assignment

2024-05-15

Import necessary libraries

```
library(readr)
library(forcats)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ lubridate 1.9.3     ✓ tibble    3.2.1
## ✓ purrr     1.0.2     ✓ tidyr     1.3.1
## ✓ stringr   1.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✕ tidyr::expand() masks Matrix::expand()
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
## ✕ purrr::lift()   masks caret::lift()
## ✕ tidyr::pack()   masks Matrix::pack()
## ✕ tidyr::unpack() masks Matrix::unpack()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(Boruta)
library(mlbench)
library(party)
```

```
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Loading required package: sandwich
##
## Attaching package: 'strucchange'
##
## The following object is masked from 'package:stringr':
##
##     boundary
##
##
## Attaching package: 'party'
##
## The following object is masked from 'package:dplyr':
##
##     where
```

```
library(gbm)
```

```
## Loaded gbm 2.1.9
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com/gbm-developers/gbm
3
```

```r
library(e1071)
library(class)
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(ipred)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Load the dataset

```r
Phish <- read_csv("phishingdata.csv", col_names = TRUE)
```

```
## Rows: 100000 Columns: 26
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (26): A01, A02, A03, A04, A05, A06, A07, A08, A09, A10, A11, A12, A13, A...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Check for missing values

```
summary(Phish)
```

```
##       A01            A02                 A03              A04       
##  Min.   : 1.0   Min.   :  0.0000   Min.   :0.0000   Min.   :2.000  
##  1st Qu.:13.0   1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:2.000  
##  Median :25.5   Median :  0.0000   Median :0.0000   Median :3.000  
##  Mean   :25.5   Mean   :  0.2121   Mean   :0.0015   Mean   :2.754  
##  3rd Qu.:38.0   3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:3.000  
##  Max.   :50.0   Max.   :396.0000   Max.   :1.0000   Max.   :8.000  
##                 NA's   :1023       NA's   :961      NA's   :961    
##       A05                A06              A07              A08        
##  Min.   :  0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0556  
##  1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.6818  
##  Median :  0.0000   Median :0.0000   Median :0.0000   Median :1.0000  
##  Mean   :  0.0236   Mean   :0.1257   Mean   :0.0021   Mean   :0.8458  
##  3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:1.0000  
##  Max.   :149.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
##  NA's   :1005       NA's   :1005     NA's   :1018     NA's   :1011    
##       A09              A10              A11                 A12       
##  Min.   :0.0000   Min.   :0.0000   Min.   :  0.0000   Min.   :  3.0  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:  0.0000   1st Qu.:232.0  
##  Median :0.0000   Median :0.0000   Median :  0.0000   Median :232.0  
##  Mean   :0.0237   Mean   :0.0394   Mean   :  0.0594   Mean   :319.8  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:  0.0000   3rd Qu.:444.0  
##  Max.   :1.0000   Max.   :1.0000   Max.   :176.0000   Max.   :695.0  
##  NA's   :1019     NA's   :957      NA's   :973        NA's   :999    
##       A13                A14              A15             A16        
##  Min.   :  0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000  
##  1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000  
##  Median :  0.0000   Median :0.0000   Median :0.000   Median :0.0000  
##  Mean   :  0.0289   Mean   :0.1345   Mean   :0.133   Mean   :0.0438  
##  3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.000   3rd Qu.:0.0000  
##  Max.   :447.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000  
##  NA's   :1011       NA's   :1022     NA's   :967     NA's   :1014    
##       A17              A18                A19              A20        
##  Min.   : 0.000   Min.   :   4.00   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.: 1.000   1st Qu.:  14.00   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median : 1.000   Median :  32.00   Median :0.0000   Median :0.0000  
##  Mean   : 1.164   Mean   :  59.16   Mean   :0.1018   Mean   :0.2372  
##  3rd Qu.: 1.000   3rd Qu.:  89.00   3rd Qu.:0.0000   3rd Qu.:0.0000  
```

```
##  Max.   :10.000   Max.   :3738.00   Max.   :1.0000   Max.   :1.0000  
##  NA's   :1006     NA's   :967       NA's   :1003     NA's   :993     
##       A21              A22              A23                A24        
##  Min.   :0.0000   Min.   :0.0012   Min.   :   0.00   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0508   1st Qu.:   8.00   1st Qu.:0.0070  
##  Median :0.0000   Median :0.0580   Median : 100.00   Median :0.0800  
##  Mean   :0.0283   Mean   :0.0558   Mean   :  68.35   Mean   :0.2636  
##  3rd Qu.:0.0000   3rd Qu.:0.0629   3rd Qu.: 105.00   3rd Qu.:0.5229  
##  Max.   :4.0000   Max.   :0.0908   Max.   :4778.00   Max.   :0.5229  
##  NA's   :1022     NA's   :1031     NA's   :1015      NA's   :986     
##       A25             Class       
##  Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.0000  
##  Mean   :0.0001   Mean   :0.3635  
##  3rd Qu.:0.0000   3rd Qu.:1.0000  
##  Max.   :0.3200   Max.   :1.0000  
##  NA's   :1004
```

```
miss_pct <- sapply(Phish, function(x) sum(is.na(x)) / length(x) * 100)
print(miss_pct)
```

```
##   A01   A02   A03   A04   A05   A06   A07   A08   A09   A10   A11   A12   A13 
## 0.000 1.023 0.961 0.961 1.005 1.005 1.018 1.011 1.019 0.957 0.973 0.999 1.011 
##   A14   A15   A16   A17   A18   A19   A20   A21   A22   A23   A24   A25 Class 
## 1.022 0.967 1.014 1.006 0.967 1.003 0.993 1.022 1.031 1.015 0.986 1.004 0.000
```

Handle missing data

Impute missing values using median for numeric features

```
Phish <- Phish %>%
  mutate_if(is.numeric, ~replace_na(., median(., na.rm = TRUE)))
```

Set the random seed using your Student ID

```
set.seed(100000) # Replace 100000 with your Student ID
```

