**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**

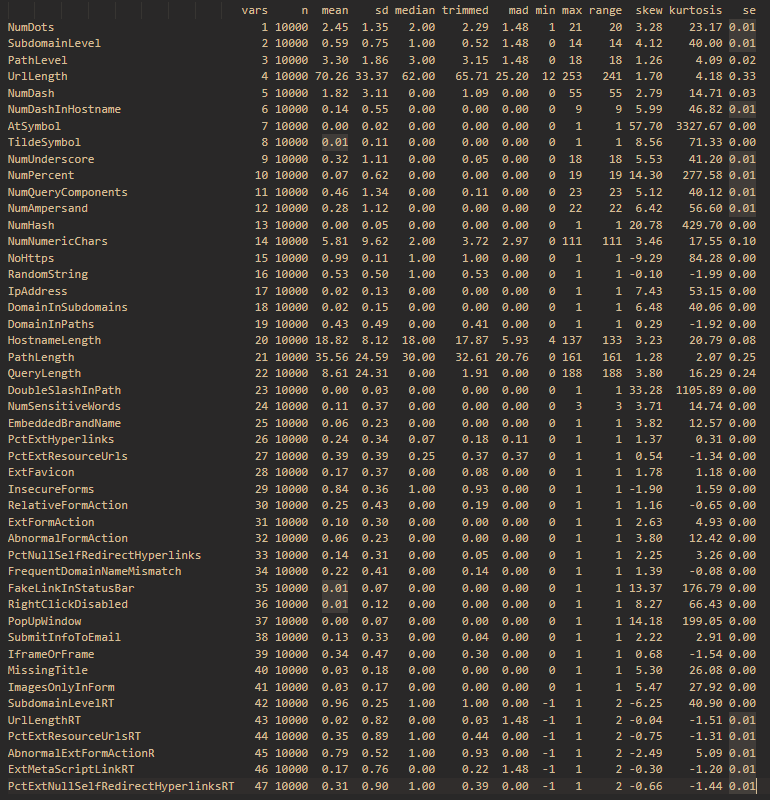
We analyzed the distributions of some attributes to get an idea about the distribution of the data.

Figure : Data Summary

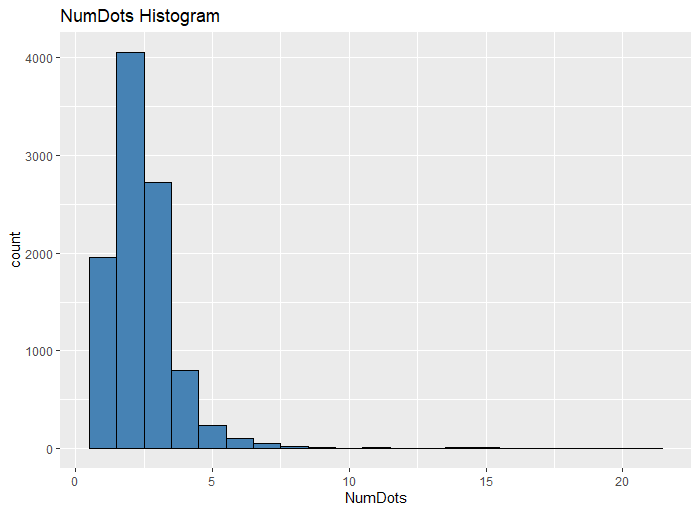
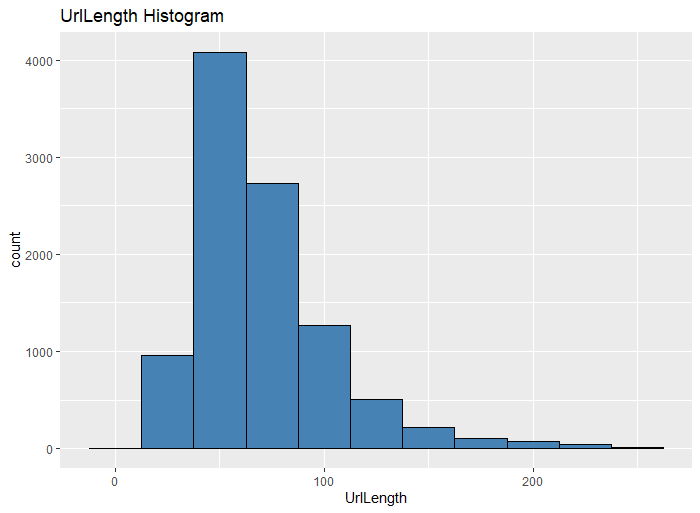


