USE CASE STUDY REPORT

Group No.: Group 04

Student Names: Ronit Mankad and Shashank Nukala

Executive Summary:

Identifying phishing websites across different platforms still proves to be a major challenge in the industry. Websites can be classified by monitoring a variety of different indicators. Example: does the website uses https or not, does the website uses an external favicon etc. As the number of indicators increase, they introduce more complexity to the classification process. We propose a solution which uses machine learning techniques like KNN, Classification Trees, Naive Bayes, Logistic regression, Neural Nets, and Support Vector Machines to model and classify the data. A well trained and generalized model will be able to classify websites with a reasonable accuracy. Furthermore, we will also implement dimension reduction techniques like Principal Component Analysis to streamline the dataset. The models will be tested for their accuracy and robustness using evaluation metrics like Lift Chart and ROC curves. The goal of study is to successfully identify a phishing website given its description and key attributes.

I. Background and Introduction

Background

Phishing is a form of cyber-attack that utilizes counterfeit websites to steal sensitive user information such as account login credentials, credit card numbers, etc. Throughout the world, phishing attacks continue to evolve and gain momentum. In June 2018, the Anti-Phishing Working Group (APWG) reported as many as 51,401 unique phishing websites.

Phishing is an example of social engineering techniques being used to deceive users. Users are often lured by communications purporting to be from trusted parties such as social web sites, auction sites, banks, online payment processors or IT administrators. Attempts to deal with phishing incidents include legislation, user training, public awareness, and technical security measures (the latter being due to phishing attacks frequently exploiting weaknesses in current web security).

Cyber Security researchers have made great strides in developing tools and methods to detect and report phishing websites. The advent of machine learning methods has also helped in developing applications which can automatically detect phishing attacks.

The Problem

Identifying phishing websites across different platforms still proves to be a major challenge in the industry. Websites can be classified by monitoring a variety of different indicators. Example: does the website uses https or not, does the website uses an external favicon etc. As the number of indicators increase, they introduce more complexity to the classification process.

The goal of this study

The goal of study is to successfully identify a phishing website given its description and key attributes.

Our Solution

We propose a solution which uses machine learning techniques like KNN, Classification Trees, Naive Bayes, Logistic regression, Neural Nets, and Support Vector Machines to model and classify the data. A well trained and generalized model will be able to classify websites with a reasonable accuracy. Furthermore, we will also implement dimension reduction techniques like Principal Component Analysis to streamline the dataset. The models will be tested for their accuracy and robustness using evaluation metrics like Lift Chart and ROC curve.

II. Data Exploration and Visualization

Data Description

This dataset contains 48 features extracted from 5000 phishing webpages and 5000 legitimate webpages, which were downloaded from January to May 2015 and from May to June 2017. An improved feature extraction technique is employed by leveraging the browser automation framework (i.e., Selenium WebDriver), which is more precise and robust compared to parsing approach based on regular expressions.

Data Visualization

We analyzed the distributions of some attributes to get an idea about the distribution of the data.

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
NumDots	1	10000	2.45	1.35	2.00	2.29	1.48	1	21	20	3.28	23.17	0.01
SubdomainLevel	2	10000	0.59	0.75	1.00	0.52	1.48	0	14	14	4.12	40.00	0.01
PathLevel	3	10000	3.30	1.86	3.00	3.15	1.48	0	18	18	1.26	4.09	0.02
UrlLength	4	10000	70.26	33.37	62.00	65.71	25.20	12	253	241	1.70	4.18	0.33
NumDash	5	10000	1.82	3.11	0.00	1.09	0.00	0	55	55	2.79	14.71	0.03
NumDashInHostname	6	10000	0.14	0.55	0.00	0.00	0.00	0	9	9	5.99	46.82	0.01
AtSymbol	7	10000	0.00	0.02	0.00	0.00	0.00	0	1	1	57.70	3327.67	0.00
TildeSymbol	8	10000	0.01	0.11	0.00	0.00	0.00	0	1	1	8.56	71.33	0.00
NumUnderscore	9	10000	0.32	1.11	0.00	0.05	0.00	0	18	18	5.53	41.20	0.01
NumPercent	10	10000	0.07	0.62	0.00	0.00	0.00	0	19	19	14.30	277.58	0.01
NumQueryComponents	11	10000	0.46	1.34	0.00	0.11	0.00	0	23	23	5.12	40.12	0.01
NumAmpersand	12	10000	0.28	1.12	0.00	0.00	0.00	0	22	22	6.42	56.60	0.01
NumHash	13	10000	0.00	0.05	0.00	0.00	0.00	0	1	1	20.78	429.70	0.00
NumNumericChars	14	10000	5.81	9.62	2.00	3.72	2.97	0	111	111	3.46	17.55	0.10
NoHttps	15	10000	0.99	0.11	1.00	1.00	0.00	0	1	1	-9.29	84.28	0.00
RandomString	16	10000	0.53	0.50	1.00	0.53	0.00	0	1	1	-0.10	-1.99	0.00
IpAddress	17	10000	0.02	0.13	0.00	0.00	0.00	0	1	1	7.43	53.15	0.00
DomainInSubdomains	18	10000	0.02	0.15	0.00	0.00	0.00	0	1	1	6.48	40.06	0.00
DomainInPaths	19	10000	0.43	0.49	0.00	0.41	0.00	0	1	1	0.29	-1.92	0.00
HostnameLength	20	10000	18.82	8.12	18.00	17.87	5.93	4	137	133	3.23	20.79	0.08
PathLength	21	10000	35.56	24.59	30.00	32.61	20.76	0	161	161	1.28	2.07	0.25
QueryLength	22	10000	8.61	24.31	0.00	1.91	0.00	0	188	188	3.80	16.29	0.24
DoubleSlashInPath	23	10000	0.00	0.03	0.00	0.00	0.00	0	1	1	33.28	1105.89	0.00
NumSensitiveWords	24	10000	0.11	0.37	0.00	0.00	0.00	0	3	3	3.71	14.74	0.00
EmbeddedBrandName	25	10000	0.06	0.23	0.00	0.00	0.00	0	1	1	3.82	12.57	0.00
PctExtHyperlinks	26	10000	0.24	0.34	0.07	0.18	0.11	0	1	1	1.37	0.31	0.00
PctExtResourceUrls	27	10000	0.39	0.39	0.25	0.37	0.37	0	1	1	0.54	-1.34	0.00
ExtFavicon	28	10000	0.17	0.37	0.00	0.08	0.00	0	1	1	1.78	1.18	0.00
InsecureForms	29	10000	0.84	0.36	1.00	0.93	0.00	0	1	1	-1.90	1.59	0.00
RelativeFormAction	30	10000	0.25	0.43	0.00	0.19	0.00	0	1	1	1.16	-0.65	0.00
ExtFormAction	31	10000	0.10	0.30	0.00	0.00	0.00	0	1	1	2.63	4.93	0.00
AbnormalFormAction	32	10000	0.06	0.23	0.00	0.00	0.00	0	1	1	3.80	12.42	0.00
PctNullSelfRedirectHyperlinks	33	10000	0.14	0.31	0.00	0.05	0.00	0	1	1	2.25	3.26	0.00
FrequentDomainNameMismatch	34	10000	0.22	0.41	0.00	0.14	0.00	0	1	1	1.39	-0.08	0.00
FakeLinkInStatusBar	35	10000	0.01	0.07	0.00	0.00	0.00	0	1	1	13.37	176.79	0.00
RightClickDisabled	36	10000	0.01	0.12	0.00	0.00	0.00	0	1	1	8.27	66.43	0.00
PopUpWindow	37	10000	0.00	0.07	0.00	0.00	0.00	0	1	1	14.18	199.05	0.00
SubmitInfoToEmail	38	10000	0.13	0.33	0.00	0.04	0.00	0	1	1	2.22	2.91	0.00
IframeOrFrame	39	10000	0.34	0.47	0.00	0.30	0.00	0	1	1	0.68	-1.54	0.00
MissingTitle	40	10000	0.03	0.18	0.00	0.00	0.00	0	1	1	5.30	26.08	0.00
ImagesOnlyInForm	41	10000	0.03	0.17	0.00	0.00	0.00	0	1	1	5.47	27.92	0.00
SubdomainLevelRT	42	10000	0.96	0.25	1.00	1.00	0.00	-1	1	2	-6.25	40.90	0.00
UrlLengthRT	43	10000	0.02	0.82	0.00	0.03	1.48	-1	1	2	-0.04	-1.51	0.01
PctExtResourceUrlsRT	44	10000	0.35	0.89	1.00	0.44	0.00	-1	1	2	-0.75	-1.31	0.01
AbnormalExtFormActionR	45	10000	0.79	0.52	1.00	0.93	0.00	-1	1	2	-2.49	5.09	0.01
ExtMetaScriptLinkRT	46	10000	0.17	0.76	0.00	0.22	1.48	-1	1	2	-0.30	-1.20	0.01
PctExtNullSelfRedirectHyperlinksRT	47	10000	0.31	0.90	1.00	0.39	0.00	-1	1	2	-0.66	-1.44	0.01