#Let's create Dataset for our use:

```
# Create a data frame with numbers 1 to 50
L <- as.data.frame(c(1:50))

# Randomly sample 10 rows from L without replacement
L <- L[sample(nrow(L), 10, replace = FALSE), ]

# Filter Phish to only include rows where A01 is in the sampled L

Phish <- Phish[(Phish$A01 %in% L), ]

# Randomly sample 2000 rows from Phish without replacement
PD <- Phish[sample(nrow(Phish), 2000, replace = FALSE), ]
```

Feature Selection

```
set.seed(100000)
default_br <- Boruta(Class ~ ., data = PD, doTrace = 2, maxRuns = 250)
```

```
##  1. run of importance source...
```

```
##  2. run of importance source...
```

```
##  3. run of importance source...
```

```
##  4. run of importance source...
```

```
##  239. run of importance source...
```

```
##  240. run of importance source...
```

```
##  241. run of importance source...
```

```
##  242. run of importance source...
```

```
##  243. run of importance source...
```

```
##  244. run of importance source...
```

```
##  245. run of importance source...
```

```
##  246. run of importance source...
```

```
##  247. run of importance source...
```

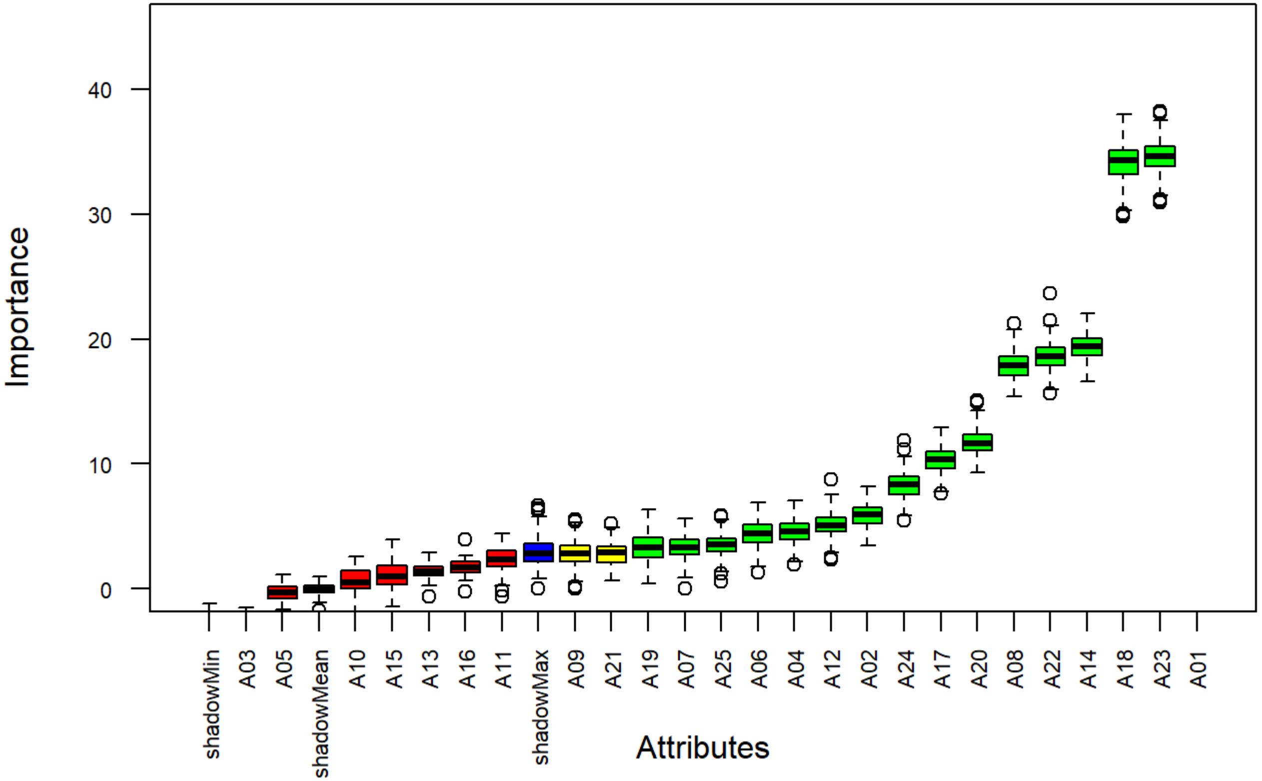
```
##  248. run of importance source...
```

```
##  249. run of importance source...
```

```
print(default_br)
```

```
## Boruta performed 249 iterations in 1.831641 mins.
##  16 attributes confirmed important: A01, A02, A04, A06, A07 and 11
## more;
##  7 attributes confirmed unimportant: A03, A05, A10, A11, A13 and 2
## more;
##  2 tentative attributes left: A09, A21;
```

```
plot(default_br, las = 2, cex.axis = 0.7, ylim = c(0, 45))
```



```
print(getSelectedAttributes(default_br))
```

```
##  [1] "A01" "A02" "A04" "A06" "A07" "A08" "A12" "A14" "A17" "A18" "A19" "A20"
## [13] "A22" "A23" "A24" "A25"
```

So, we have 16 Important attributes

Filtered features after removing highly correlated features

```
filtered_features <- getSelectedAttributes(default_br)  # Get the confirmed important features

# Select only the filtered features and the target variable
PD <- PD[, c(filtered_features, "Class")]
```

#Task 1.1: Proportion of phishing sites to legitimate sites

```
phish_prop <- PD %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  mutate(proportion = count / sum(count))

print("Proportion of phishing sites to legitimate sites:")
```

```
## [1] "Proportion of phishing sites to legitimate sites:"
```

```
print(phish_prop)
```

```
## # A tibble: 2 × 3
##   Class count proportion
##   <dbl> <int>      <dbl>
## 1     0  1228      0.614
## 2     1   772      0.386
```

Conclusion:The proportion of phishing sites to legitimate sites is approximately balanced in dataset. There are no attributes that need to be omitted outright.But for Better Model Performance , we have to remove some columns.

