.723  0.000  0.000  0.000  3.370  0.000
0.793  0.000  0.204
  A17    A18    A19    A20    A21    A22    A23    A24    A25
 0.000  7.654  0.000  0.610  0.000  6.309 20.901  0.899  0.000
> # Random Forest
> importance <- importance(PD.rf)
> varImportance <- data.frame(Variables = row.names(importance), Importance = importance[,
1])
> varImportance <- varImportance[order(-varImportance$Importance), ]
> print(varImportance)
    Variables Importance
A01       A01   1.09e+02
A23       A23   7.41e+01
A18       A18   6.17e+01
```

```
A22      A22   5.62e+01
A08      A08   2.44e+01
A14      A14   2.28e+01
A24      A24   2.26e+01
A12      A12   2.02e+01
A17      A17   9.59e+00
A20      A20   8.16e+00
A04      A04   7.49e+00
A02      A02   5.08e+00
A06      A06   4.54e+00
A15      A15   4.36e+00
A19      A19   4.31e+00
A16      A16   3.39e+00
A09      A09   1.82e+00
A10      A10   1.56e+00
A11      A11   1.31e+00
A21      A21   1.24e+00
A05      A05   4.45e-02
A03      A03   2.07e-02
A13      A13   9.64e-03
A25      A25   8.51e-03
A07      A07   5.65e-03
> #Question 9
> PD.train$Class <- factor(PD.train$Class, levels = c(0, 1))
> PD.test$Class <- factor(PD.test$Class, levels = c(0, 1))
> # Train a simpler Decision Tree model using only three attributes (A01, A23, A18)
> PDtree_simple = tree(Class ~ A01 + A23 + A18, data = PD.train)
> print(summary(PDtree_simple))

Classification tree:
tree(formula = Class ~ A01 + A23 + A18, data = PD.train)
Number of terminal nodes:  7
Residual mean deviance:  0.797 = 1090 / 1370
Misclassification error rate: 0.167 = 230 / 1379
> # Plot the simpler Decision Tree
> plot(PDtree_simple)
> text(PDtree_simple, pretty = 0)
> # Predict using the simpler Decision Tree
> PD.predtree_simple = predict(PDtree_simple, PD.test, type = "class")
> t1_simple = table(Predicted_Class = PD.predtree_simple, Actual_Class = PD.test$Class)
> cat("\n# Simpler Decision Tree Confusion Matrix\n")

# Simpler Decision Tree Confusion Matrix
> print(t1_simple)
```

```
             Actual_Class
Predicted_Class   0   1
            0 391  89
            1  23  97
> # Calculate accuracy for the simpler Decision Tree
> accuracy_simple = sum(diag(t1_simple)) / sum(t1_simple)
> print(paste("Simpler Decision Tree Accuracy:", accuracy_simple))
[1] "Simpler Decision Tree Accuracy: 0.813333333333333"
> # Generate ROC curve and calculate AUC for the simpler Decision Tree
> PD.pred.tree_simple = predict(PDtree_simple, PD.test, type = "vector")
> PD.pred.tree_simple = as.data.frame(PD.pred.tree_simple)
> PDpred_simple = ROCR::prediction(PD.pred.tree_simple[, 2], PD.test$Class)
> PDperf_simple = performance(PDpred_simple, "tpr", "fpr")
> plot(PDperf_simple, col = "black", add = TRUE, lwd = 2)
> PDauc_simple = performance(PDpred_simple, "auc")
> cat("Simpler Decision Tree AUC:", as.numeric(PDauc_simple@y.values), "\n")
Simpler Decision Tree AUC: 0.841
> # Set up control function for cross-validation with class balancing
> train_control <- trainControl(method = "cv", number = 5, sampling = "smote")
> # Define the grid of hyperparameters to search
> tune_grid <- expand.grid(mtry
+
+                        = seq(1, 4, by = 1))
> PD.train<- na.omit(PD.train)
> PD.test <- na.omit(PD.test)
> # Train the model using cross-validation
> tuned_rf <- train(Class ~ ., data = PD.train, method = "rf", trControl = train_control, tuneGrid =
tune_grid, ntree = 500, maxnodes = 30)
> # Print the best parameters
> print(tuned_rf$bestTune)
  mtry
3    3
> # Evaluate the tuned Random Forest
> PDpred.rf <- predict(tuned_rf, PD.test)
> confusion_matrix_rf <- table(Predicted_Class = PDpred.rf, Actual_Class = PD.test$Class)
> accuracy_rf <- sum(diag(confusion_matrix_rf)) / sum(confusion_matrix_rf)
> print(paste("Tuned Random Forest Accuracy:", accuracy_rf))
[1] "Tuned Random Forest Accuracy: 0.794759825327511"
> # AUC for tuned Random Forest
> PDpred_rf_roc <- ROCR::prediction(as.numeric(PDpred.rf), as.numeric(PD.test$Class))
> PDrfAuc <- performance(PDpred_rf_roc, "auc")
> cat("Tuned Random Forest AUC:", as.numeric(PDrfAuc@y.values), "\n")
Tuned Random Forest AUC: 0.781
> # Omit NA values from the datasets
```

