Detection of Phishing Websites using Machine Learning Algorithms

Naman Agrawal

December 26, 2020

Contents

Executive Summary

The purpose of this project is to use different machine learning approaches to come up with suitable algorithms for the detection of phishing websites. The project will first provide an introduction to phishing. This is followed by a description of the data set used to build the algorithms. Some exploratory data analysis has also been done to provide a qualitative and quantitative explanation of the various characteristics of the data set. The project then illustrates the various machine learning algorithms that will be used for prediction. A comparative analysis of these algorithms has also been done. The project concludes with a brief overview of the results and highlights some future considerations.

Introduction

With rapid advancements in the field of digital technology and software-systems, various issues related to cyber-safety have been gaining importance. One of the most common issues in this regard is Personal Identity Theft. Identity theft simply means one person taking the identity of another person.

One of the most common forms of identity theft is phishing. Phishing involves tempting the target to reveal sensitive information such as passwords, codes, account details, etc, which may be used for illegal activities. Through emails, messages, calls, etc, the users may be directed to fake websites where they unknowingly reveal personal information. These websites are called phishing websites. They are designed to imitate legitimate websites but use subterfuge to gather details. However, there are certain features which can be used to differentiate phishing and legitimate websites. Though these features are no definite rule, they can be exploited through appropriate manipulation and implementation of suitable algorithms to predict website legitimacy to a reasonable accuracy.

The project will be based on the following methodology:

1. The data set is downloaded and pre-processed to choose features which show high correlation with the outcome.
2. The selected features are analyzed individually to garner important information regarding their distribution, prevalence, etc.
3. Data set is split into train and test sets. The train set is used to build the algorithms and the test set is used exclusively for testing.
4. Algorithms are constructed with the distinguishing features as the predictors and the result indicating whether the given website is phishing or legitimate.
5. The most suitable algorithm (maximum accuracy for train and test set) is chosen.

The project will use R for all of the above-mentioned tasks. The following libraries are required:

```r
library(tidyverse)
library(RWeka)
library(Hmisc)
library(caret)
library(tidyr)
library(purrr)
library(party)
library(partykit)
library(gam)
library(e1071)
library(Rborist)
library(randomForest)
library(LiblineaR)
library(lubridate)
library(knitr)
library(ggcorrplot)
library(gbm)
library(xfun)
library(reactable)
library(kableExtra)
library(tinytex)
library(MLmetrics)
library(lemon)
```

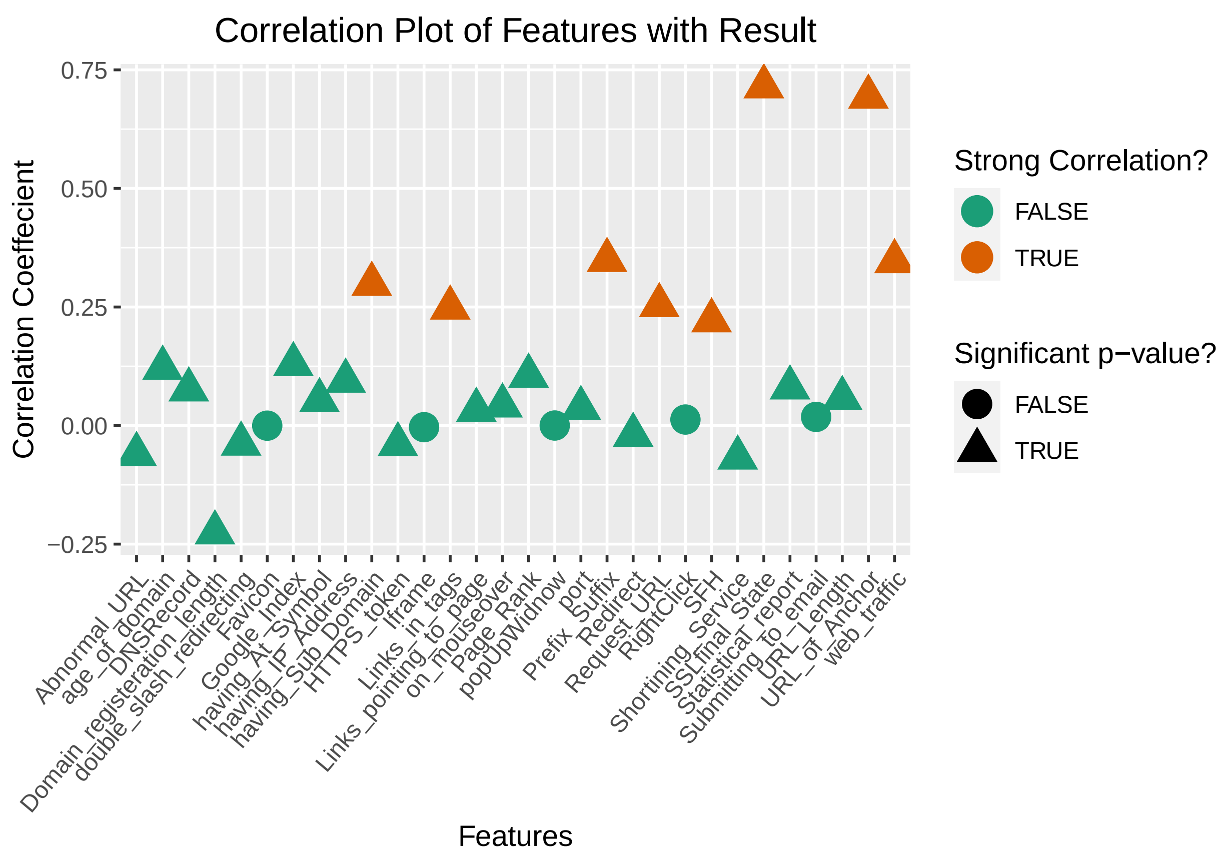
Data Set Description

The data set used for this project has been downloaded from the UCI Machine Learning Repository. It has been provided by Rami Mustafa A Mohammad, Lee McCluskey, and Fadi Thabtah. The data set can be downloaded and saved in '.rda' format by the following code:

```r
#File URL
url <-
"https://archive.ics.uci.edu/ml/machine-learning-databases/00327/Training%20Dataset.arff"

#Creating a temporary file name
temp <- tempfile()

#Downloading the file.
download.file(url,temp)

#Reading the file into an R object
phishing <- read.arff(temp)

#remove the temporary file
unlink(temp)
```

There are 31 attributes in this data set. The 31st attribute is 'Result', which is the outcome (indicating whether the website is actually phishing or legitimate). There are 11,055 observations in the data set. All values are categorical in nature, and comprise two/three levels indicating that the website may be phishing (-1), suspicious (0), or legitimate (1).