Figure 1: Data Summary

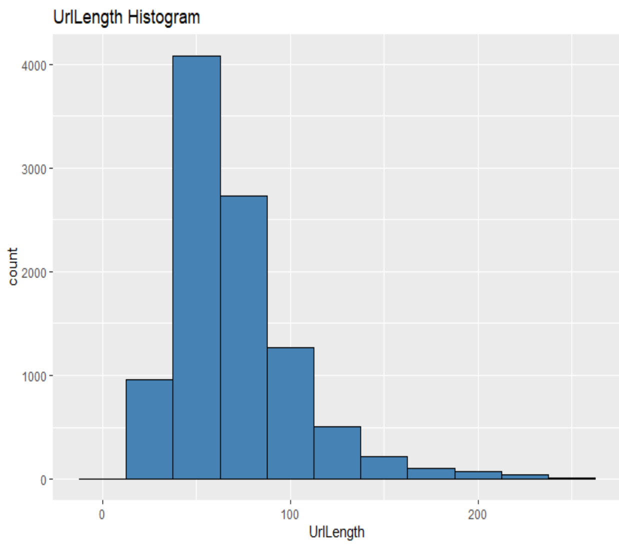


Figure 2: UrlLength Histogram

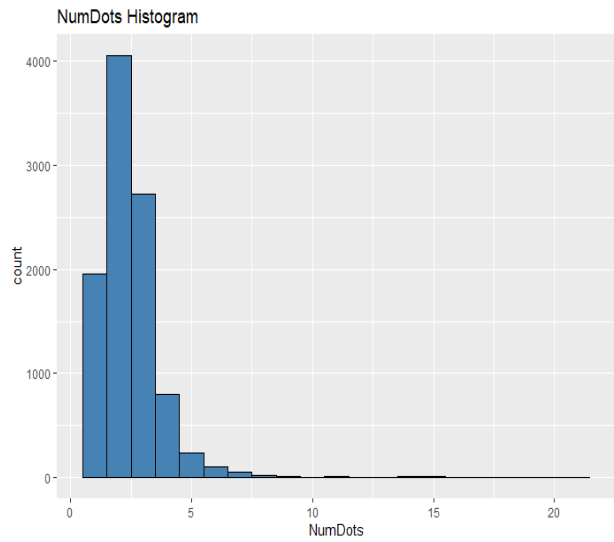


Figure 3: NumDots Histogram

An interesting observation we found was that all websites which contained an IP Address in their URL where phishing websites.

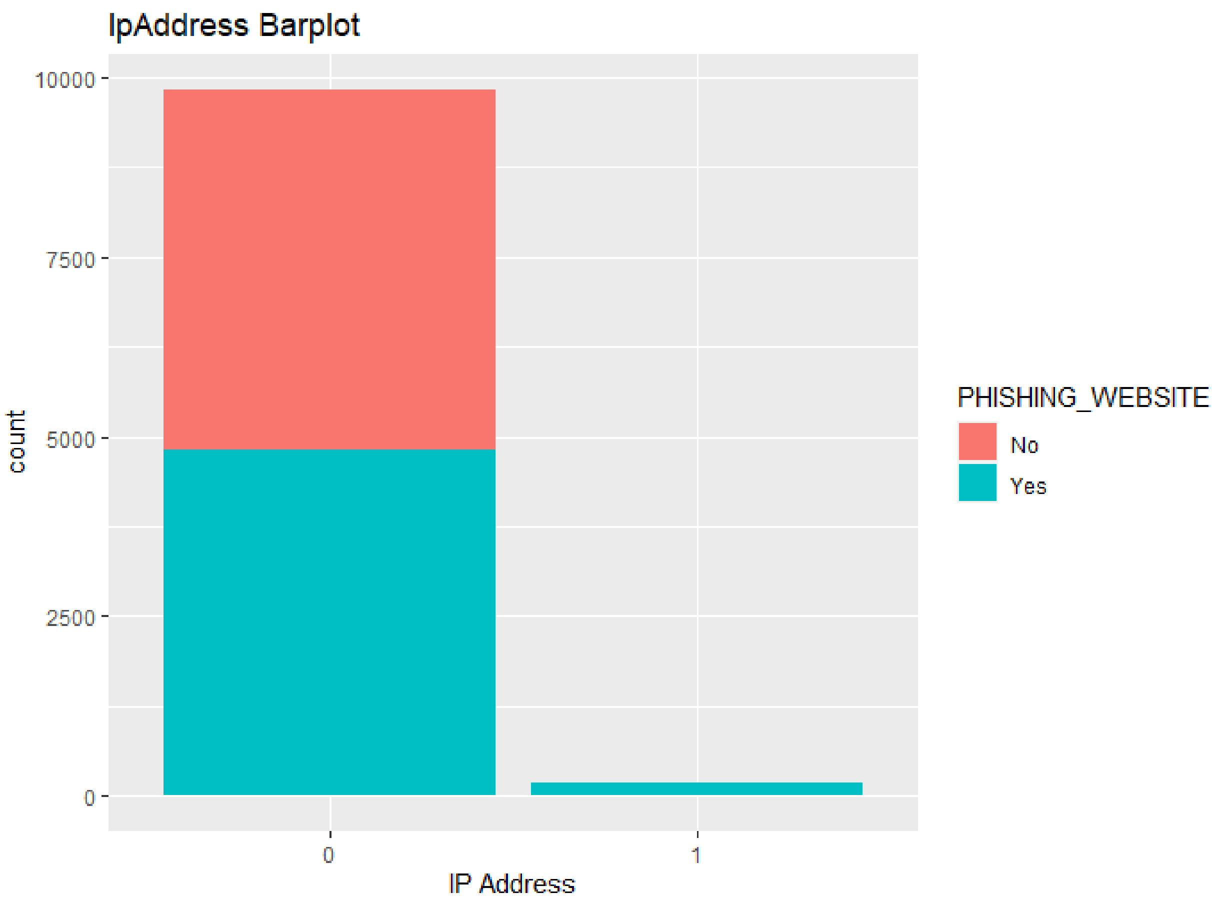


Figure 4: IPAddress Barplot

The data also contained a lot of correlated attributes as we can see from the correlation matrix.