#Task1.2: Descriptions of predictor variables

```
numeric_cols <- sapply(PD, is.numeric)
numeric_data <- PD[, numeric_cols]

for (col in names(numeric_data)) {
  print(paste0("Summary for ", col, ":"))
  print(summary(numeric_data[, col]))
  cat("\n")
}
```

```
## [1] "Summary for A01:"
##       A01
##  Min.   : 4.00
##  1st Qu.:14.00
##  Median :29.00
##  Mean   :27.88
##  3rd Qu.:43.00
##  Max.   :49.00
##
## [1] "Summary for A02:"
##       A02
##  Min.   :  0.0000
##  1st Qu.:  0.0000
##  Median :  0.0000
##  Mean   :  0.2385
##  3rd Qu.:  0.0000
##  Max.   :105.0000
##
## [1] "Summary for A04:"
##       A04
##  Min.   :2.000
##  1st Qu.:2.000
##  Median :3.000
##  Mean   :2.729
##  3rd Qu.:3.000
##  Max.   :7.000
##
## [1] "Summary for A06:"
##       A06
##  Min.   :0.00
##  1st Qu.:0.00
##  Median :0.00
##  Mean   :0.14
##  3rd Qu.:0.00
##  Max.   :1.00
##
## [1] "Summary for A07:"
##       A07
```

```
##  Min.   :0.000  
##  1st Qu.:0.000  
##  Median :0.000  
##  Mean   :0.002  
##  3rd Qu.:0.000  
##  Max.   :1.000  
## 
## [1] "Summary for A08:"
##       A08        
##  Min.   :0.1707  
##  1st Qu.:0.6814  
##  Median :1.0000  
##  Mean   :0.8435  
##  3rd Qu.:1.0000  
##  Max.   :1.0000  
## 
## [1] "Summary for A12:"
##       A12       
##  Min.   : 82.0  
##  1st Qu.:232.0  
##  Median :232.0  
##  Mean   :320.5  
##  3rd Qu.:418.0  
##  Max.   :692.0  
## 
## [1] "Summary for A14:"
##       A14      
##  Min.   :0.00  
##  1st Qu.:0.00  
##  Median :0.00  
##  Mean   :0.14  
##  3rd Qu.:0.00  
##  Max.   :1.00  
## 
## [1] "Summary for A17:"
##       A17       
##  Min.   :0.000  
##  1st Qu.:1.000  
```

```
##  Median :1.000
##  Mean   :1.162
##  3rd Qu.:1.000
##  Max.   :4.000
##
## [1] "Summary for A18:"
##       A18
##  Min.   :   4.00
##  1st Qu.:  14.00
##  Median :  32.00
##  Mean   :  59.67
##  3rd Qu.:  88.00
##  Max.   :3364.00
##
## [1] "Summary for A19:"
##       A19
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.1015
##  3rd Qu.:0.0000
##  Max.   :1.0000
##
## [1] "Summary for A20:"
##       A20
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.2355
##  3rd Qu.:0.0000
##  Max.   :1.0000
##
## [1] "Summary for A22:"
##       A22
##  Min.   :0.01179
##  1st Qu.:0.05151
##  Median :0.05799
##  Mean   :0.05583
```

```
##  3rd Qu.:0.06250
##  Max.   :0.08003
##
## [1] "Summary for A23:"
##       A23
##  Min.   :   0.00
##  1st Qu.:   7.00
##  Median :  68.00
##  Mean   :  66.45
##  3rd Qu.: 104.00
##  Max.   :2602.00
##
## [1] "Summary for A24:"
##       A24
##  Min.   :0.000000
##  1st Qu.:0.005977
##  Median :0.079963
##  Mean   :0.251379
##  3rd Qu.:0.522907
##  Max.   :0.522907
##
## [1] "Summary for A25:"
##       A25
##  Min.   :0.000000
##  1st Qu.:0.000000
##  Median :0.000000
##  Mean   :0.000156
##  3rd Qu.:0.000000
##  Max.   :0.156000
##
## [1] "Summary for Class:"
##      Class
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.386
##  3rd Qu.:1.000
##  Max.   :1.000
```

Task 2:# Hence , we did Normalization, Endoing and Handling Missing Value in Preprocessing steps(its done eariler)

#Task3: Split the data into training and test sets

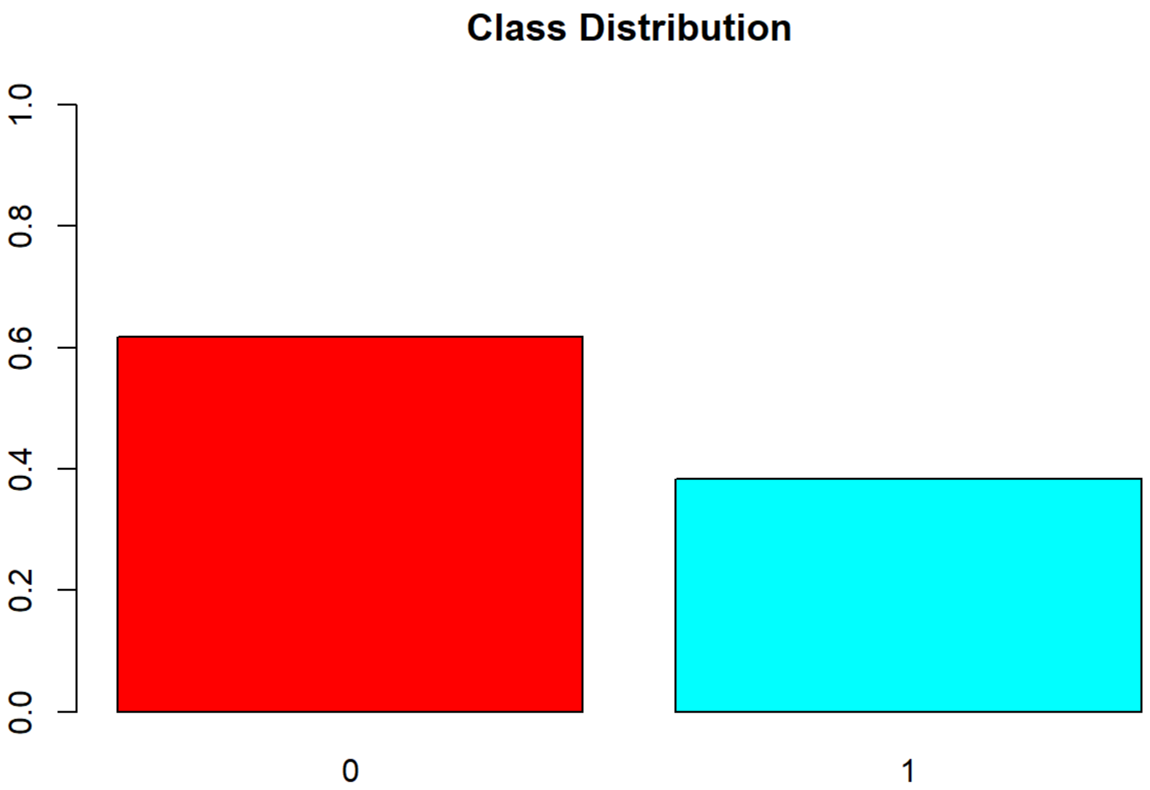
```
set.seed(100000)  # For reproducibility
trainIndex <- createDataPartition(PD$Class, p = 0.7, list = FALSE)
trainData <- PD[trainIndex, ]
testData <- PD[-trainIndex, ]
```