Before analysis, the data set is pre-processed to only include those attributes which show a strong correlation with the result (>=0.2) and a significant associated p-value (<=0.05). This allows us to not only choose the best predictors, but also avoid long computational time due to several predictors. The figure below shows the correlation coefficient for the 30 attributes with respect to 'Result'.



In this plot, the 8 orange triangles indicate the features which show a high degree of correlation which significant p-values. These are the variables that will be selected from the phishing data set to be used for analysis and prediction. The final data set is called 'phish'. It has 9 attributes:

Table 1: Attributes Information

Attribute	Value_Set	Explanation
Prefix_Suffix	{-1,1}	Hyphens are rarely used in legitimate URLs. Phishers tend to add prefixes or suffixes separated by hyphen from the domain name to trick users into assuming that they are dealing with a legitimate webpage. If the domain name part includes hyphen, the value is '-1' (indicating phishing). Otherwise, the value is '1' (indicating legitimate).
having_Sub_Domain	{-1, 0, 1}	The number of dots in the URL is counted after omiting the 'www.' and the country-code top-level domain (ccTLD). If the number of dots so counted is 1, the attribute is given a value '1' (indicating legitimate). If the number of dots is 2, the value is '0' (indicating suspicious). If the number of dots is more than 2, the value is '-1' (indicating phishing).

Table 1: Attributes Information *(continued)*

Attribute	Value_Set	Explanation
SSLfinal_State	{-1, 0, 1}	SSL stands for Secure Sockets Layer. The certificate assigned with HTTPS is thououghly checked. The minimum age of a reputable certificate is two years. If the website uses https, the issuer is trusted and the age of certificate is more than 1 year, the value of the attribute is '1' (indicating legitimate). If the website uses https and the issuer is not trusted, the value is '0' (indicating suspicious). Otherwise, the value is '-1' (indicating phishing).
Request_URL	{-1, 1}	Request URL examines whether the external objects contained within a webpage such as images, videos and sounds are loaded from another domain. In legitimate webpages, the webpage address and most of objects embedded within the webpage are sharing the same domain. If request URLs loaded from other domains are less than 22%, the value of the attribute is '1' (indicating legitimate). If they are more than or equal to 22%, but less than 61%, the value is '0' (indicating suspicious). Otherwise, the value is '-1' (indicating phishing). In our data set, none of the websites are given the value 0. Hence, the possible values are only 1 and -1.
URL_of_Anchor	{-1, 0, 1}	An anchor is an element defined by the <a> tag. It is examined if the <a> tags and the website have different domain names and if the anchors link to any webpage. If such anchors are less than 31%, the attribute value is '1' (indicating legitimate). If they are more than or equal to 31%, but less than 67%, the value is '0' (indicating suspicious). Otherwise, the value is '-1' (indicating phishing).
Links_in_tags	{-1, 0, 1}	It is common for legitimate websites to use <Meta> tags to offer metadata about the HTML document, <Script> tags to create a client side script, and <Link> tags to retrieve other web resources. If the links in such tags are less than 17%, the attribute value is '1' (indicating legitimate). If they are more than or equal to 17%, but less than 81%, the value is '0' (indicating suspicious). Otherwise, the value is '-1' (indicating phishing).
SFH	{-1, 0, 1}	SFH stands for Server Form Handler.If the SFHs contain an empty string or 'about$:$blank' the attribute is assigned a value of '-1' (indicating phishing). If the domain name in SFHs are differnet from the domain name of the webpage, the value is '0' (indicating suspicious). Otherwise, the value is '1' (indicating legitimate).
web_traffic	{-1, 0, 1}	Website traffic models the popularity of the website in terms of website rank (depending on the number of visitors and the number of pages they visit). If the website rank is less than 100,000, the attribute value is '1' (indicating legitimate). If the website rank is greater than 100,000, the value is '0' (indicating suspicious). Otherwise, (if the domain has no traffic or is not recognized by the database), the value is '-1' (indicating phishing).
Result	{-1, 1}	Result is the outcome. If the website is actually legitimate, the value is '1'. Otherwise, it is given a value of '-1'.

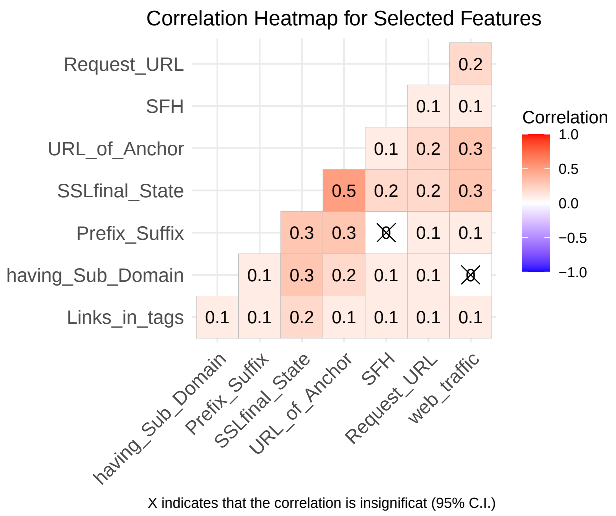
The first 6 rows of the data set ‘phish’ has been displayed for viewing here.