Figure : UrlLength Histogram

Figure : NumDots Histogram

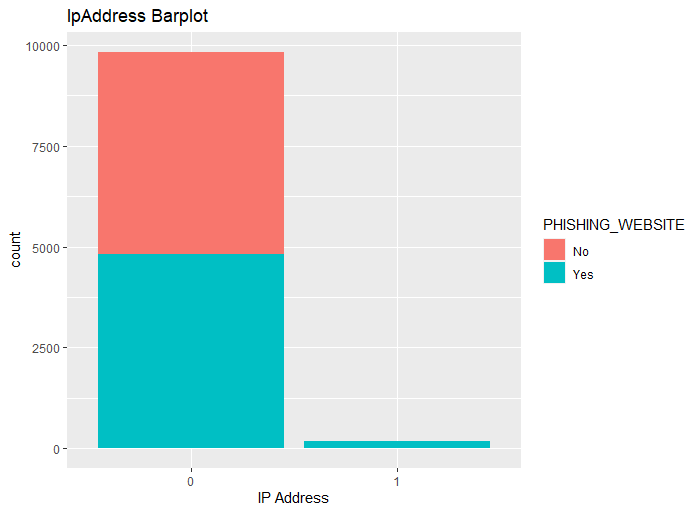
An interesting observation we found was that all websites which contained an IP Address in their URL where phishing websites.