Check for class imbalance in the training set

```
table(trainData["Class"])
```

```
## Class
##   0   1
## 864 536
```

```
barplot(prop.table(table(trainData["Class"])),
        col = rainbow(2),
        ylim = c(0, 1),
        main = "Class Distribution")
```



conlcusion: It is approx balanced

Feature scaling

```
data.pre <- preProcess(trainData, method = "range")
trainData_scaled <- predict(data.pre, trainData)
testData_scaled <- predict(data.pre, testData)

# Append the target variable back to the scaled datasets
trainData_scaled$Class <- trainData$Class
testData_scaled$Class <- testData$Class
```

#Task4: Train Models : we are using Decision Tree, Naive Bayes,Bagging Model,xgboost,Random Forest for Training

```
library(rpart)

# Train a Decision Tree model
trainData_scaled$Class <- factor(trainData_scaled$Class)
tree <- rpart(Class ~ ., trainData_scaled)

# Train a Naive Bayes model
naive_bayes_model <- naiveBayes(Class ~ ., data = trainData_scaled)

# Train a Bagging model
bagging_model <- bagging(Class ~ ., data = trainData_scaled, nbagg = 25)


library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# Train an XGBoost model
xgb_train <- xgb.DMatrix(data = as.matrix(trainData_scaled[-ncol(trainData_scaled)]), label = as.numeric(trainData_scaled$Cl
ass) - 1)
xgb_test <- xgb.DMatrix(data = as.matrix(testData_scaled[-ncol(testData_scaled)]), label = as.numeric(testData_scaled$Class)
- 1)
xgb_params <- list(
  booster = "gbtree",
  eta = 0.01,
  max_depth = 8,
  gamma = 4,
  subsample = 0.75,
  colsample_bytree = 1,
  objective = "binary:logistic",
  eval_metric = "logloss"
)
xgb_model <- xgb.train(
  params = xgb_params,
  data = xgb_train,
  nrounds = 5000,
  verbose = 1
)
xgb_model
```

```
## ##### xgb.Booster
## raw: 16.1 Mb
## call:
##   xgb.train(params = xgb_params, data = xgb_train, nrounds = 5000,
##     verbose = 1)
## params (as set within xgb.train):
##   booster = "gbtree", eta = "0.01", max_depth = "8", gamma = "4", subsample = "0.75", colsample_bytree = "1", objective = 
"binary:logistic", eval_metric = "logloss", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 16
## niter: 5000
## nfeatures : 16
```

```
library(randomForest)
# Train a Random Forest model
random_forest_model = randomForest(x = trainData_scaled[-17],
                          y = trainData_scaled$Class,
                          ntree = 500, random_state = 42)
```

#Task 5: Make Predictions

```r
# Predict using the Decision Tree model
dt_pred <- predict(tree, testData_scaled, type = "class")

# Predict using the Naive Bayes model
nb_pred <- predict(naive_bayes_model, newdata = testData_scaled)

# Predict using the Bagging model
bagging_pred <- predict(bagging_model, newdata = testData_scaled)


# Predict using the XGBoost model
xgb_pred_prob <- predict(xgb_model, newdata = xgb_test)
xgb_pred <- ifelse(xgb_pred_prob > 0.5, 1, 0)


# Predict using the Random Forest model

rf_pred <- predict(random_forest_model, newdata = testData_scaled[-17], type = "class")
```

Convert Predictions to Factors

```
testData_scaled$Class <- as.factor(testData_scaled$Class)
levels(testData_scaled$Class) <- c("0", "1")

dt_pred <- as.factor(dt_pred)
levels(dt_pred) <- c("0", "1")

nb_pred <- as.factor(nb_pred)
levels(nb_pred) <- c("0", "1")

bagging_pred <- as.factor(bagging_pred)
levels(bagging_pred) <- c("0", "1")

xgb_pred <- as.factor(xgb_pred)
levels(xgb_pred) <- c("0", "1")

rf_pred <- as.factor(rf_pred)
levels(rf_pred) <- c("0", "1")
```

Confusion Matrix and Accuracy

```
# Decision Tree
dt_conf_matrix <- confusionMatrix(dt_pred, testData_scaled$Class)
dt_accuracy <- dt_conf_matrix$overall['Accuracy']

# Naive Bayes
nb_conf_matrix <- confusionMatrix(nb_pred, testData_scaled$Class)
nb_accuracy <- nb_conf_matrix$overall['Accuracy']

# Bagging
bagging_conf_matrix <- confusionMatrix(bagging_pred, testData_scaled$Class)
bagging_accuracy <- bagging_conf_matrix$overall['Accuracy']

# XGBoost
xgb_conf_matrix <- confusionMatrix(xgb_pred, testData_scaled$Class)
xgb_accuracy <- xgb_conf_matrix$overall['Accuracy']


# Random Forest
rf_conf_matrix <- confusionMatrix(rf_pred, testData_scaled$Class)
rf_accuracy <- rf_conf_matrix$overall['Accuracy']
```