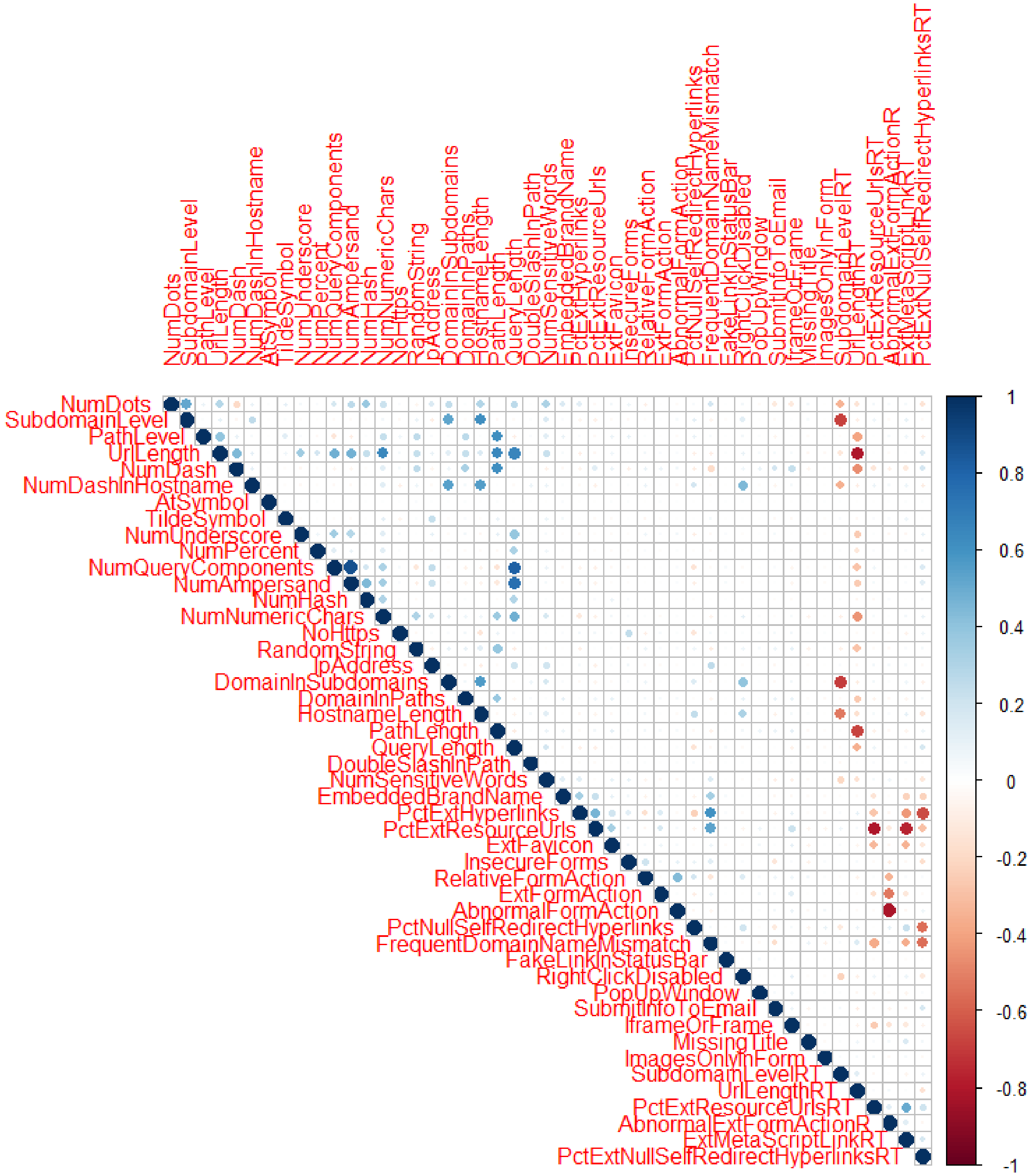


Figure 5: Data Correlation Plot

It looks like there are several attributes which are either highly positively or highly negatively correlated with each other. This is a redundant dataset and will have to undergo cleaning and transformation operations to be suitable for data mining algorithms.

III. Data Preparation and Preprocessing

Data Summary

Our dataset is mostly comprised of numerical attributes with a few categorial attributes. The PHISHING_WEBSITE column represents the response variable which is a binary categorical variable.

Variable Selection

Since our dataset comprises of many correlated attributes, it becomes a necessity to select only linearly independent attributes. However, selecting and discarding them requires domain knowledge in cyber security. To tackle this, we perform principal component analysis on the data to reduce it into independent and information dense components.

Preprocessing and Dimensionality Reduction

Our dataset has 48 attributes all in different scales. We first normalize our entire dataset of 10,000 rows using Min-Max Normalization. We then use Principal Component Analysis to reduce the dimensions of our data from 48 attributes to just 13 principal components which capture 61% of the data's variance. The number of components were selected by doing scree plot and parallel analysis on the data.