Table 2: First 6 Rows of the Data Set

Prefix_Suffix	having_Sub_Domain	SSLfinal_State	Request_URL	URL_of_Anchor	Links_in_tags	SFH	web_traffic	Result
-1	-1	-1	1	-1	1	-1	-1	-1
-1	0	1	1	0	-1	-1	0	-1
-1	-1	-1	1	0	-1	-1	1	-1
-1	-1	-1	-1	0	0	-1	1	-1
-1	1	1	1	0	0	-1	0	1
-1	1	1	1	0	0	-1	1	1

Exploratory Data Analysis

The correlation among the variables in the final pre-processed data set can be described by the following correlation heat map:

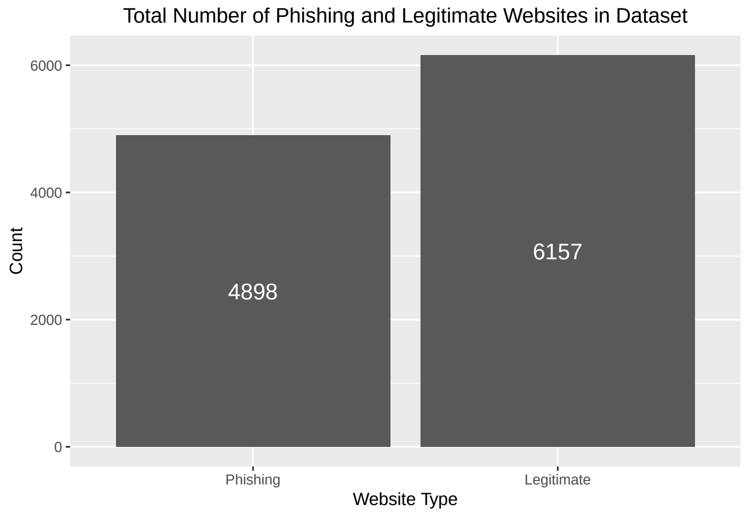


The project shall now examine some noteworthy characteristics of the data set. The prevalence (proportion of phishing websites in the data set) can be calculated as follows:

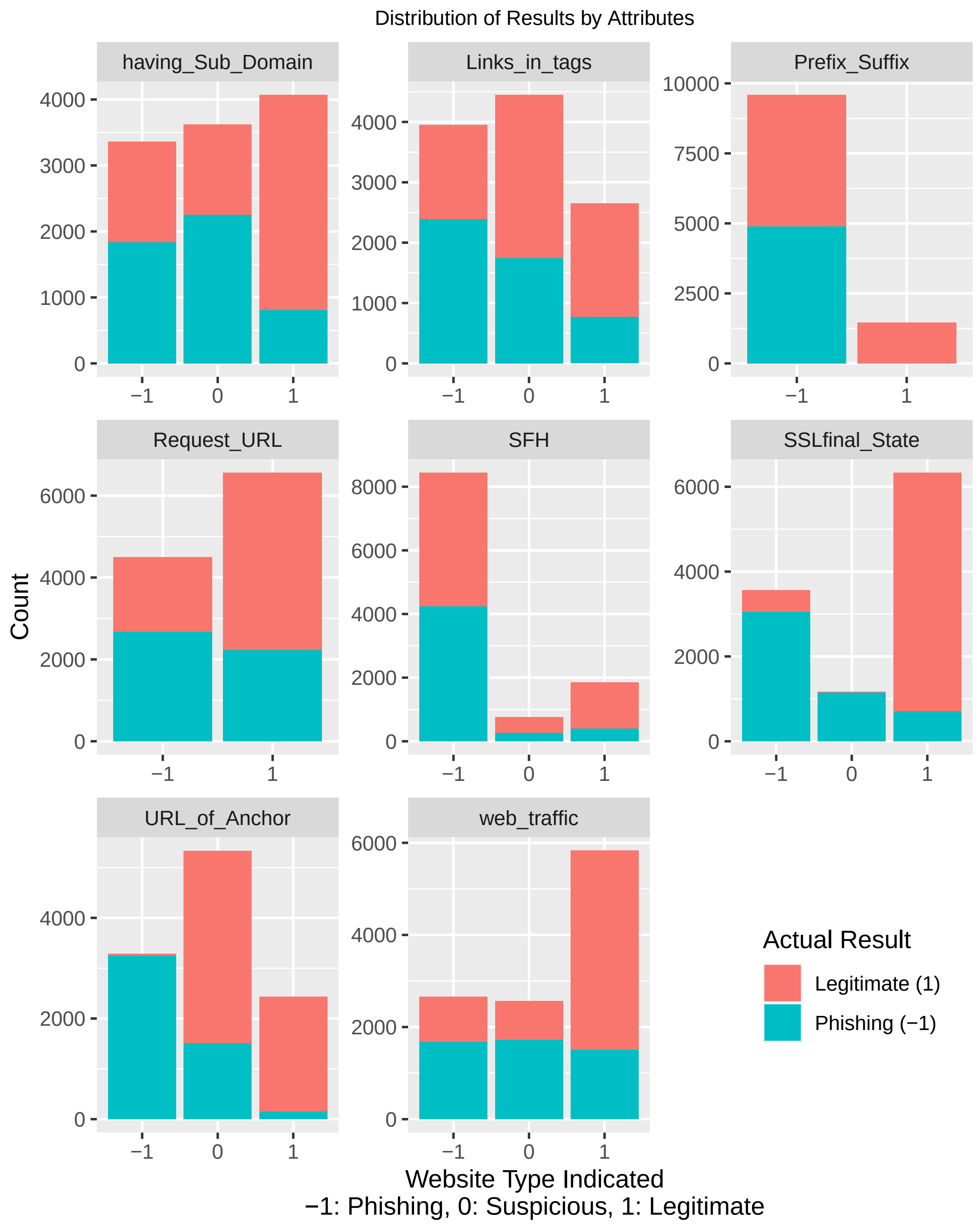
```r
#Finding the proportion of '-1's in the data set:
phish %>% summarise(pi = mean(Result == -1)) %>% pull(pi)
```

```
## [1] 0.4430574
```

As the prevalence is close to 0.5, the number of legitimate websites in only slightly more than the number of phishing websites. The same can be visualized through the following bar plot:



To understand the distribution of the features of the data set and to give an idea of the actual outcomes with respect to the indicated outcome, the following plot has been made:



Machine Learning Algorithms

Different machine learning algorithms are used to predict whether a given website is phishing or not based on the 8 features described above. Before using the algorithms, the data set will have to be split into test and train sets. The train set will be used to build the algorithm and chose optimal parameters, while the test set will be used solely for evaluating the algorithm. The following code enables us to split our data set into two equal parts:

```r
#Setting the seed to 820. This will be done everywhere to have uniformity in prediction.
set.seed(820, sample.kind = "Rounding")

y <- phish$Result
#Creating the text index:
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)

#Creating the test and training sets:
test_set <- phish[test_index, ]
train_set <- phish[-test_index, ]
```

Since, the data set is large (11,055 observations), an allotment of 50% of the observations for testing is feasible. Some of the machine learning algorithms will be described in the subsequent sections.