Figure : IPAddress Barplot

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

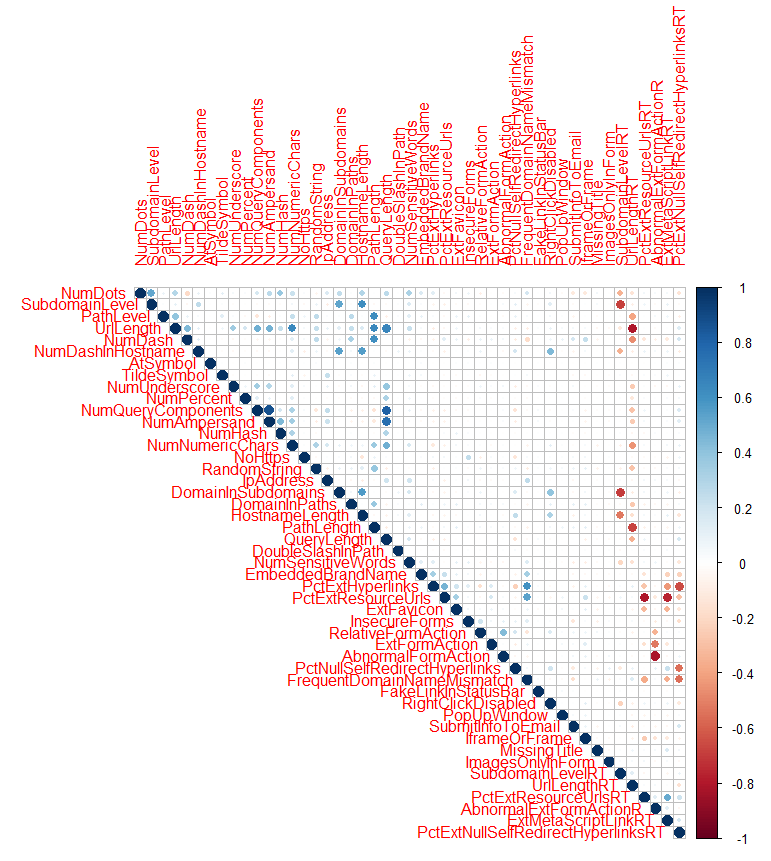


Figure : Data Correlation Plot

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**

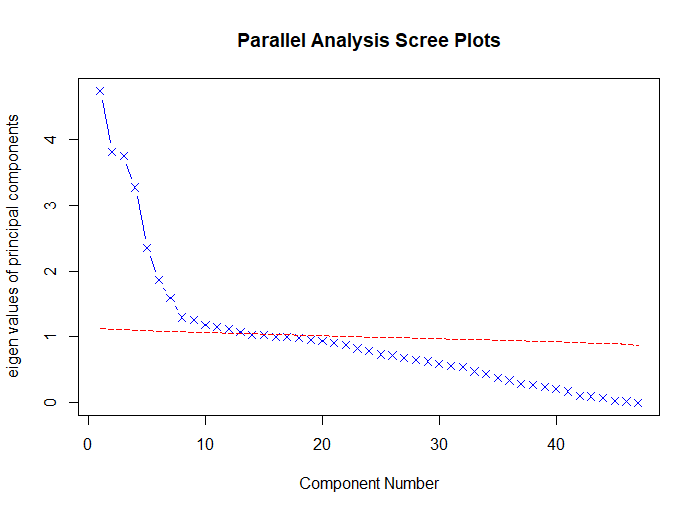
Our dataset has 48 attributes all in different scales. We first normalize our entire dataset of 10,000 rows using Min-Max Normalization. We then use Principal Component Analysis to reduce the dimensions of our data from 48 attributes to just 13 principal components which capture 61% of the data’s variance.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

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

After performing PCA, we plot the correlation matrix.