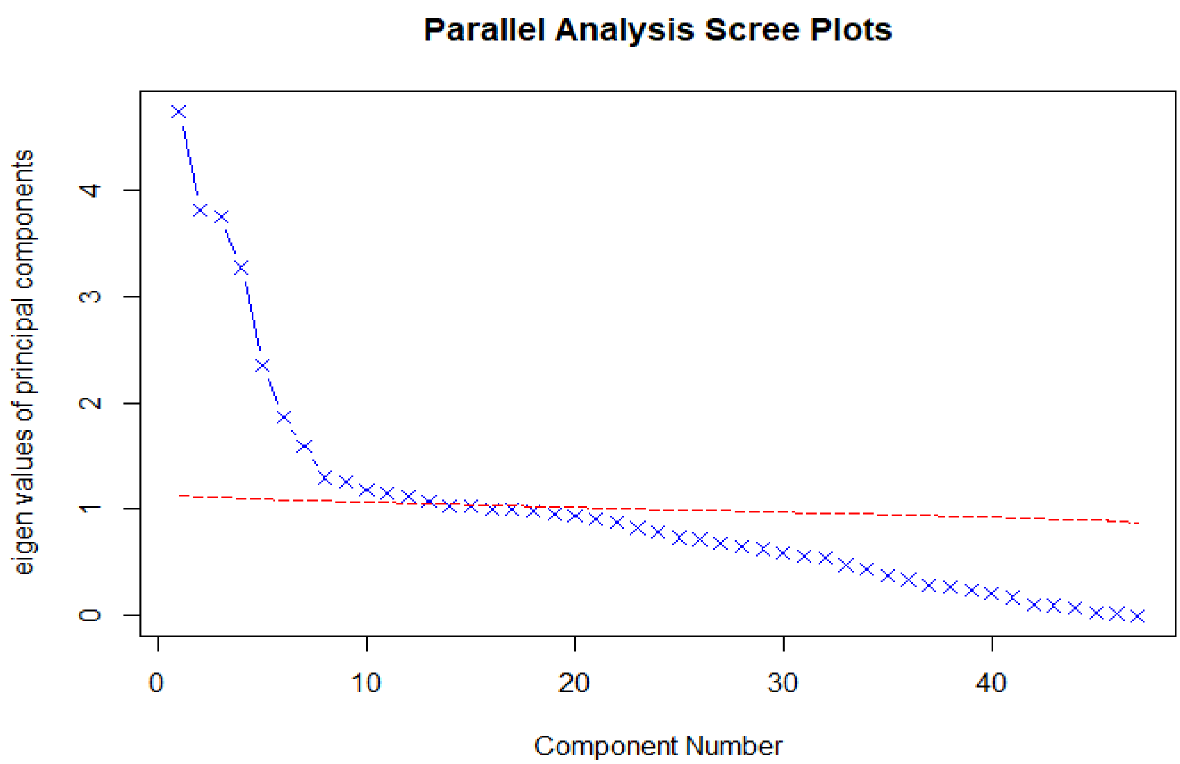


Figure 6: Scree Plot, we can see the appropriate number of principal components to use will be 13

After performing PCA, we plot the correlation matrix.

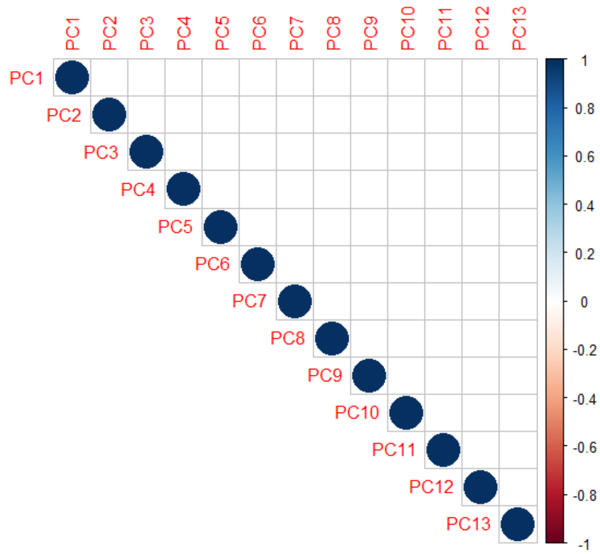


Figure 7: Correlation Plot for PCA data

We then split the data into training and validations sets with a 60/40 split. The training set contains 6000 rows and validation set contains 4000 rows.

IV. Data Mining Techniques and Implementation

Important: *Assume the data used in the following algorithms to be the normalized dataset with PCA performed on it (with 13 predictor variables) unless otherwise mentioned.*

We then use the following algorithms on for the classification –
1. K Nearest Neighbors
2. Logistic Regression
3. Naïve Bayes Classifier
4. Classification Tree
5. Random Forests
6. Support Vector Machine
7. Artificial Neural Network

Flowchart

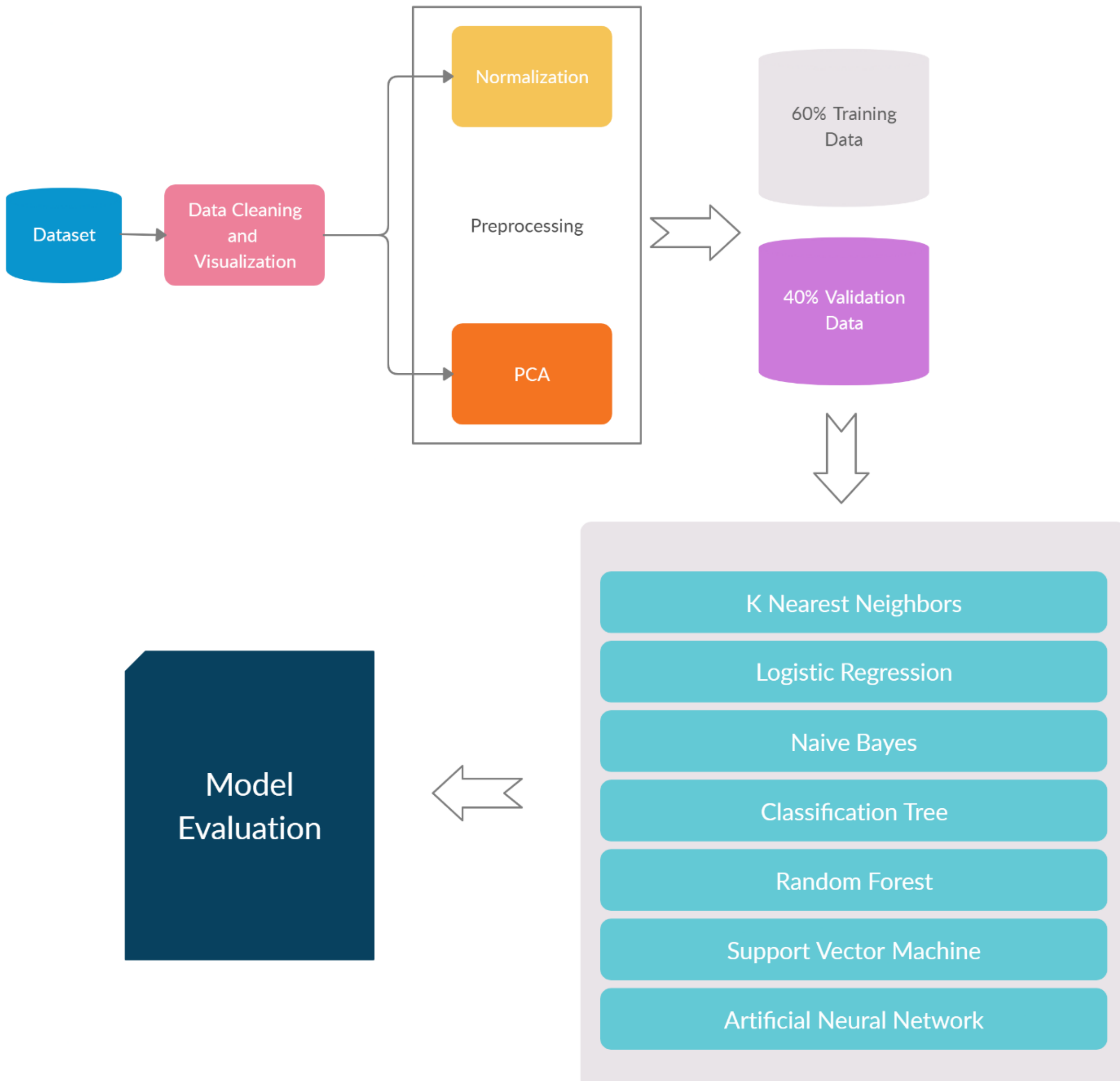


Figure 8: Data Mining Process Flow Chart

V. Performance Evaluation

We applied the above-mentioned algorithms on the dataset and evaluated them using their ROC Curves, Area Under Curve Values, Lift Charts and Accuracy on the validation set.