Regresions

Two kinds of regressions have been used: simple regression and logistic regression.

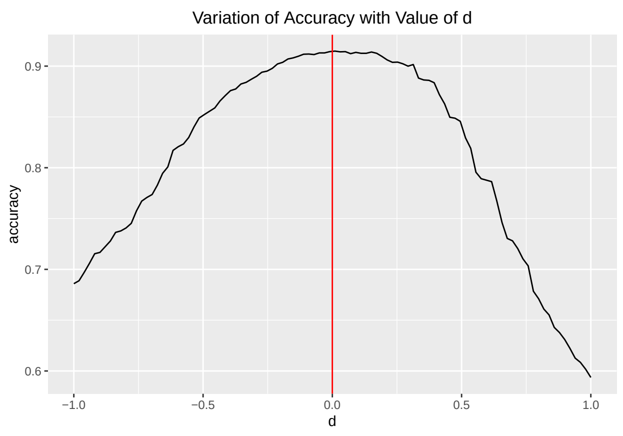
Simple Regression: For simple regression, one can input the train set as a categorical data. However, to allow more flexibility in the regression model, the categorical data is converted into numeric form. Hence, the data set (phish) is converted to numeric form using the following code:

```
phish_numeric <- data.frame(apply(phish, 2, function(x) as.numeric(as.character(x))))
```

Next, a simple regression can be used on the train set as follows:

```
lm_fit <- lm(Result ~ ., data = train_set)
p_hat <- predict(lm_fit, test_set)
```

The predicted outcomes are obtained as continuous numbers, instead of the original discrete values '-1' and '1'. So, it becomes important to use a suitable decision rule to classify the values in p_hat. The following graph has been made to determine the cut-off:



From the graph, it is evident that the accuracy on the train set is maximized when d is close to 0. This value of d is used to calculate the accuracy on the test set.

```
d <- 0
y_hat <- ifelse(p_hat >= d, 1, -1) %>% factor()
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2188  166
##         1   265 2909
##
```

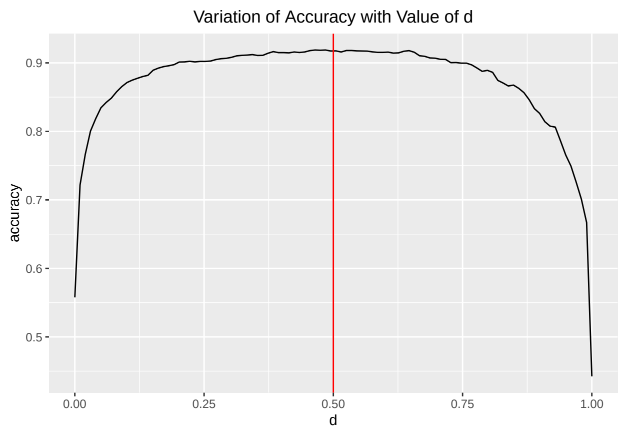
```
##                Accuracy : 0.922
##                  95% CI : (0.9146, 0.929)
##     No Information Rate : 0.5563
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8414
##
##  Mcnemar's Test P-Value : 2.353e-06
##
##             Sensitivity : 0.8920
##             Specificity : 0.9460
##          Pos Pred Value : 0.9295
##          Neg Pred Value : 0.9165
##              Prevalence : 0.4437
##          Detection Rate : 0.3958
##    Detection Prevalence : 0.4258
##       Balanced Accuracy : 0.9190
##
##        'Positive' Class : -1
##
```

An accuracy of 92.2% is obtained using this model.

Logistic Regression: Logistic regression requires the outcome to have values between 0 and 1. So prior to using logistic regression, the data set is manipulated to make all values between 0 and 1. Hence, the '-1' is converted into '0', and '0' into '0.5'. In addition, the data set is again converted to numeric form for more flexibility. The final model can be run as follows:

```
glm_fit <- glm(Result ~ ., data=train_set, family = "binomial")
p_hat <- predict(glm_fit, test_set)
```

However, the predicted values are again obtained as continuous numbers, instead of the original discrete values. So, a decision rule is used, whose cut-off value can be determined from the following graph:



From the graph, it is evident that the accuracy on the train set is maximized when d is close to 0.5. This value of d is used to calculate the accuracy on the test set:

```r
d <- 0.5
y_hat <- ifelse(p_hat >= d, 1, -1) %>% factor()
ytest <- as.factor(phish[test_index, ]$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2251  241
##         1   202 2834
##
##                Accuracy : 0.9199
##                  95% CI : (0.9124, 0.9269)
##     No Information Rate : 0.5563
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8379
##
##  Mcnemar's Test P-Value : 0.07101
##
##             Sensitivity : 0.9177
##             Specificity : 0.9216
##          Pos Pred Value : 0.9033
##          Neg Pred Value : 0.9335
##              Prevalence : 0.4437
##          Detection Rate : 0.4072
##    Detection Prevalence : 0.4508
##       Balanced Accuracy : 0.9196
##
##        'Positive' Class : -1
##
```

An accuracy of 91.99% is obtained using this model.

k-nearest neighbors (knn)

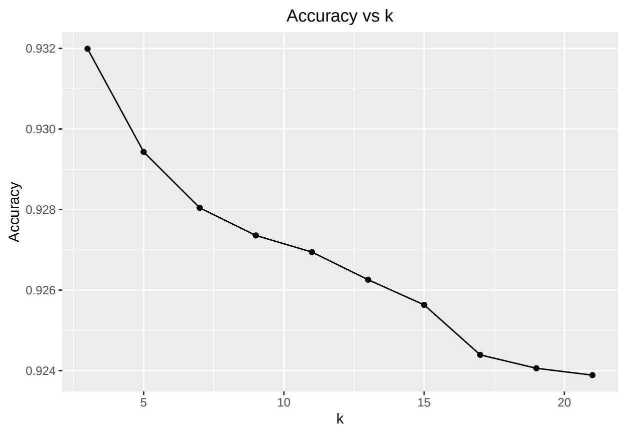
knn can be implemented using the following code:

```r
train_knn <- train(Result ~ ., method = "knn",
                   data = train_set)
```

However, to improve the performance of the algorithms, the value of k (a parameter that defines the number of nearest neighbors to include in the algorithm) can be manipulated as follows:

```r
train_knn <- train(Result ~ ., method = "knn",
                   data = train_set,
                   tuneGrid = data.frame(k = seq(3, 21, 2)))
```