ROC Curves and AUC

```
# Calculate probabilities
dt_prob <- predict(tree, newdata = testData_scaled, type = "prob")[,2]
nb_prob <- predict(naive_bayes_model, newdata = testData_scaled, type = "raw")[,2]
bagging_prob <- predict(bagging_model, newdata = testData_scaled, type = "prob")[,2]
rf_prob <- predict(random_forest_model, newdata = testData_scaled, type = "prob")[,2]
xgb_prob <- xgb_pred_prob


# ROC Curves and AUC
#roc
roc_dt <- roc(testData_scaled$Class, dt_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
roc_nb <- roc(testData_scaled$Class, nb_prob)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_bagging <- roc(testData_scaled$Class, bagging_prob)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
#roc_boosting <- roc(testData_scaled$Class, boosting_prob)
roc_rf <- roc(testData_scaled$Class, rf_prob)
```

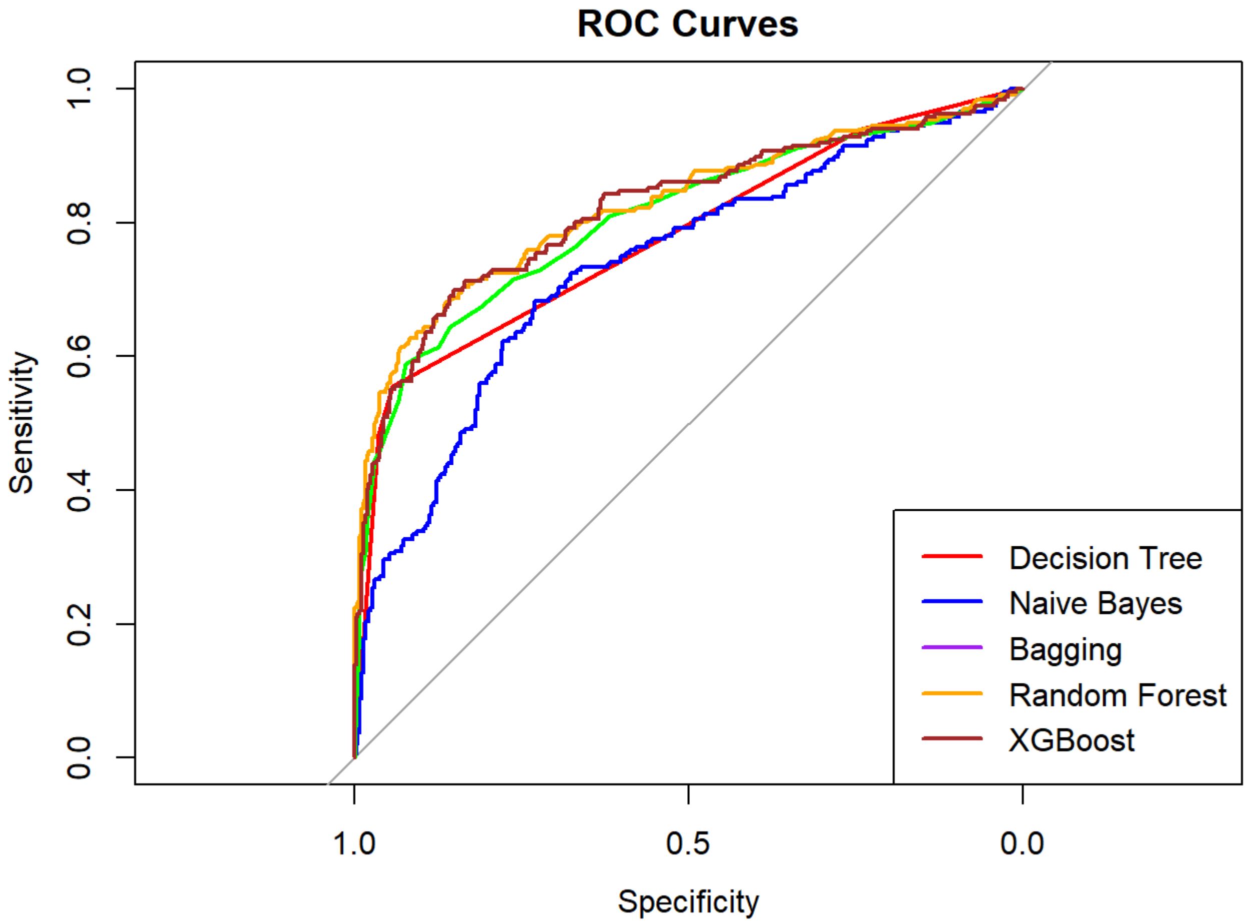
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_xgb <- roc(testData_scaled$Class, as.numeric(xgb_prob))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
#auc
auc_dt <- auc(roc_dt)
auc_nb <- auc(roc_nb)
auc_bagging <- auc(roc_bagging)
#auc_boosting <- auc(roc_boosting)
auc_rf <- auc(roc_rf)
auc_xgb <- auc(roc_xgb)

# Plot ROC Curves
plot(roc_dt, col = "red", main = "ROC Curves")
plot(roc_nb, col = "blue", add = TRUE)
plot(roc_bagging, col = "green", add = TRUE)
#plot(roc_boosting, col = "purple", add = TRUE)
plot(roc_rf, col = "orange", add = TRUE)
plot(roc_xgb, col = "brown", add = TRUE)
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging",  "Random Forest", "XGBoost"), col = c("red", "bl
ue", "purple", "orange", "brown"), lwd = 2)
```



#Task 7: Comparsion Table:

```
comparison_table <- data.frame(
  Model = c("Decision Tree", "Naive Bayes", "Bagging", "Random Forest", "XGBoost"),
  Accuracy = c(dt_accuracy, nb_accuracy, bagging_accuracy, rf_accuracy, xgb_accuracy),
  AUC = c(auc_dt, auc_nb, auc_bagging, auc_rf, auc_xgb)
)

print(comparison_table)
```

```
##           Model  Accuracy       AUC
## 1 Decision Tree 0.7750000 0.7787472
## 2   Naive Bayes 0.7066667 0.7363569
## 3       Bagging 0.7716667 0.8066737
## 4 Random Forest 0.8033333 0.8245134
## 5       XGBoost 0.7883333 0.8200433
```

```
# Determine the best classifier based on Accuracy
best_classifier <- comparison_table[which.max(comparison_table$Accuracy),]
print(best_classifier)
```

```
##           Model  Accuracy       AUC
## 4 Random Forest 0.8033333 0.8245134
```

So, we have best model Random Forest having 80.33% of accuracy and Auc of 82.4%

#task 8:

```
# Random Forest feature importance
rf_importance <- importance(random_forest_model)
rf_importance_df <- data.frame(Variable = rownames(rf_importance), Importance = rf_importance[, 1])
rf_importance_df <- rf_importance_df[order(-rf_importance_df$Importance), ]

# XGBoost feature importance
xgb_importance <- xgb.importance(model = xgb_model)
xgb_importance_df <- as.data.frame(xgb_importance)
xgb_importance_df <- xgb_importance_df[order(-xgb_importance_df$Gain), ]

# Combine and compare importance
combined_importance <- merge(rf_importance_df, xgb_importance_df, by.x = "Variable", by.y = "Feature", all = TRUE)
combined_importance <- combined_importance[order(-combined_importance$Importance), ]

print("Combined Feature Importance:")
```

```
## [1] "Combined Feature Importance:"
```

```
print(combined_importance)
```

```
##    Variable  Importance        Gain      Cover   Frequency
## 1       A01 134.5664315 0.221110557 0.15908939 0.122778675
## 14      A23 104.3999472 0.240806691 0.14490424 0.149307032
## 13      A22  95.4838928 0.179962198 0.20232918 0.242156279
## 10      A18  92.2761111 0.127467150 0.14819042 0.159510246
## 6       A08  47.5656640 0.077633990 0.10759745 0.102542301
## 15      A24  33.4072740 0.038428252 0.05239901 0.059518748
## 7       A12  31.6735483 0.040453855 0.06374637 0.066831052
## 8       A14  23.1490889 0.024195677 0.02760764 0.018195732
## 12      A20  15.0034624 0.014415689 0.02276931 0.017090383
## 9       A17  13.9131102 0.010640978 0.01717270 0.018110705
## 3       A04  10.4845937 0.007722602 0.01032325 0.011393589
## 2       A02   9.6814630 0.010814062 0.02874605 0.019811241
## 4       A06   7.6127130 0.004682231 0.01236085 0.009523000
## 11      A19   6.8552858 0.001666068 0.00276413 0.003231018
## 5       A07   1.0116396          NA         NA          NA
## 16      A25   0.9761198          NA         NA          NA
```

```
# Identify least important features
least_important_rf <- tail(rf_importance_df, 5)
least_important_xgb <- tail(xgb_importance_df, 5)

print("Least Important Features in Random Forest:")
```

```
## [1] "Least Important Features in Random Forest:"
```

```
print(least_important_rf)
```

```
##     Variable Importance
## A02      A02  9.6814630
## A06      A06  7.6127130
## A19      A19  6.8552858
## A07      A07  1.0116396
## A25      A25  0.9761198
```

```
print("Least Important Features in XGBoost:")
```

```
## [1] "Least Important Features in XGBoost:"
```

```
print(least_important_xgb)
```

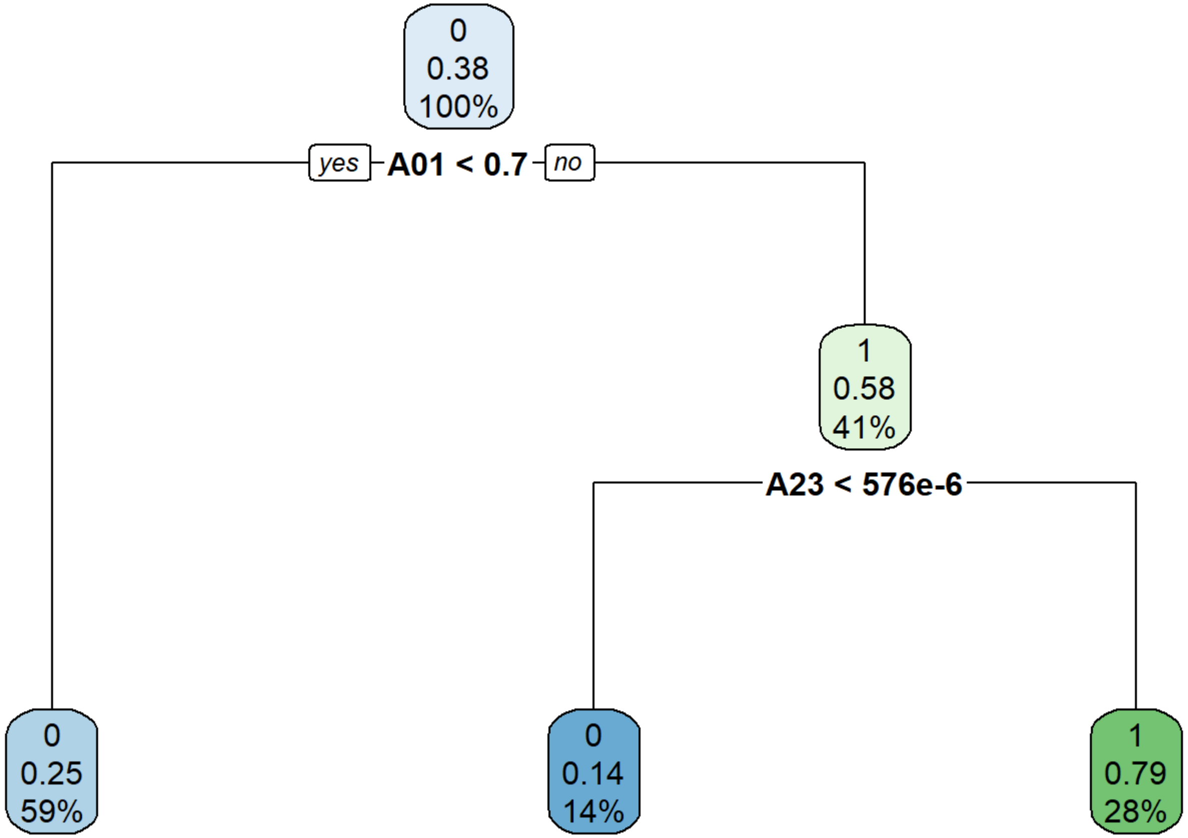
```
##    Feature        Gain      Cover   Frequency
## 10     A02 0.010814062 0.02874605 0.019811241
## 11     A17 0.010640978 0.01717270 0.018110705
## 12     A04 0.007722602 0.01032325 0.011393589
## 13     A06 0.004682231 0.01236085 0.009523000
## 14     A19 0.001666068 0.00276413 0.003231018
```

Conclusion:

Features to Consider for Omission: A02: Identified as low importance in both Random Forest and XGBoost. A06: Identified as low importance in both Random Forest and XGBoost. A19: Identified as low importance in both Random Forest and XGBoost. A07: Identified as low importance in Random Forest. A25: Identified as low importance in Random Forest. A17: Identified as low importance in XGBoost. A04: Identified as low importance in XGBoost. So, we can remove or not consider these features for Building our model.

#task:9