The following classification tree and neural network architectures performed the best on the data:

Classification Tree

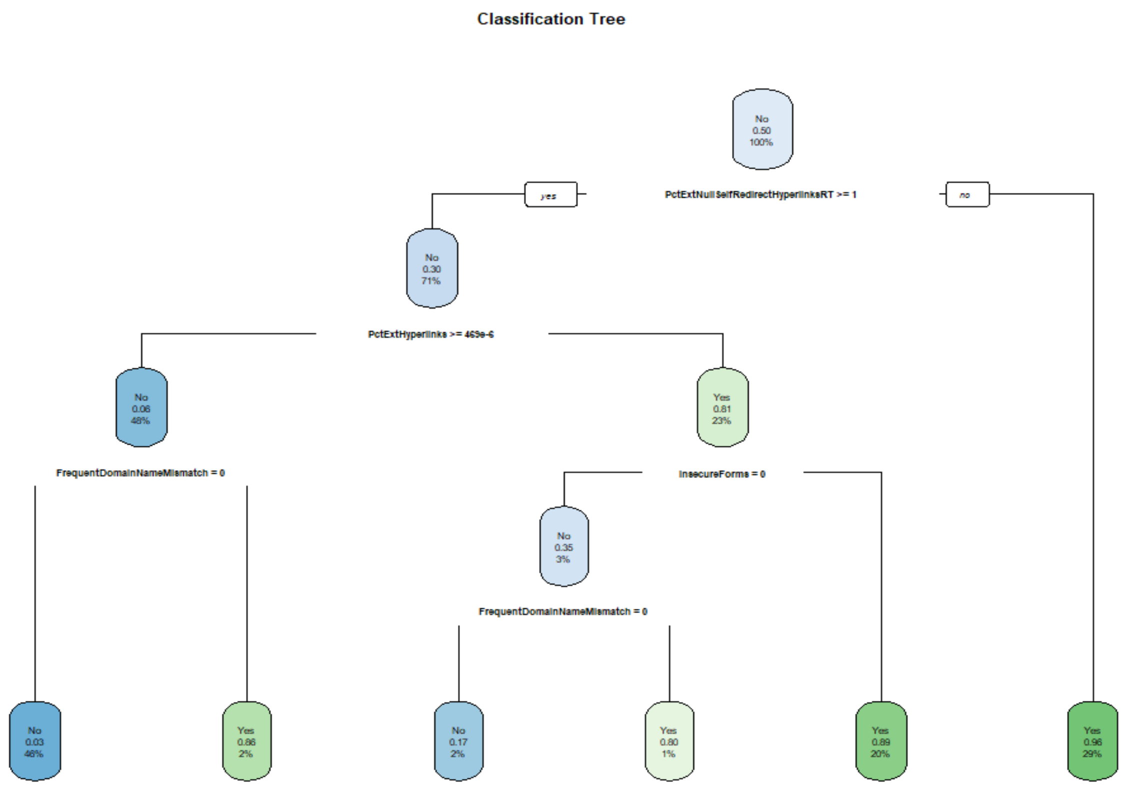


Figure 9: Classification Tree

Classification Tree (PCA)

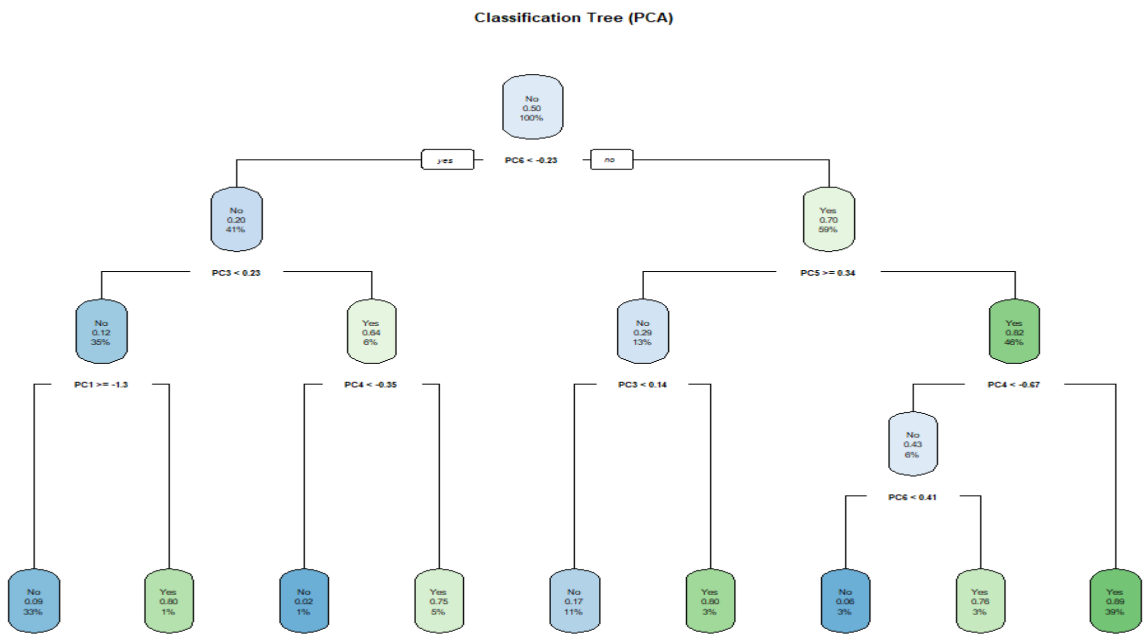


Figure 10: Classification Tree (PCA)

Neural Network (PCA)