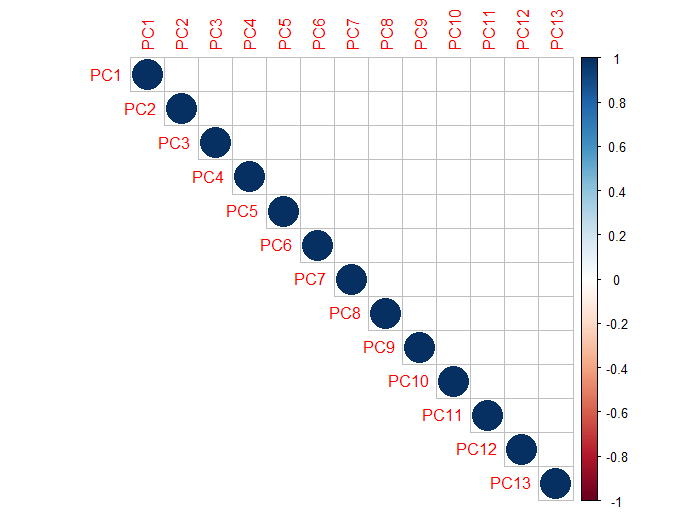


Figure : Correlation Plot for PCA data

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**

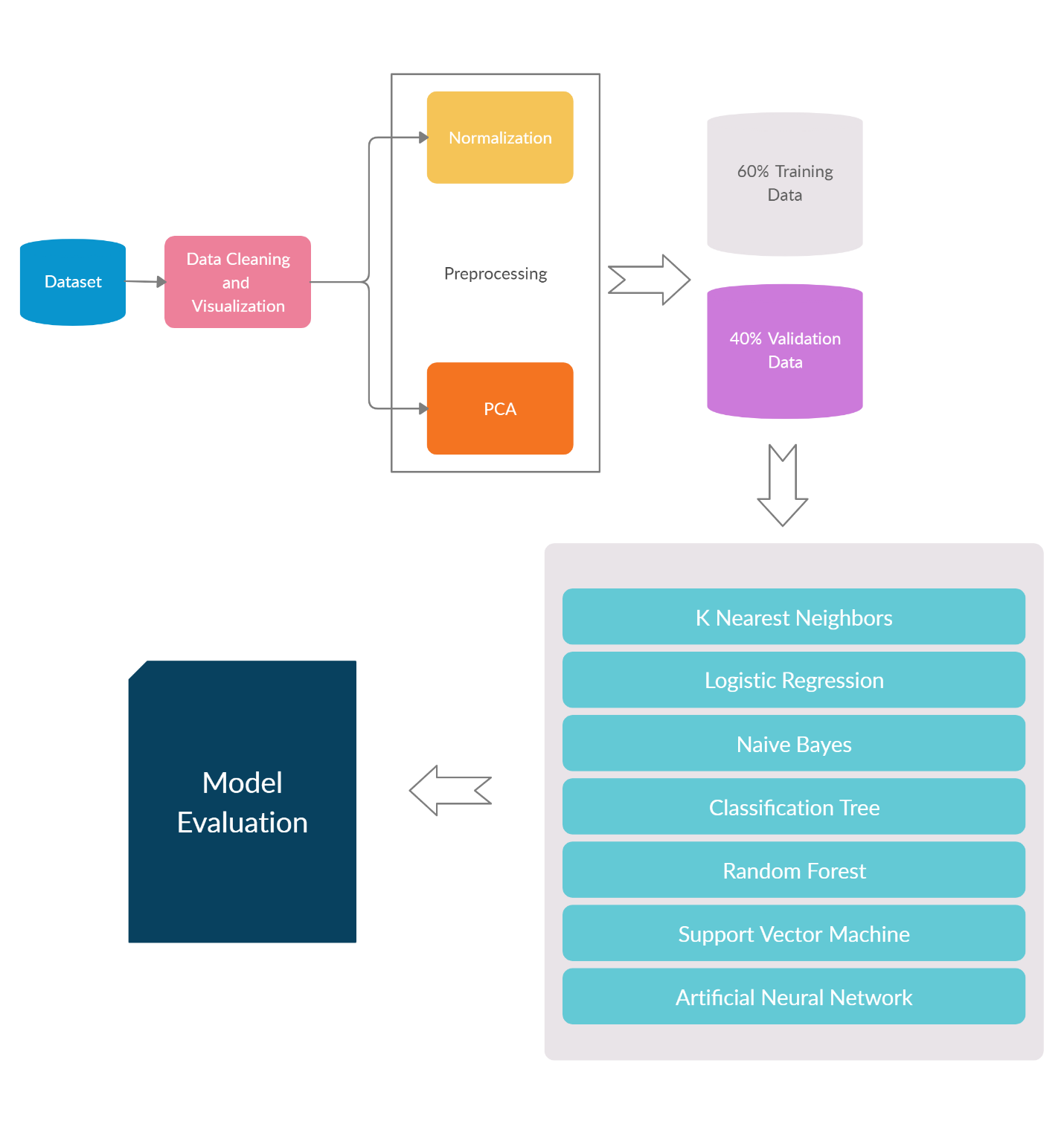
****

Figure : 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:

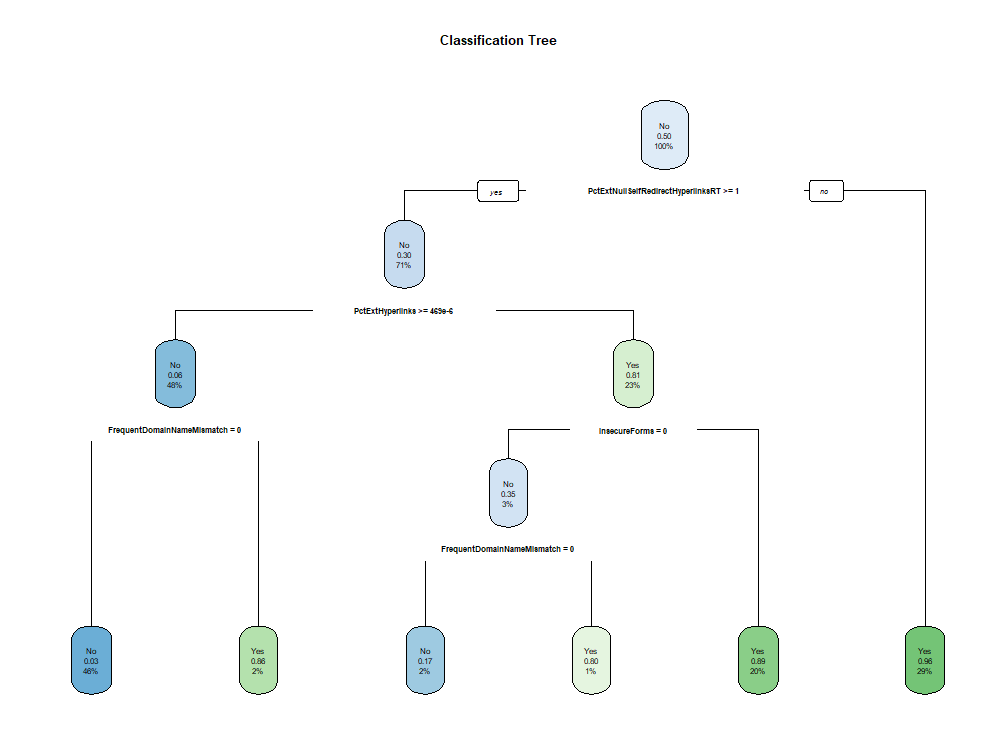
**Classification Tree**

Figure : Classification Tree