```
library(rpart.plot)
# Create a simple Decision Tree model
simple_tree_model <- rpart(Class ~ ., data = trainData_scaled, control = rpart.control(maxdepth = 2))
rpart.plot(simple_tree_model)
```




```
# Predict using the simple tree model
simple_tree_pred <- predict(simple_tree_model, newdata = testData_scaled, type = "class")

# Evaluate performance
simple_tree_conf_matrix <- confusionMatrix(simple_tree_pred, testData_scaled$Class)
simple_tree_accuracy <- simple_tree_conf_matrix$overall['Accuracy']
simple_tree_roc <- roc(testData_scaled$Class, as.numeric(predict(simple_tree_model, newdata = testData_scaled)[, 2]))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
simple_tree_auc <- auc(simple_tree_roc)

print(paste("Simple Tree Accuracy:", simple_tree_accuracy))
```

```
## [1] "Simple Tree Accuracy: 0.768333333333333"
```

```
print(paste("Simple Tree AUC:", simple_tree_auc))
```

```
## [1] "Simple Tree AUC: 0.753183786552431"
```

```
# Compare with other models
comparison_table <- rbind(comparison_table, data.frame(
  Model = "Simple Tree",
  Accuracy = simple_tree_accuracy,
  AUC = simple_tree_auc
))

print(comparison_table)
```

```
##                     Model  Accuracy       AUC
## 1           Decision Tree 0.7750000 0.7787472
## 2             Naive Bayes 0.7066667 0.7363569
## 3                 Bagging 0.7716667 0.8066737
## 4           Random Forest 0.8033333 0.8245134
## 5                 XGBoost 0.7883333 0.8200433
## Accuracy      Simple Tree 0.7683333 0.7531838
```

#task :10 Create the Best Tree-Based Classifier Model: Decision Tree with max depth of 2 Features: Key features identified from previous analyses

```
# Cross-validation and parameter tuning for Random Forest
control <- trainControl(method = "cv", number = 5)
tunegrid <- expand.grid(.mtry = c(2, 4, 6, 8, 10))

set.seed(100000)
rf_optimized <- train(Class ~ ., data = trainData_scaled, method = "rf", trControl = control, tuneGrid = tunegrid)

# Get the best model
best_rf_model <- rf_optimized$finalModel

# Predict using the optimized Random Forest model
best_rf_pred <- predict(best_rf_model, newdata = testData_scaled)

# Evaluate performance
best_rf_conf_matrix <- confusionMatrix(best_rf_pred, testData_scaled$Class)
best_rf_accuracy <- best_rf_conf_matrix$overall['Accuracy']
best_rf_roc <- roc(testData_scaled$Class, as.numeric(predict(best_rf_model, newdata = testData_scaled, type = "prob")[, 2]))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
best_rf_auc <- auc(best_rf_roc)

print(paste("Optimized Random Forest Accuracy:", best_rf_accuracy))
```

```
## [1] "Optimized Random Forest Accuracy: 0.806666666666667"
```

```
print(paste("Optimized Random Forest AUC:", best_rf_auc))
```

```
## [1] "Optimized Random Forest AUC: 0.817505587632706"
```

```
# Compare with other models
comparison_table <- rbind(comparison_table, data.frame(
  Model = "Optimized Random Forest",
  Accuracy = best_rf_accuracy,
  AUC = best_rf_auc
))

print(comparison_table)
```

```
##                             Model  Accuracy       AUC
## 1                   Decision Tree 0.7750000 0.7787472
## 2                     Naive Bayes 0.7066667 0.7363569
## 3                         Bagging 0.7716667 0.8066737
## 4                   Random Forest 0.8033333 0.8245134
## 5                         XGBoost 0.7883333 0.8200433
## Accuracy              Simple Tree 0.7683333 0.7531838
## Accuracy1 Optimized Random Forest 0.8066667 0.8175056
```

#Conclusion: The Simple Decision Tree model offers a terrific stability between interpretability and performance, making it a suitable desire for guide category obligations while remaining aggressive with extra complicated models.

#Task:11 Implement an Artificial Neural Network (ANN) Classifier

```
library(nnet)
# Install caret package if not already installed
if (!requireNamespace("caret", quietly = TRUE)) {
  install.packages("caret")
}