```
> PD.train <- na.omit(PD.train)
> PD.test <- na.omit(PD.test)
> # Ensure Class column is a factor
> PD.train$Class <- as.factor(PD.train$Class)
> PD.test$Class <- as.factor(PD.test$Class)
> # Combine train and test datasets to create dummy variables
> combined_data <- rbind(PD.train, PD.test)
> # Create dummy variables using model.matrix only for the necessary columns
> ptmm <- model.matrix(~ A01 + A08 + A23 + A18 + A10 + A12 + A17 + A14 + A22 - 1, data =
combined_data)
> # Combine the dummy variables with the Class column
> ptcombined <- data.frame(ptmm, Class = combined_data$Class)
> # Debug: Check the structure of ptcombined
> str(ptcombined)
'data.frame':	1560 obs. of  10 variables:
 $ A01  : num  49 1 5 1 18 18 1 17 1 34 ...
 $ A08  : num  0.263 1 1 0.727 0.371 ...
 $ A23  : num  3 102 100 115 1 100 27 104 100 0 ...
 $ A18  : num  36 112 5 26 17 6 43 96 14 6 ...
 $ A10  : num  0 0 0 0 0 0 0 0 1 0 ...
 $ A12  : num  418 232 141 554 227 504 232 232 210 650 ...
 $ A17  : num  3 1 1 1 1 1 1 1 1 1 ...
 $ A14  : num  0 0 0 0 0 0 0 1 0 0 ...
 $ A22  : num  0.0456 0.0604 0.0674 0.053 0.0549 ...
 $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
> # Split the data back into train and test sets
> pttest <- ptcombined[(nrow(PD.train) + 1):nrow(ptcombined), ]
> pttrain <- ptcombined[1:nrow(PD.train), ]
> # Debug: Check the structure of pttrain
> str(pttrain)
'data.frame':	1102 obs. of  10 variables:
 $ A01  : num  49 1 5 1 18 18 1 17 1 34 ...
 $ A08  : num  0.263 1 1 0.727 0.371 ...
 $ A23  : num  3 102 100 115 1 100 27 104 100 0 ...
 $ A18  : num  36 112 5 26 17 6 43 96 14 6 ...
 $ A10  : num  0 0 0 0 0 0 0 0 1 0 ...
 $ A12  : num  418 232 141 554 227 504 232 232 210 650 ...
 $ A17  : num  3 1 1 1 1 1 1 1 1 1 ...
 $ A14  : num  0 0 0 0 0 0 0 1 0 0 ...
 $ A22  : num  0.0456 0.0604 0.0674 0.053 0.0549 ...
 $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
> # Ensure the resampling is done correctly
> pttrain <- as.data.frame(pttrain)  # Ensure it's a data frame
> str(pttrain)  # Debug: Check the structure of pttrain
```