The following plot shows the accuracy of the algorithm on the train set for different values of k:



The value of k that maximizes accuracy on the train set is obtained by:

```r
train_knn$bestTune
```

```
##   k
## 1 3
```

The algorithm is now applied on the test set.

```r
set.seed(820, sample.kind = "Rounding")
y_hat <- predict(train_knn, test_set, type = "raw")
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2260  154
##         1   189 2925
##
##                Accuracy : 0.938
##                  95% CI : (0.9313, 0.9442)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8741
##
##  Mcnemar's Test P-Value : 0.06638
##
##             Sensitivity : 0.9228
##             Specificity : 0.9500
##          Pos Pred Value : 0.9362
##          Neg Pred Value : 0.9393
##              Prevalence : 0.4430
##          Detection Rate : 0.4088
##    Detection Prevalence : 0.4367
##       Balanced Accuracy : 0.9364
##
##        'Positive' Class : -1
##
```

An accuracy of 93.8% is obtained using this model.

Support Vector Machines

Support Vector Machines can be used in the following way:

```
train_svm <- train(Result ~ ., method = "svmLinearWeights2",
                   data = train_set)
```

The algorithm is applied on the test set.

```
set.seed(820, sample.kind = "Rounding")
y_hat <- predict(train_svm, test_set, type = "raw")
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2245  155
##         1   204 2924
##
##                Accuracy : 0.9351
##                  95% CI : (0.9282, 0.9414)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8681
##
##  Mcnemar's Test P-Value : 0.0113
##
##             Sensitivity : 0.9167
##             Specificity : 0.9497
##          Pos Pred Value : 0.9354
##          Neg Pred Value : 0.9348
##              Prevalence : 0.4430
##          Detection Rate : 0.4061
##    Detection Prevalence : 0.4342
##       Balanced Accuracy : 0.9332
##
##        'Positive' Class : -1
##
```

An accuracy of 93.51% is obtained using this model.

Gradient Boosting Machines

Gradient Boosting Machines can be used in the following way:

```
fitControl <- trainControl(method = "repeatedcv", number = 4, repeats = 4)
train_gbm <- train(Result ~ ., data = train_set,
                   method = "gbm", verbose = FALSE, trControl = fitControl)
```

Here, cross validation (4 fold, repeated 4 times) has been used. In other words, the data set is first randomly spit into 4 parts. Each of these 4 parts is used for testing, while the other 3 parts are used for training. The entire process is repeated 4 times. Cross validation deters overfitting of the model. It is particularly useful here since gradient boosting machines can be prone to over fitting.

The algorithm is applied on the test set.

```
set.seed(820, sample.kind = "Rounding")
y_hat <- predict(train_gbm, test_set, type = "raw")
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

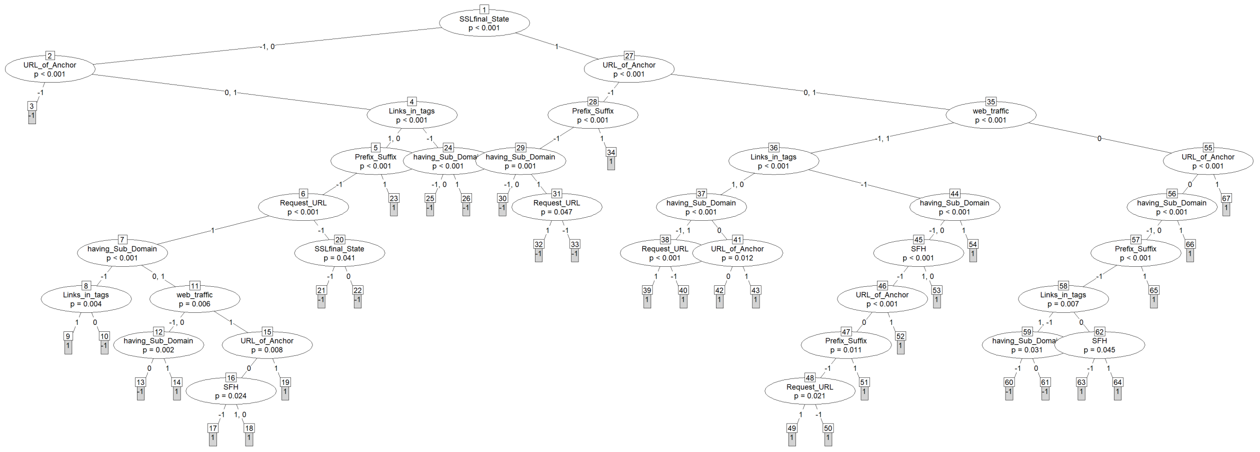
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2281  173
##         1   168 2906
##
##                Accuracy : 0.9383
##                  95% CI : (0.9316, 0.9445)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.875
##
##  Mcnemar's Test P-Value : 0.8285
##
##             Sensitivity : 0.9314
##             Specificity : 0.9438
##          Pos Pred Value : 0.9295
##          Neg Pred Value : 0.9453
##              Prevalence : 0.4430
##          Detection Rate : 0.4126
##    Detection Prevalence : 0.4439
##       Balanced Accuracy : 0.9376
##
##        'Positive' Class : -1
##
```

An accuracy of 93.83% is obtained using this model.

Decision Tree

Decision trees are helpful in visualizing and representing decision making, particularly for categorical variables. For the given data set, a decision tree can be easily created using the following code:

Following is the constructed decision tree.