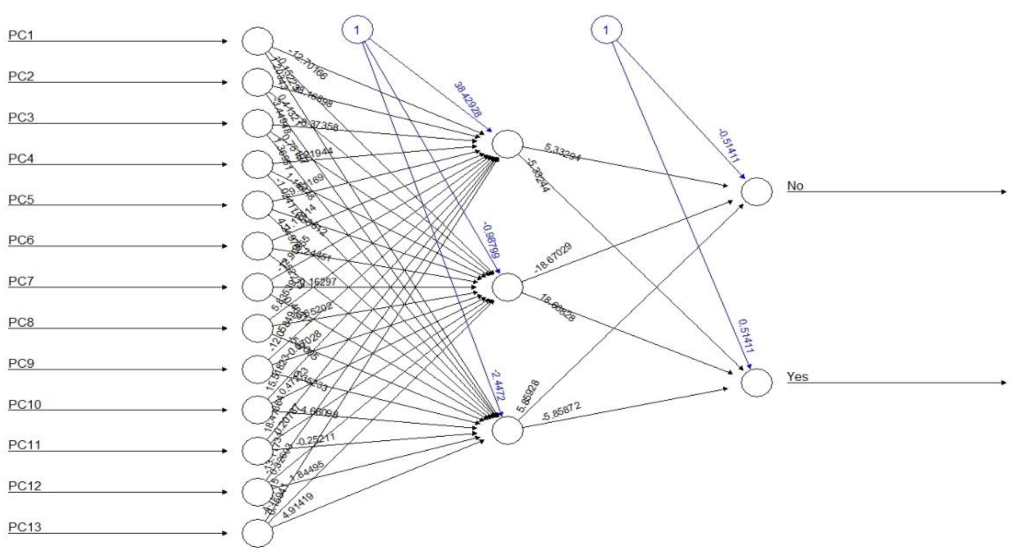
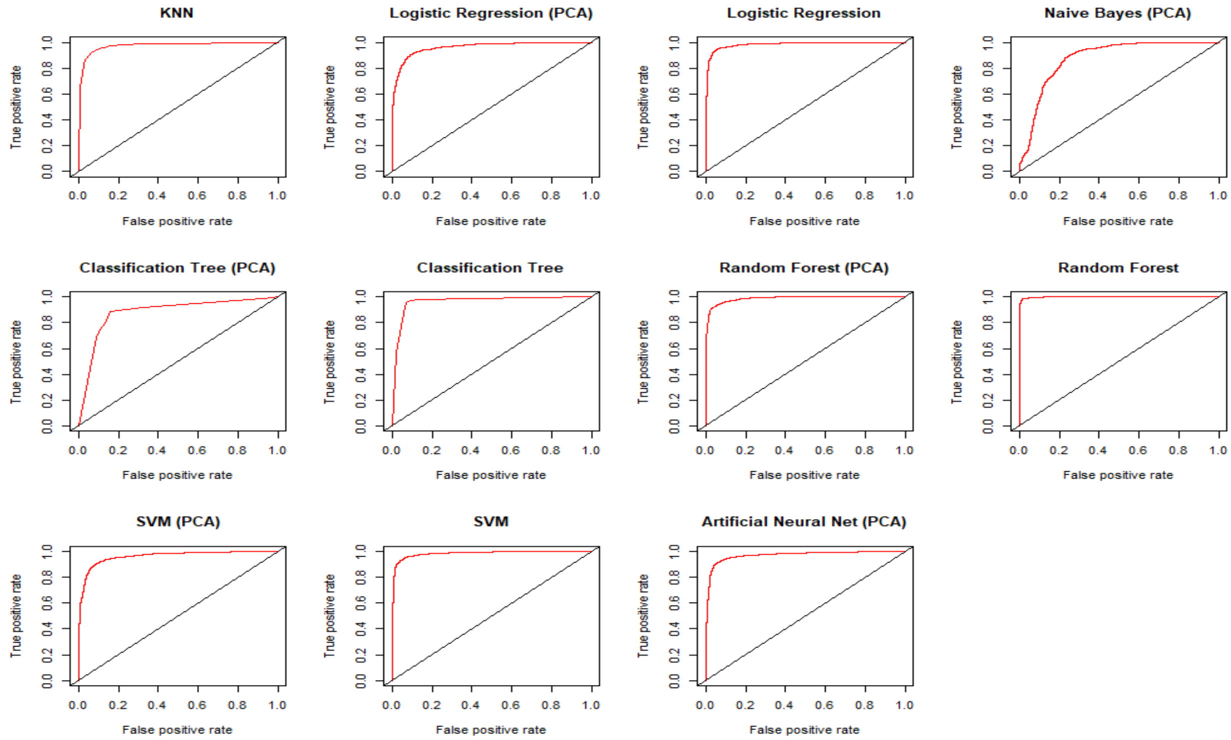


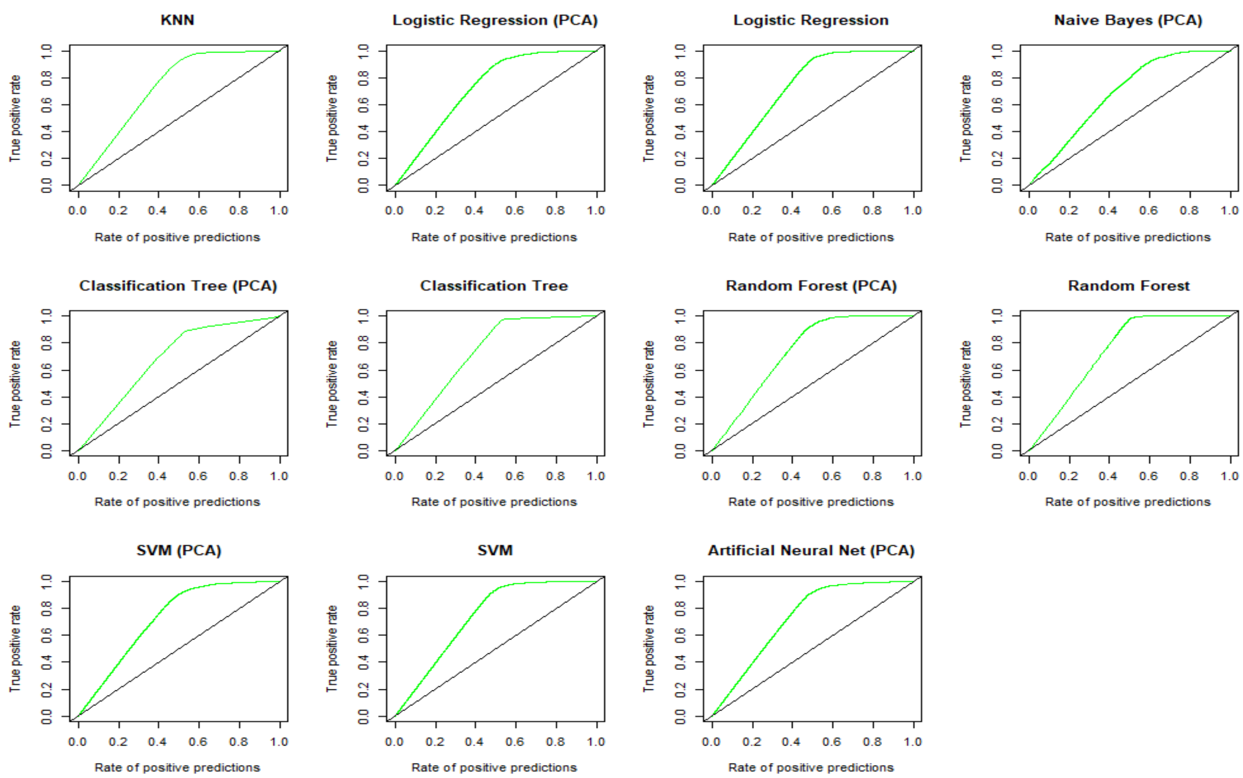
Figure 11: Artificial Neural Net with 1 hidden layer

We compared the models by plotting their ROC curves and Lift Chart

ROC Curves



Lift Charts



We then created a performance table to compare them all –

Model	AUC	Accuracy(%)
KNN	0.975035823	93.05
Logistic Regression (PCA)	0.962171997	90.375
Logistic Regression	0.983244034	94.5
Naive Bayes (PCA)	0.876928724	71.125
Classification Tree (PCA)	0.883643237	86.3
Classification Tree	0.960858574	94.5
Random Forest (PCA)	0.987085167	93.875
Random Forest	0.998671199	98.35
SVM (PCA)	0.96090908	90.775
SVM	0.983473814	94.3
Artificial Neural Net (PCA)	0.970019285	92.65

After evaluating and comparing all the models we concluded that Random Forest classifiers performed the best on this dataset achieving a validation accuracy of 98.35% and an Area Under Curve (AUC) value of 0.9986. Note that these metrics are for a random forest classifier trained on the normalized full dataset (not the PCA one).