```
'data.frame':	1102 obs. of  10 variables:
 $ A01  : num  49 1 5 1 18 18 1 17 1 34 ...
 $ A08  : num  0.263 1 1 0.727 0.371 ...
 $ A23  : num  3 102 100 115 1 100 27 104 100 0 ...
 $ A18  : num  36 112 5 26 17 6 43 96 14 6 ...
 $ A10  : num  0 0 0 0 0 0 0 0 1 0 ...
 $ A12  : num  418 232 141 554 227 504 232 232 210 650 ...
 $ A17  : num  3 1 1 1 1 1 1 1 1 1 ...
 $ A14  : num  0 0 0 0 0 0 0 1 0 0 ...
 $ A22  : num  0.0456 0.0604 0.0674 0.053 0.0549 ...
 $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
> # Resampling with replacement to create a larger training set
> set.seed(9999)
> pttrain <- pttrain[sample(nrow(pttrain), 100, replace = TRUE), ]
> # Debug: Check the structure again after resampling
> str(pttrain)
'data.frame':	100 obs. of  10 variables:
 $ A01  : num  23 42 3 5 49 1 17 17 1 3 ...
 $ A08  : num  0.688 1 0.542 1 0.615 ...
 $ A23  : num  100 8 100 0 3 100 100 100 100 127 ...
 $ A18  : num  19 16 8 131 35 74 29 16 22 165 ...
 $ A10  : num  0 0 0 0 0 0 0 0 1 0 ...
 $ A12  : num  227 232 692 232 504 232 227 48 232 232 ...
 $ A17  : num  1 1 2 1 1 0 1 2 1 1 ...
 $ A14  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ A22  : num  0.0398 0.0662 0.0489 0.0642 0.0679 ...
 $ Class: Factor w/ 2 levels "0","1": 1 2 1 1 2 1 1 1 1 1 ...
> # Ensure Class column is a factor in pttrain
> pttrain$Class <- factor(pttrain$Class)
> # Define the formula for the ANN
> formula <- Class ~ A01 + A23 + A18 + A22 + A08 + A14 + A10 + A17 + A12
> # Train the neural network
> set.seed(9999) # For reproducibility
> web.nn <- neuralnet::neuralnet(formula, data = pttrain, hidden = c(5, 3), linear.output =
FALSE)
> # Print the neural network result matrix
> print(web.nn$result.matrix)
                        [,1]
error                18.81938
reachedthreshold       0.00711
steps                232.00000
Intercept.to.1layhid1   1.08410
A01.to.1layhid1         0.84311
A23.to.1layhid1         0.49439
```

A18.to.1layhid1	*-0.77302*
A22.to.1layhid1	*2.90382*
A08.to.1layhid1	*0.90888*
A14.to.1layhid1	*-0.96308*
A10.to.1layhid1	*-0.18829*
A17.to.1layhid1	*1.42037*
A12.to.1layhid1	*0.98644*
Intercept.to.1layhid2	*-0.44994*
A01.to.1layhid2	*1.07167*
A23.to.1layhid2	*-0.41657*
A18.to.1layhid2	*-0.42914*
A22.to.1layhid2	*-0.11666*
A08.to.1layhid2	*-0.29854*
A14.to.1layhid2	*-0.17924*
A10.to.1layhid2	*2.33290*
A17.to.1layhid2	*-0.29895*
A12.to.1layhid2	*1.73420*
Intercept.to.1layhid3	*0.50184*
A01.to.1layhid3	*0.91845*
A23.to.1layhid3	*0.16616*
A18.to.1layhid3	*-0.10651*
A22.to.1layhid3	*-0.02277*
A08.to.1layhid3	*-0.46903*
A14.to.1layhid3	*-1.32928*
A10.to.1layhid3	*1.16020*
A17.to.1layhid3	*-0.21783*
A12.to.1layhid3	*0.20753*
Intercept.to.1layhid4	*-0.23110*
A01.to.1layhid4	*-3.36506*
A23.to.1layhid4	*0.28651*
A18.to.1layhid4	*1.38127*
A22.to.1layhid4	*1.40316*
A08.to.1layhid4	*0.63110*
A14.to.1layhid4	*0.02509*
A10.to.1layhid4	*18.55262*
A17.to.1layhid4	*-0.91919*
A12.to.1layhid4	*-0.60117*
Intercept.to.1layhid5	*0.88241*
A01.to.1layhid5	*0.03414*
A23.to.1layhid5	*0.75454*
A18.to.1layhid5	*-0.90547*
A22.to.1layhid5	*-10.89603*
A08.to.1layhid5	*0.64428*
A14.to.1layhid5	*-21.45017*