1 SSLfinal_State p < 0.001
2 URL_of_Anchor p < 0.001
3 -1
4 Links_in_tags p < 0.001
5 Prefix_Suffix p < 0.001
6 Request_URL p < 0.001
7 having_Sub_Domain p < 0.001
8 Links_in_tags p = 0.004
9 1
10 -1
11 web_traffic p = 0.006
12 having_Sub_Domain p = 0.002
13 -1
14 1
15 URL_of_Anchor p = 0.008
16 SFH p = 0.024
17 1
18 1
19 1
20 SSLfinal_State p = 0.041
21 -1
22 -1
23 1
24 having_Sub_Domain p < 0.001
25 -1
26 -1
27 URL_of_Anchor p < 0.001
28 Prefix_Suffix p < 0.001
29 having_Sub_Domain p = 0.001
30 -1
31 Request_URL p = 0.047
32 -1
33 -1
34 1
35 web_traffic p < 0.001
36 Links_in_tags p < 0.001
37 having_Sub_Domain p < 0.001
38 Request_URL p < 0.001
39 1
40 1
41 URL_of_Anchor p = 0.012
42 1
43 1
44 having_Sub_Domain p < 0.001
45 SFH p < 0.001
46 URL_of_Anchor p < 0.001
47 Prefix_Suffix p = 0.011
48 Request_URL p = 0.021
49 1
50 1
51 1
52 1
53 1
54 1
55 URL_of_Anchor p < 0.001
56 having_Sub_Domain p < 0.001
57 Prefix_Suffix p < 0.001
58 Links_in_tags p = 0.007
59 having_Sub_Domain p = 0.031
60 -1
61 -1
62 SFH p = 0.045
63 1
64 1
65 1
66 1
67 1

The algorithm is now applied on the test set.

```r
set.seed(820, sample.kind = "Rounding")
y_hat <- predict(output.tree, test_set)
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    -1    1
##         -1 2243  151
##         1   206 2928
##
##                Accuracy : 0.9354
##                  95% CI : (0.9286, 0.9418)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8688
##
##  Mcnemar's Test P-Value : 0.004263
##
##             Sensitivity : 0.9159
##             Specificity : 0.9510
##          Pos Pred Value : 0.9369
##          Neg Pred Value : 0.9343
##              Prevalence : 0.4430
##          Detection Rate : 0.4058
##    Detection Prevalence : 0.4331
##       Balanced Accuracy : 0.9334
##
##        'Positive' Class : -1
##
```

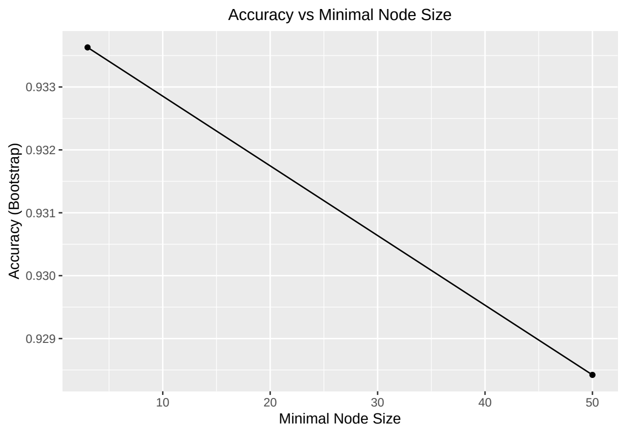
An accuracy of 93.54% is obtained using this model.

Random Forest

Random forests are an ensemble of several decision trees (based on the bagging method). These decision trees can be trained from different subsets of the train set. The following code constructs a random forest for the train set.

```
train_rf <- train(Result ~ .,
                    method = "Rborist",
                    tuneGrid = data.frame(predFixed = 2, minNode = c(3, 50)),
                    data = train_set)
```

Here, algorithms based on the different values of the minimum node size have been built. A plot of these values against the accuracy on the train set is shown below.



The algorithm is applied on the test set.

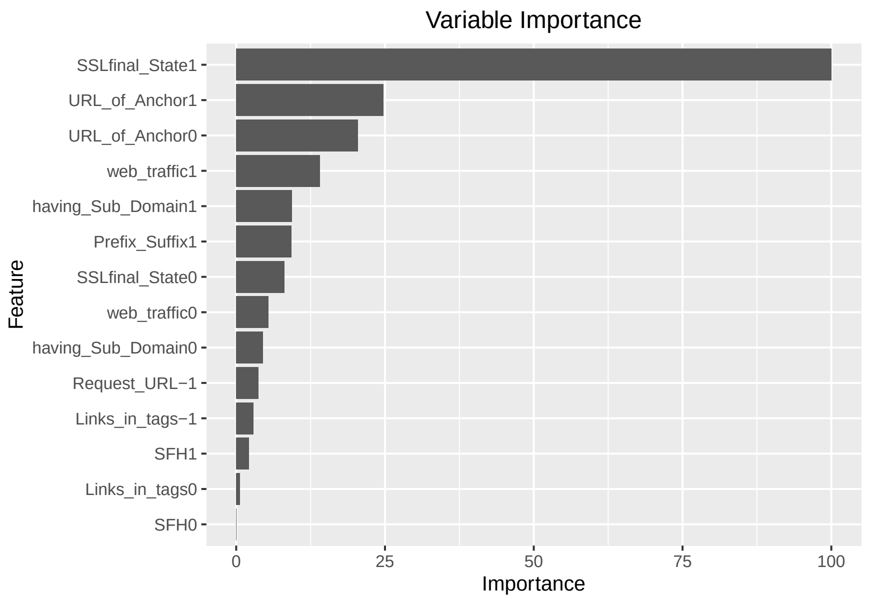
```
set.seed(820, sample.kind = "Rounding")
y_hat <- predict(train_rf, test_set, type = "raw")
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2260  153
##         1   189 2926
##
##                Accuracy : 0.9381
##                  95% CI : (0.9315, 0.9443)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8744
##
```

```
##  Mcnemar's Test P-Value : 0.05841
##
##             Sensitivity : 0.9228
##             Specificity : 0.9503
##          Pos Pred Value : 0.9366
##          Neg Pred Value : 0.9393
##              Prevalence : 0.4430
##          Detection Rate : 0.4088
##    Detection Prevalence : 0.4365
##       Balanced Accuracy : 0.9366
##
##        'Positive' Class : -1
##
```