VI. Discussion and Recommendation

Although we found that random forest classifier performs the best on this task, a strong case can be made that a Support Vector Machine or a Neural Network can perform better on the data on further hyperparameter tuning and better feature selection. However, feature selection requires domain knowledge on the subject. Moreover, a case against Neural Nets can also be given that they are black boxes while decision trees and forests are easy to interpret.

VII. Summary

In this study we test many different machine learning models to perform this binary classification task. We train the model on a normalized dataset. We conclude that a Random Forest Classifier performs the best in this task.

Appendix: R Code for use case study

```
# Case Study
library(tidyverse)
library(psych)
library(caret)
library(FNN)
library(ISLR)
library(tree)
library(randomForest)
library(neuralnet)
library(ROCR)
library(e1071)
library(gains)
library(ggplot2)
library(reshape2)
library(rpart)
library(rpart.plot)
library(corrplot)

# Load phishing_websites.csv
df <- data.frame(read.csv("./data/phishing_websites.csv"))
# Remove "HttpsInHostname" column because it contains a few NAs
df$PHISHING_WEBSITE <- as.factor(ifelse(df$CLASS_LABEL == 1, "Yes", "No"))
df <- df[, !colnames(df) %in% c("HttpsInHostname", "CLASS_LABEL")]

describe(df)
##################################################################
## Data Visualization

# Let's look at NumDots Histogram
ggplot(df, aes(NumDots)) +
  geom_histogram(binwidth = 1, color = "black", fill = "steelblue") +
  ggtitle("NumDots Histogram")

# Let's look at the UrlLength Histogram
ggplot(df, aes(UrlLength)) +
  geom_histogram(binwidth = 25, color = "black", fill = "steelblue") +
  ggtitle("UrlLength Histogram")

# Let's look at whether having an IP address in the Url gives us any information as
# to whether the website is a phishing website or not
ggplot(df, aes(as.factor(IpAddress), fill = PHISHING_WEBSITE)) +
  geom_histogram(stat = "count") +
  ggtitle("IpAddress Barplot") +
  labs(x = "IP Address")
```

```
# corrplot

cor <- round(cor(df[, 1:47]), 2)
corrplot(cor, type = "upper")
## Looks like all sites having an IP Address in the Url are phishing websites.

#################################################################
## Data Pre-processing

# Define the normalize function
normalize <- function(x) {
  return((x - min(x))) / (max(x) - min(x))
}

# Normalize the data frame
df.norm <- as.data.frame(cbind(
  as.data.frame(lapply(df[1:47], normalize)),
  df$PHISHING_WEBSITE
)) %>%
  rename(PHISHING_WEBSITE = "df$PHISHING_WEBSITE")


#################################################################
## Data Reduction and Transformation


# Performing PCA on the data
# Perform Scree Plot and Parallel Analysis
fa.parallel(df.norm[, 1:47], fa = "pc", n.iter = 100, show.legend = FALSE)

# Perform PCA with 13 components
pc <- principal(df.norm[, 1:47], nfactors = 13, rotate = "none", scores = TRUE)
pc <- cbind(as.data.frame(pc$scores), df.norm$PHISHING_WEBSITE) %>%
  rename(PHISHING_WEBSITE = "df.norm$PHISHING_WEBSITE")

#################################################################
## Data Mining Techniques
# Splitting data into training and validation sets
# Generate the training data indices

set.seed(20)
indices <- sample(seq_len(nrow(pc)), size = floor(0.6 * nrow(pc)))
# Get training and validation data
train_data <- pc[indices, ]
validation_data <- pc[-indices, ]
```

```
levels(train_data$PHISHING_WEBSITE) <-
  make.names(levels(factor(train_data$PHISHING_WEBSITE)))
levels(validation_data$PHISHING_WEBSITE) <-
  make.names(levels(factor(validation_data$PHISHING_WEBSITE)))

# corrplot of pca data
cor <- cor(pc[, 1:13])
corrplot(cor, type = "upper")

# Also keep a set of train and validation sets without PCA
df.norm.train <- as.data.frame(lapply(df.norm[indices, ], as.numeric))
df.norm.validation <- as.data.frame(lapply(df.norm[-indices, ], as.numeric))

df.norm.train <- df.norm[indices, ]
df.norm.validation <- df.norm[-indices, ]
df.norm.train$PHISHING_WEBSITE <- as.factor(df.norm.train$PHISHING_WEBSITE)
df.norm.validation$PHISHING_WEBSITE <-
as.factor(df.norm.validation$PHISHING_WEBSITE)

levels(df.norm.train$PHISHING_WEBSITE) <-
  make.names(levels(factor(df.norm.train$PHISHING_WEBSITE)))
levels(df.norm.validation$PHISHING_WEBSITE) <-
  make.names(levels(factor(df.norm.validation$PHISHING_WEBSITE)))



# Creating a performance list for each algorithm
performance_list <- data.frame(
  "Model" = character(),
  "AUC" = numeric(),
  "Accuracy" = numeric()
)

model_names <- list()
lift_charts <- list()
roc_curves <- list()

# Helper Function to plot ROC Curve and Calculate Accuracy

evaluate_performance <- function(pred, labels, model_name) {
  model_names[[length(model_names) + 1]] <<- model_name

  # Accuracy
  pred.class <- ifelse(slot(pred, "predictions")[[1]] > 0.5, "Yes", "No")
  levels(pred.class) <- make.names(levels(factor(pred.class)))
```

```
  acc <- confusionMatrix(table(pred.class, labels))$overall[[1]] * 100

  # ROC Plot
  roc <- performance(pred, "tpr", "fpr")
  plot(roc, col = "red", lwd = 2, main = paste0(model_name, " ROC Curve"))
  abline(a = 0, b = 1)

  roc_curves[[length(roc_curves) + 1]] <<- roc


  auc <- performance(pred, measure = "auc")

  temp <- data.frame(
    "Model" = model_name,
    "AUC" = auc@y.values[[1]],
    "Accuracy" = acc
  )
  performance_list <<- rbind(performance_list, temp)
  print("Updated Performance List")

  lift <- performance(pred, "tpr", "rpp")
  plot(lift, main = paste0(model_name, " Lift Curve"), col = "green")
  abline(a = 0, b = 1)

  lift_charts[[length(lift_charts) + 1]] <<- lift

  rm(list = c("auc", "acc", "roc", "pred.class", "temp", "lift"))
}


#######################