# Load caret package
library(caret)


# Convert the target variable to a factor
trainData_scaled$Class <- as.factor(trainData_scaled$Class)
testData_scaled$Class <- as.factor(testData_scaled$Class)

# Train the ANN model
set.seed(100000)
ann_model <- nnet(Class ~ ., data = trainData_scaled, size = 10, maxit = 200, linout = FALSE)
```

```
## # weights:  181
## initial  value 939.317518
## iter  10 value 747.026626
## iter  20 value 688.779080
## iter  30 value 653.750915
## iter  40 value 625.469192
## iter  50 value 603.765174
## iter  60 value 586.501850
## iter  70 value 565.600692
## iter  80 value 551.872205
## iter  90 value 537.415471
## iter 100 value 523.321775
## iter 110 value 516.221882
## iter 120 value 509.743187
## iter 130 value 503.913511
## iter 140 value 493.252306
## iter 150 value 483.550976
## iter 160 value 477.972906
## iter 170 value 473.789699
## iter 180 value 470.323208
## iter 190 value 468.046938
## iter 200 value 466.754726
## final  value 466.754726
## stopped after 200 iterations
```

```
# Predict using the ANN model
ann_pred_prob <- predict(ann_model, newdata = testData_scaled, type = "raw")
ann_pred <- ifelse(ann_pred_prob > 0.5, 1, 0)

# Evaluate performance
ann_conf_matrix <- confusionMatrix(as.factor(ann_pred), testData_scaled$Class)
ann_accuracy <- ann_conf_matrix$overall['Accuracy']
ann_roc <- roc(testData_scaled$Class, as.numeric(ann_pred_prob))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
ann_auc <- auc(ann_roc)

print(paste("ANN Accuracy:", ann_accuracy))
```

```
## [1] "ANN Accuracy: 0.705"
```

```
print(paste("ANN AUC:", ann_auc))
```

```
## [1] "ANN AUC: 0.713005215123859"
```

```
# Compare with other models
comparison_table <- rbind(comparison_table, data.frame(
  Model = "ANN",
  Accuracy = ann_accuracy,
  AUC = ann_auc
))

print(comparison_table)
```

```
##                              Model  Accuracy       AUC
## 1                    Decision Tree 0.7750000 0.7787472
## 2                      Naive Bayes 0.7066667 0.7363569
## 3                          Bagging 0.7716667 0.8066737
## 4                    Random Forest 0.8033333 0.8245134
## 5                          XGBoost 0.7883333 0.8200433
## Accuracy               Simple Tree 0.7683333 0.7531838
## Accuracy1 Optimized Random Forest 0.8066667 0.8175056
## Accuracy2                      ANN 0.7050000 0.7130052
```

#Analysis: The ANN classifier done an accuracy of 0.7050 and an AUC of 0.7130. These results are decrease than the ones of the Random Forest and XGBoost models. The complexity and intensity of the neural community version won't be completely utilized given the dimensions and nature of the dataset. Additionally, neural networks normally require more statistics to gain better overall performance, that could give an explanation for why less complicated models carried out higher in this example. Moreover, the ANN might need tunning and a more big hyperparameter tunning to improve its performance.

#Task 12: Implement a New Classifier (Support Vector Machine with Radial Basis Function Kernel) # Data Preprocessing : It will be same as prev models

```
# Install and load required packages
if (!requireNamespace("e1071", quietly = TRUE)) {
  install.packages("e1071")
}

library(e1071)
# Ensure the caret package is installed and loaded
if (!requireNamespace("caret", quietly = TRUE)) {
  install.packages("caret")
}




# Train the SVM model with RBF kernel
set.seed(100000)
# Load necessary library
library(e1071)

# Define and train the SVM model
svm_model <- svm(Class ~ ., data = trainData_scaled, probability = TRUE)

# Predict using the SVM model
svm_pred <- predict(svm_model, newdata = testData_scaled, probability = TRUE)
svm_prob <- attr(svm_pred, "probabilities")[, 2]

# Calculate performance metrics
svm_conf_matrix <- confusionMatrix(as.factor(svm_pred), as.factor(testData_scaled$Class))
svm_roc <- roc(testData_scaled$Class, svm_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls > cases
```

```
svm_auc <- auc(svm_roc)

# Print results
svm_accuracy <- svm_conf_matrix$overall['Accuracy']
print(paste("SVM Accuracy:", svm_accuracy))
```

```
## [1] "SVM Accuracy: 0.776666666666667"
```

```
print(paste("SVM AUC:", svm_auc))
```

```
## [1] "SVM AUC: 0.80007333767927"
```

```
# Compare with other models
comparison_table <- rbind(comparison_table, data.frame(
  Model = "SVM",
  Accuracy = svm_accuracy,
  AUC = svm_auc
))

print(comparison_table)
```

```
##                             Model  Accuracy       AUC
## 1                   Decision Tree 0.7750000 0.7787472
## 2                     Naive Bayes 0.7066667 0.7363569
## 3                         Bagging 0.7716667 0.8066737
## 4                   Random Forest 0.8033333 0.8245134
## 5                         XGBoost 0.7883333 0.8200433
## Accuracy              Simple Tree 0.7683333 0.7531838
## Accuracy1 Optimized Random Forest 0.8066667 0.8175056
## Accuracy2                     ANN 0.7050000 0.7130052
## Accuracy3                     SVM 0.7766667 0.8000733
```

#Conclusion: The SVM classifier carried out an accuracy of zero.7767 and an AUC of 0.8001. While the SVM performed better than some Models like Naive Bayes and ANN, it did no longer surpass the performance of Random Forest, Optimized Random Forest, or XGBoost in terms of accuracy. However, it’s far competitive in phrases of AUC, indicating a great potential to distinguish between phishing and valid sites.

Description of the SVM Model:

SVm is a robust model,which selects the hyperplane that maximizes the margin between the nearest data points of each class, known as support vectors. It having Kernal which helps in Hyper Parameter tuning to acheive good accuracy. #Final Conclusion: But for this particular Dataset,Optimized Random Forest , and Xgboost is working well.