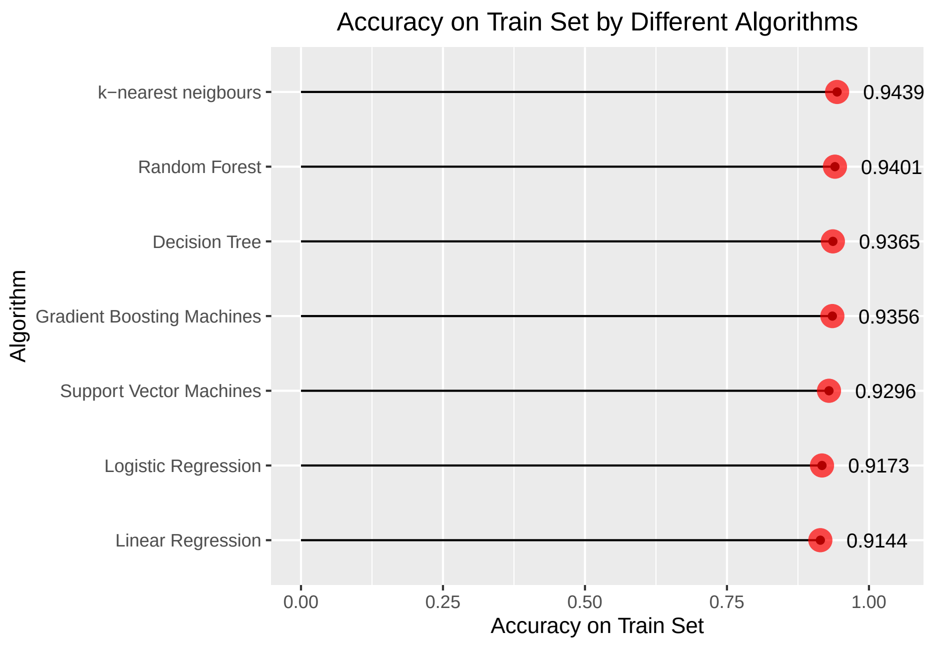
An accuracy of 93.81% is obtained using this model.

The importance given to the different predictors in this algorithm can be visualized through the plot below:



Ensemble

Ensembling is the final strategy that will be used in this project to maximize the accuracy on the test set. For ensemble, the four algorithms that have yield maximum accuracy on the train set will be used (since it is presumed that the outcome on the test set is unknown). The following graphs plots the accuracy on the train set for the algorithms that have been used so far.



It is important to note that decision trees are excluded from the ensemble model since random forest is already a bagging ensemble of decision trees. Hence, the four algorithms that are chosen for the ensemble are k-nearest neighbors, Support Vector Machines, Gradient Boosting Machines and Random Forest. The following codes make and run the required algorithm.

```r
#Generating the models:
models <- c("rf", "gbm", "knn", "svmLinearWeights2")
set.seed(820, sample.kind = "Rounding")
fits <- lapply(models, function(model){
  print(model)
  train(Result ~ ., method = model, data = train_set)})

#Applying the models on the test set and creating a data frame:
set.seed(820, sample.kind = "Rounding")
mod <- sapply(fits, function(fit){
  predict(fit, test_set)})
mod <- as.data.frame(mod, stringsAsFactors =TRUE)
colnames(mod) <- models
```

The above code creates a table of predictions made by each of the 4 algorithms for every set of observed features in the test set. The head of the table can be viewed below:

Table 3: First 6 Rows of the Data Set Mod

rf	gbm	knn	svmLinearWeights2
-1	-1	-1	-1
-1	-1	-1	-1
1	1	1	1
-1	-1	-1	-1
1	-1	1	1
-1	-1	-1	-1

The final predictions are then made based on the frequency of the predicted outcome by each of these algorithms. In case of a tie i.e. two of the algorithms predict '1' and two others predict '-1', importance is given to '1' based on the fact that the prevalence of '-1' is slightly less than 0.5 in the train set.

```
mod_final <- mod %>% mutate(prop_1 = rowSums((mod == 1)/4)) %>%
  mutate(final_pred = ifelse(prop_1 >= 0.5,1,-1)) %>% select(-prop_1)
set.seed(820, sample.kind = "Rounding")
y_hat <- as.factor(mod_final$final_pred)
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2234  118
##         1   215 2961
##
##                Accuracy : 0.9398
##                  95% CI : (0.9332, 0.9459)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8774
##
##  Mcnemar's Test P-Value : 1.435e-07
##
##             Sensitivity : 0.9122
##             Specificity : 0.9617
##          Pos Pred Value : 0.9498
##          Neg Pred Value : 0.9323
##              Prevalence : 0.4430
##          Detection Rate : 0.4041
##    Detection Prevalence : 0.4255
##       Balanced Accuracy : 0.9369
##
##        'Positive' Class : -1
##
```

An accuracy of 93.98% is obtained using this model.

Weighted Ensemble

Ensembles worked well. However, an assumption was made that since there are more '1' s in the data set, a prediction of '1' should be made in case of a tie. There are 121 such ties. But, from the users' point of view, preventing false negatives is important i.e. protecting a user from accessing a phishing website on the pretext that it is legitimate. So it becomes important to change the strategy for these 121 instances. So, instead of basing the model on any assumption, a new technique has been implemented, wherein weights have been assigned to the models based on their accuracy on the train set. A new data frame with these weights is created and applied on the test set as follows:

```r
#Creating the required data set
set.seed(820, sample.kind = "Rounding")
#Applying the model on train set:
mod_train <- sapply(fits, function(fit){ predict(fit, train_set)})

#Calculating model wise accuracy on train set:
accuracy_model_wise <- colMeans(mod_train == train_set$Result)
df_model <- data.frame(model = models, accuracy = accuracy_model_wise)
mod_weighted <- data.frame(apply(mod, 2, function(x) as.numeric(as.character(x))))

mod_weighted$rf <- (mod_weighted$rf) *
  ((df_model %>% filter(model == "rf") %>% select(accuracy))[1,1])
mod_weighted$gbm <- (mod_weighted$gbm) *
  ((df_model %>% filter(model == "gbm") %>% select(accuracy))[1,1])
mod_weighted$knn <- (mod_weighted$knn) *
  ((df_model %>% filter(model == "knn") %>% select(accuracy))[1,1])
mod_weighted$svmLinearWeights2 <- (mod_weighted$svmLinearWeights2) *
  ((df_model %>% filter(model == "svmLinearWeights2") %>%
      select(accuracy))[1,1])

#Final predictions are calculated:
mod_weighted_final <- mod_weighted %>% mutate(rm = rowMeans(mod_weighted)) %>%
  mutate(final_pred = ifelse(rm>=0,1,-1)) %>% select(-rm)

#Appliance of the algorithm on test set:
y_hat <- as.factor(mod_weighted_final$final_pred)
ytest <- as.factor(test_set$Result)
confusionMatrix(y_hat, ytest)
```