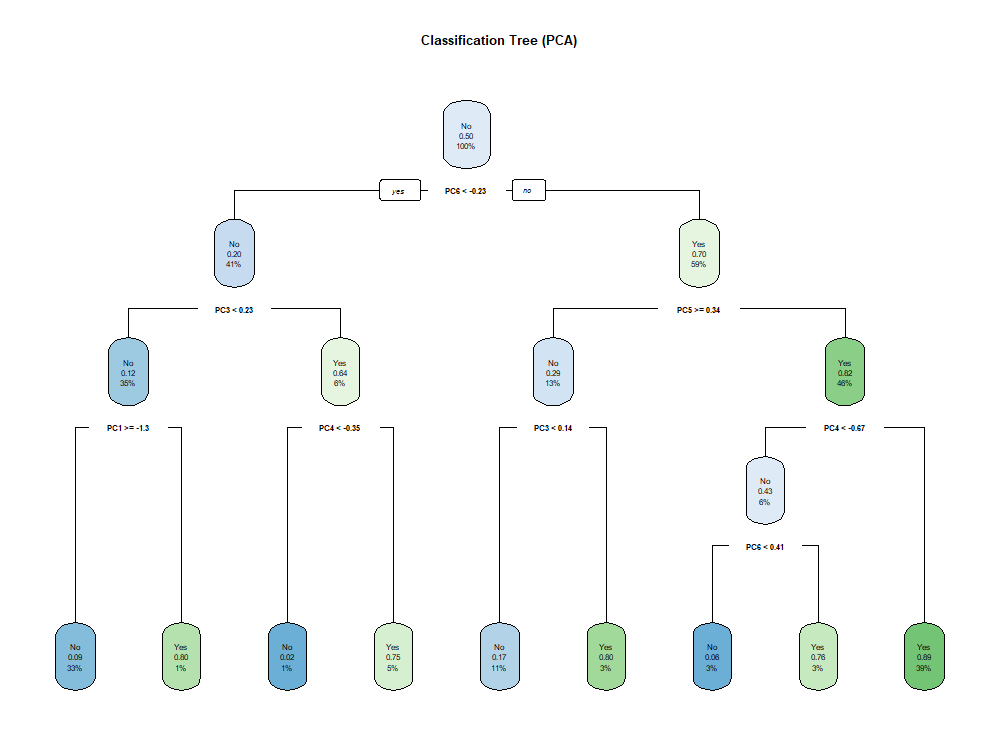
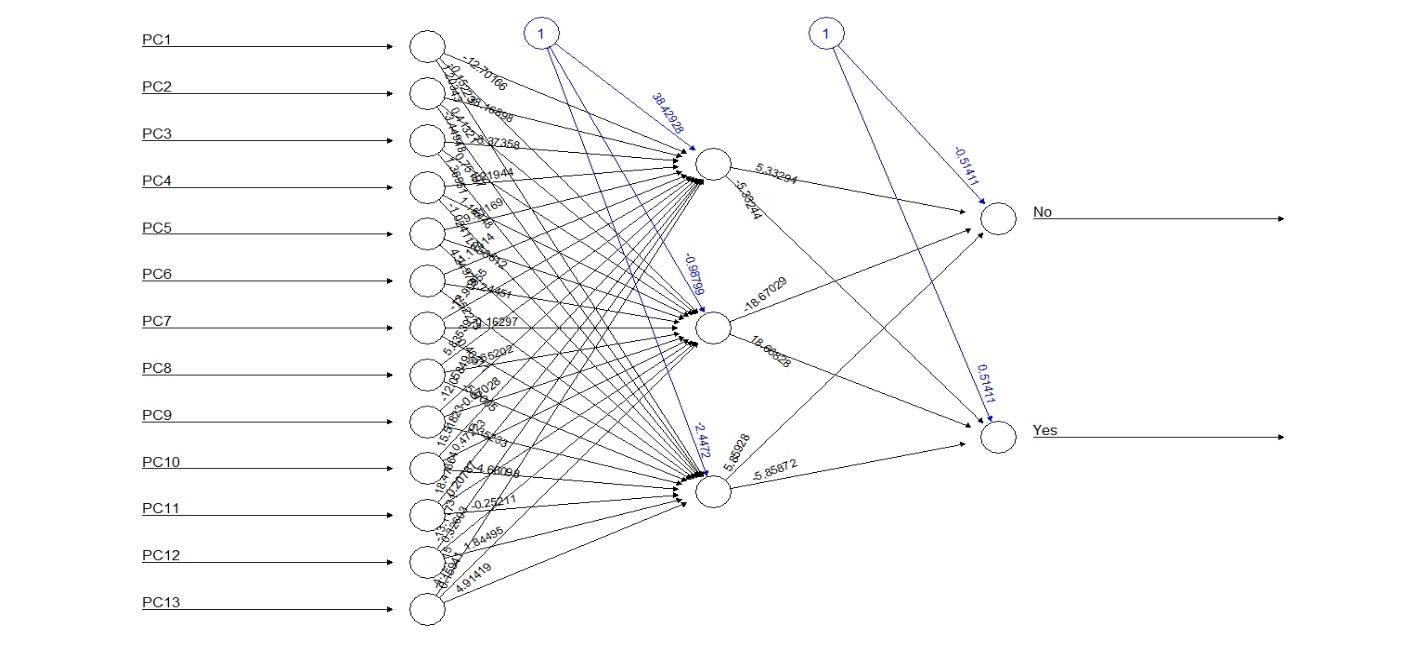
**Classification Tree (PCA)**