```
A10.to.1layhid5       -21.11123
A17.to.1layhid5        -0.74218
A12.to.1layhid5        -0.47448
Intercept.to.2layhid1   0.45451
1layhid1.to.2layhid1   -0.80967
1layhid2.to.2layhid1    0.45174
1layhid3.to.2layhid1   -0.47330
1layhid4.to.2layhid1    6.21546
1layhid5.to.2layhid1   -0.81985
Intercept.to.2layhid2  -0.24992
1layhid1.to.2layhid2   -1.40432
1layhid2.to.2layhid2   -0.24691
1layhid3.to.2layhid2    0.09653
1layhid4.to.2layhid2   16.48128
1layhid5.to.2layhid2  -20.38522
Intercept.to.2layhid3  -1.91527
1layhid1.to.2layhid3    0.63464
1layhid2.to.2layhid3   -0.02032
1layhid3.to.2layhid3    0.04150
1layhid4.to.2layhid3   13.70166
1layhid5.to.2layhid3  -22.90483
Intercept.to.0         -0.30314
2layhid1.to.0           1.43944
2layhid2.to.0           2.69512
2layhid3.to.0           0.93245
Intercept.to.1         -0.25790
2layhid1.to.1           0.92211
2layhid2.to.1          -2.53875
2layhid3.to.1          -2.83277
> # Predict using the neural network
> web.pred <- neuralnet::compute(web.nn, pttest[, c("A01", "A23", "A18", "A22", "A14", "A08",
"A10", "A17", "A12")])
> # Get the probabilities
> prob <- web.pred$net.result[, 1]  # Assuming the first column is the relevant one
> # Convert probabilities to binary predictions
> pred <- ifelse(prob > 0.5, 1, 0)
> # Confusion matrix
> confusion_matrix <- table(observed = pttest$Class, predicted = pred)
> print(confusion_matrix)
        predicted
observed   1
       0 316
       1 142
> # Calculate accuracy
```

```
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> print(paste("Accuracy:", accuracy))
[1] "Accuracy: 0.689956331877729"
> # Convert Class to binary factor and then to numeric
> PD.train$Class <- as.numeric(as.character(as.factor(PD.train$Class)))
> PD.test$Class <- as.numeric(as.character(as.factor(PD.test$Class)))
> # Convert data to matrix format for xgboost
> train_matrix <- as.matrix(PD.train[, -which(names(PD.train) == "Class")])
> train_labels <- PD.train$Class
> test_matrix <- as.matrix(PD.test[, -which(names(PD.test) == "Class")])
> test_labels <- PD.test$Class
> # Set up parameters for xgboost
> params <- list(
+   objective = "binary:logistic",
+   eval_metric = "auc",
+   max_depth = 6,
+   eta = 0.1,
+   nthread = 2
+ )
> # Train the model
> xgb_model <- xgboost(
+   data = train_matrix,
+   label = train_labels,
+   params = params,
+   nrounds = 100,
+   verbose = 0
+ )
> # Predict on test data
> xgb_predictions <- predict(xgb_model, test_matrix)
> xgb_pred_labels <- ifelse(xgb_predictions > 0.5, 1, 0)
> # Create a confusion matrix
> xgb_confusion_matrix <- table(Predicted = xgb_pred_labels, Actual = test_labels)
> print(xgb_confusion_matrix)
         Actual
Predicted   0   1
        0 280  50
        1  36  92
> # Calculate AUC using pROC
> roc_obj <- roc(test_labels, xgb_predictions)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc_value <- auc(roc_obj)
> print(paste("XGBoost AUC:", auc_value))
[1] "XGBoost AUC: 0.852079247637725"
```

```
> # Plot ROC curve
> plot(roc_obj, col = "orange", lwd = 2, main = "ROC Curve for XGBoost")
> abline(0, 1, lty = 2, col = "gray")
> legend("bottomright", legend = c("XGBoost"), col = c("orange"), lwd = 2)
```