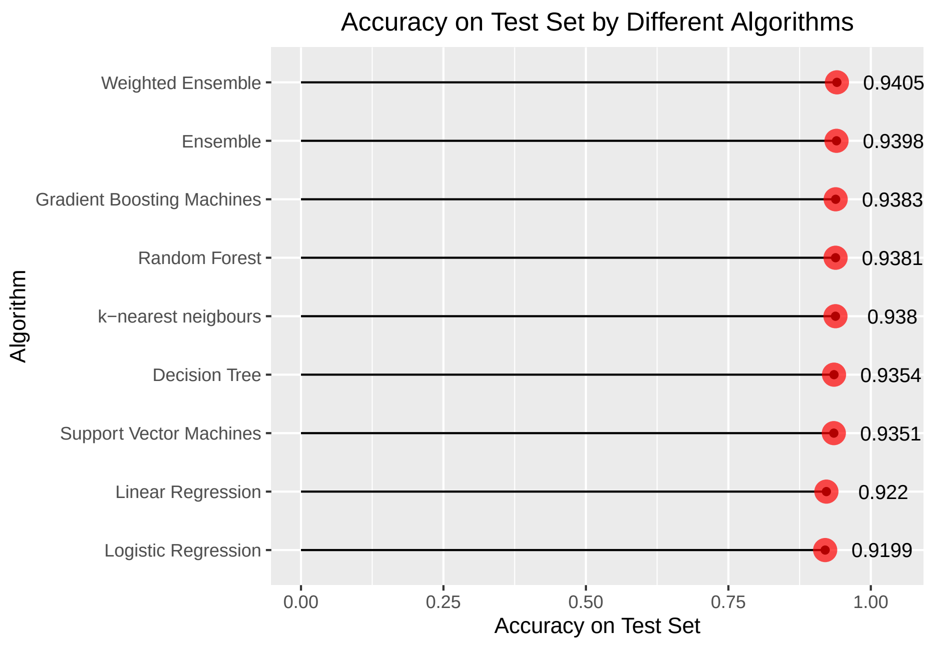
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   -1    1
##         -1 2275  155
##         1   174 2924
##
##                Accuracy : 0.9405
##                  95% CI : (0.9339, 0.9466)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8793
##
##  Mcnemar's Test P-Value : 0.321
```

```
##
##             Sensitivity : 0.9290
##             Specificity : 0.9497
##          Pos Pred Value : 0.9362
##          Neg Pred Value : 0.9438
##              Prevalence : 0.4430
##          Detection Rate : 0.4115
##    Detection Prevalence : 0.4396
##       Balanced Accuracy : 0.9393
##
##        'Positive' Class : -1
##
```

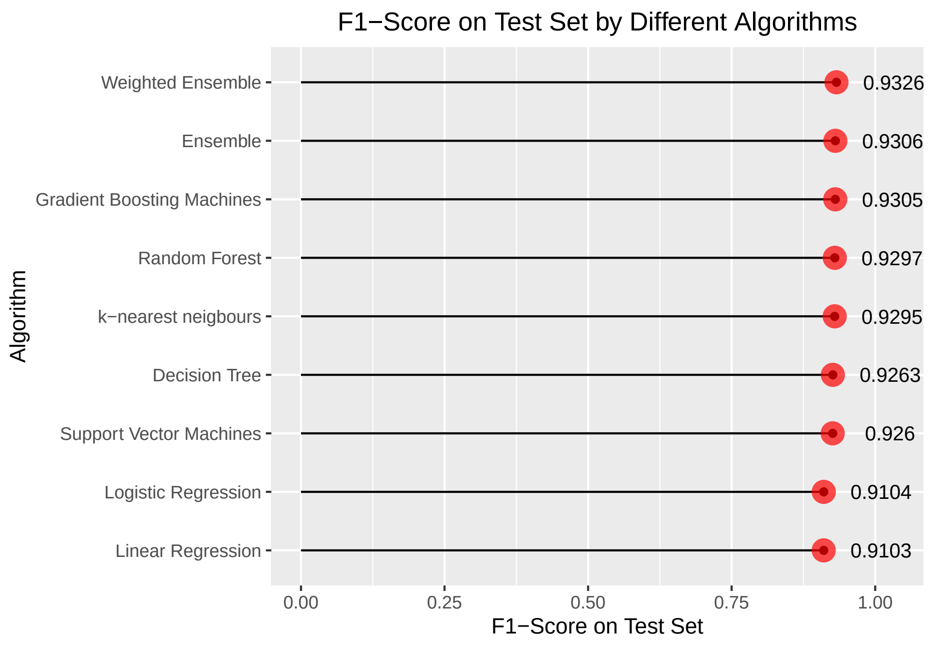
An accuracy of 94.05% is obtained using this model.

Results

All the algorithms that have been used have given different accuracies. The following graph compares the accuracies on the test set for these algorithms.



To get an idea about the balance between precision and recall, it becomes imperative to know the F1-Scores. The following graph compares the F1-Scores on the test set for the algorithms.



From these plots, it is evident that weighted ensemble has given the maximum accuracy of 94.05% on the test set.

Conclusion

Throughout the project, a variety of algorithms have been used, giving different measures of accuracy, precision and recall. Among the various algorithms, the weighted ensemble has yielded an accuracy of 94.05% on the test set. Both sensitivity and specificity are high, indicating a balanced performance.

Meanwhile, it is imperative to recognize the limitations of the project. Firstly, the computational time for some algorithms, particularly the ensemble models, is quite high (around 15-20 minutes). Secondly, the usage of these algorithms requires the users to make careful, detailed observations of websites, thereby making the utility cumbersome. Finally, the accuracy of the algorithm is limited due to a lack of predictors having a significant correlation with the outcome.

Future Considerations

To improve upon the performance of the algorithms, it is suggested that the algorithms be constantly retrained on fresh data sets. A possible future development could be to construct a platform which directly predicts website legitimacy without requiring the users to manually enter values for the predictors. Such a platform would have to read a website's URL, classify the different components, and then fill in the values for the features based on pre-defined rules. In addition, adaptive sorting algorithms can be developed to exploit the properties of a data set to greatly reduce computational time. Finally, new predictors can be explored to enhance the accuracy of the algorithms.

References

1. ‘Introduction to Data Science: Data Analysis and Prediction Algorithms with R’ by Rafael A. Irizarry
2. ‘R Markdown Cookbook’ by Yihui Xie, Christophe Dervieux, and Emily Riederer
3. ‘R Markdown: The Definitive Guide’ by Yihui Xie, J. J. Allaire, and Garrett Grolemund