## Implementing KNN

# Setting up train controls
tc <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)


set.seed(20)
knn.model <- train(PHISHING_WEBSITE ~ .,
  data = train_data, method = "knn",
```

```
  trControl = tc,
  metric = "ROC",
  tuneLength = 10
)

# Look at the KNN Model
knn.model
plot(knn.model)

# get predictions for validation data
knn.pred <- predict(knn.model, validation_data, type = "prob")
pred.val <- prediction(knn.pred[, 2], validation_data$PHISHING_WEBSITE)


evaluate_performance(pred.val, validation_data$PHISHING_WEBSITE, "KNN")

rm(list = c("knn.model", "tc", "knn.pred", "pred.val"))


#######################
## Implementing Logistic Regression
# On PCA Dataset
set.seed(20)
glm.fit.pc <- glm(PHISHING_WEBSITE ~ ., data = train_data, family = binomial)

glm.probs.pc <- predict(glm.fit.pc, newdata = validation_data, type = "response")
pred.val <- prediction(glm.probs.pc, validation_data$PHISHING_WEBSITE)

evaluate_performance(
  pred.val, validation_data$PHISHING_WEBSITE,
  "Logistic Regression (PCA)"
)


# On Original Dataset
set.seed(20)
glm.fit <- glm(PHISHING_WEBSITE ~ ., data = df.norm.train, family = binomial)

glm.probs <- predict(glm.fit, newdata = df.norm.validation, type = "response")
pred.val <- prediction(glm.probs, df.norm.validation$PHISHING_WEBSITE)

evaluate_performance(
  pred.val, validation_data$PHISHING_WEBSITE,
  "Logistic Regression"
)
```

```
rm(list = c(
  "glm.fit", "glm.probs",
  "glm.fit.pc", "glm.probs.pc",
  "pred.val"
))

#######################
## Implementing Naive Bayes
set.seed(20)
nb <- naiveBayes(PHISHING_WEBSITE ~ ., data = train_data)

nb.pred <- predict(nb, newdata = validation_data, type = "raw")
pred.val <- prediction(nb.pred[, 2], validation_data$PHISHING_WEBSITE)

evaluate_performance(pred.val, validation_data$PHISHING_WEBSITE, "Naive Bayes (PCA)")

rm(list = c("nb", "nb.pred", "pred.val"))

#######################
## Implementing Decision Tree
# Classification tree on PCA Dataset

set.seed(20)
tree.pca <- rpart(PHISHING_WEBSITE ~ ., data = train_data, method = "class")
rpart.plot(tree.pca, main = "Classification Tree (PCA)")
tree.pca.pred <- predict(tree.pca, validation_data)
pred.val <- prediction(tree.pca.pred[, 2], validation_data$PHISHING_WEBSITE)


evaluate_performance(
  pred.val,
  validation_data$PHISHING_WEBSITE,
  "Classification Tree (PCA)"
)

# Classification tree on Original Dataset
set.seed(20)
tree <- rpart(PHISHING_WEBSITE ~ ., data = df.norm.train, method = "class")
rpart.plot(tree, main = "Classification Tree")

print(tree$variable.importance)

tree.pred <- predict(tree, df.norm.validation)
pred.val <- prediction(tree.pred[, 2], df.norm.validation$PHISHING_WEBSITE)

evaluate_performance(
```

```
  pred.val,
  validation_data$PHISHING_WEBSITE,
  "Classification Tree"
)

rm(list = c(
  "tree.pca", "tree.pca.pred", "tree",
  "tree.pred", "pred.val"
))

#######################
## Implementing Random Forests

# On PCA dataset
set.seed(20)
rf.pca <- randomForest(PHISHING_WEBSITE ~ ., data = train_data)
rf.pca.pred <- predict(rf.pca, validation_data, type = "prob")
pred.val <- prediction(rf.pca.pred[, 2], validation_data$PHISHING_WEBSITE)

evaluate_performance(
  pred.val,
  validation_data$PHISHING_WEBSITE,
  "Random Forest (PCA)"
)
# On original dataset
set.seed(20)
rf <- randomForest(PHISHING_WEBSITE ~ ., data = df.norm.train)
rf.pred <- predict(rf, df.norm.validation, type = "prob")
pred.val <- prediction(rf.pred[, 2], df.norm.validation$PHISHING_WEBSITE)

evaluate_performance(
  pred.val,
  df.norm.validation$PHISHING_WEBSITE,
  "Random Forest"
)


rm(list = c(
  "rf.pca", "rf.pca.pred", "rf",
  "rf.pred", "pred.val"
))

#######################
## Implementing Support Vector Machine
tc <- trainControl(
  method = "repeatedcv",
```

```
  number = 5,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

set.seed(20)
svm.pca <- train(PHISHING_WEBSITE ~ ., train_data,
  method = "svmLinear",
  trControl = tc, tuneLength = 10
)

svm.pca.pred <- predict(svm.pca, validation_data, type = "prob")

pred.val <- prediction(svm.pca.pred[, 2], validation_data$PHISHING_WEBSITE)

evaluate_performance(
  pred.val,
  validation_data$PHISHING_WEBSITE,
  "SVM (PCA)"
)

# On Original Dataset
tc <- trainControl(
  method = "repeatedcv",
  number = 5,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

set.seed(20)
svm <- train(PHISHING_WEBSITE ~ .,
  df.norm.train,
  method = "svmLinear",
  preProcess = NULL,
  trControl = tc,
  metric = "ROC",
  tuneLength = 10
)

svm.pred <- predict(svm, df.norm.validation, type = "prob")

pred.val <- prediction(svm.pred[, 2], df.norm.validation$PHISHING_WEBSITE)

evaluate_performance(
```

```
  pred.val,
  validation_data$PHISHING_WEBSITE,
  "SVM"
)

rm(list = c("tc", "svm.pca", "svm", "svm.pred", "svm.pca.pred", "pred.val"))
#######################
## Implementing Artificial Neural Networks
# On PCA Dataset
set.seed(20)
nn.pca <- neuralnet(PHISHING_WEBSITE ~ .,
  data = train_data,
  hidden = 3,
  act.fct = "logistic",
  linear.output = FALSE
)

plot(nn.pca, main = "Artificial Neural Net (PCA)")

nn.pca.pred <- neuralnet::compute(nn.pca, validation_data[, 1:13])$net.result
pred.val <- prediction(nn.pca.pred[, 2], validation_data$PHISHING_WEBSITE)
evaluate_performance(
  pred.val, validation_data$PHISHING_WEBSITE,
  "Artificial Neural Net (PCA)"
)

rm(list = c("nn.pca", "nn.pca.pred", "pred.val"))
##############################################################

write.csv(performance_list, "performance_list.csv")

# Plot all ROC Curves
par(mfrow = c(3, 4))
for(i in 1:length(roc_curves)) {
  plot(roc_curves[[i]], main = model_names[[i]], col = "red")
  abline(a = 0, b = 1)
}


# Plot all lift charts
par(mfrow = c(3, 4))
for(i in 1:length(lift_charts)) {
  plot(lift_charts[[i]], main = model_names[[i]], col = "green")
  abline(a = 0, b = 1)
}
```