Figure : Classification Tree (PCA)

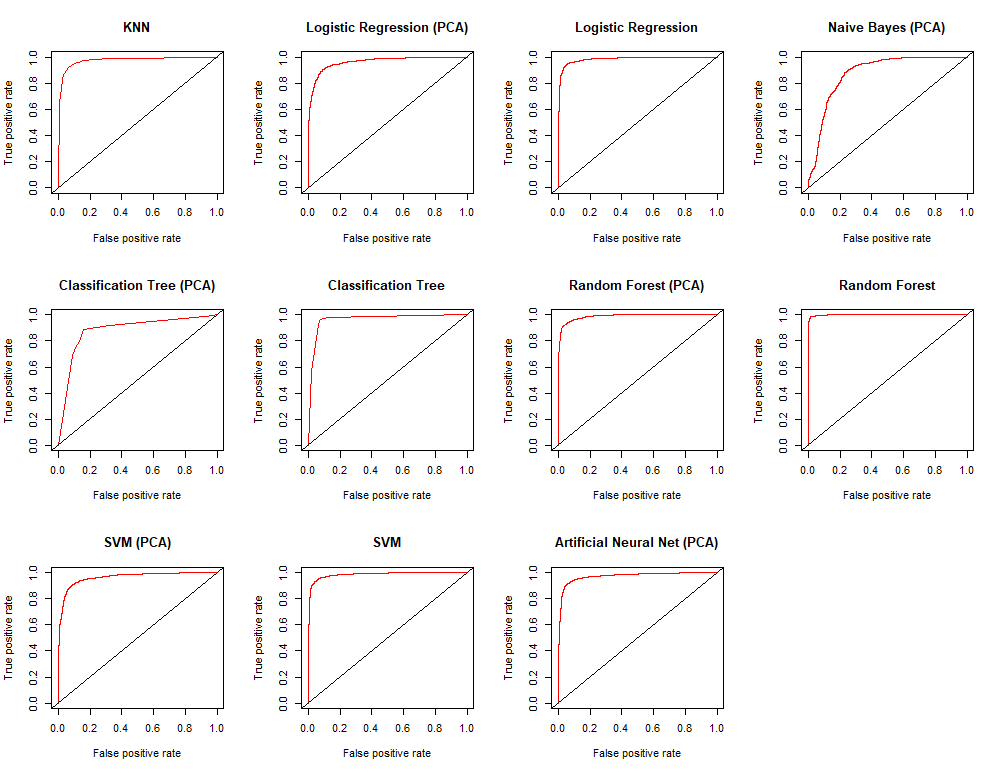
**Neural Network (PCA)**

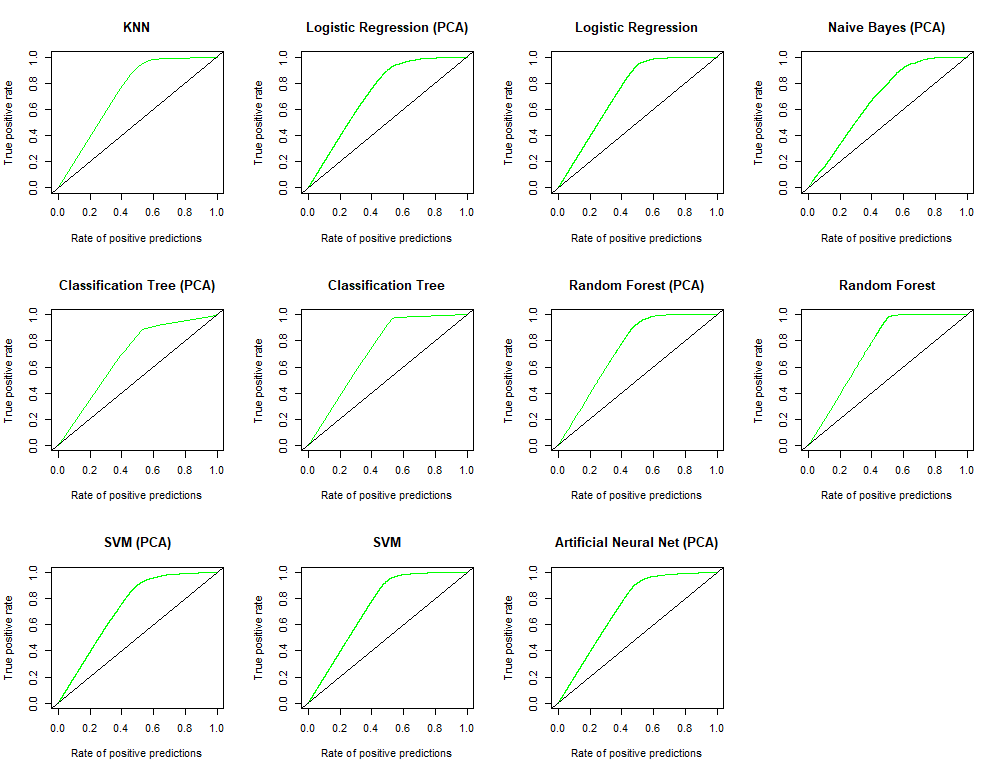
Figure : Artificial Neural Net with 1 hidden layer

****

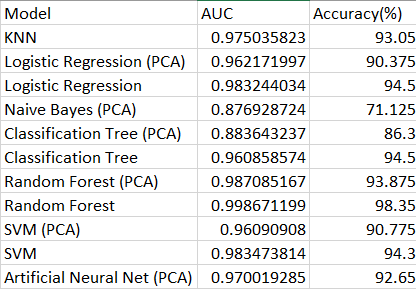
We compared the models by plotting their ROC curves and Lift Chart

**ROC Curves**

****

**Lift Charts**

We then created a performance table to compare them all –



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)

}