IS 734 - Data Analytics for Cyber Security Project
Final Report

Detection of Phishing Websites

Submitted To: Prof. Faisal Quader

Priyadarshini Arcot: HW51333

Introduction:

The most frequent and pervasive kind of cybercrime nowadays is phishing scams. Phishing attacks can appear on a plethora of websites and domains, including E-commerce domains, online payment systems, websites with cross-origin citations, file hosting or cloud storage, and many others. These hacking attempts can potentially steal a person's sensitive data, including credit cards, bank accounts, personal data, social security numbers, photographs, and documents saved on cloud storage. People who are less familiar with social media, the Internet, and ecommerce may be more vulnerable to these cyberattacks. This effort is carried out to identify phishing websites to comprehend and raise awareness among people.

Problem Statement:

Phishing is done using social engineering tactics by a person or group to get personal information from unwitting consumers. Phishing emails are intended to appear as though a reputable company or well-known person sent them. These emails frequently try to persuade recipients to click a link that would lead them to a fake website that looks official. The user may then be prompted to submit private data, such as usernames and passwords for accounts, which could expose them to future breaches. These fake websites might also have dangerous software on them. The sophistication and difficulty of today's phishing attacks are increasing. According to a survey by Intel, 97% of cybersecurity professionals cannot differentiate between legitimate and phishing emails. As a result, finding fake websites is becoming more widespread.

Description:

We are creating a model to predict whether a particular URL is safe-listed, or block listed. We are doing this by considering the features of a URL and including factors like URL length, special characters, prefixes, suffixes, and more. Thus, identifying the websites based on these features. This model can help detect phishing sites and can help reduce damage caused by phishing attacks. To create this model, we extracted data from the below datasets. Specifically, we aim to create a classification-based model that will classify those URLs as legitimate and fake based on the feature classification.

Data Collection:

Dataset: Phishtank.com

Code: Python

The dataset has columns and important attributes like UrlLength, IpAddress, DomainInSubdomains, DomainInPaths, HttpsInHostname, HostnameLengh, NoHttps, Missing Title, and more.

Observations:

There are total 10,000 instances of URL in the dataset:
Attributes: 49
Classes: 4 (Minimum, Maximum, Mean, StdDev)

Data Preprocessing or Cleaning:

We used multiple steps to clean our data sets. There was not much of the cleaning process involved as we intended to use the data as it is. Though we handled some missing values, we faced challenges in recovering the missing values. For other attributes, we ignored the rows with missing values as the dataset had thousands of rows, which is already massive data. We uploaded the data into MySQL and used queries/functions to retrieve all the non-empty rows. As a part of cleaning, we dropped the columns which were not necessary and will be unused data. We used Python's drop () function to drop the columns from the dataset.

A sample of Dataset is shown Below:

id	NumDots	SubdomainLevel	PathLevel	UrlLength	NumDash	NumDashInHostname	AtSymbol	TildeSymbol	NumUnderscore	NumPercent	NumQueryComponents	NumAmpersand	NumHash	NumNumericChars	Nol
1	3	1	5	72	0	0	0	0	0	0	0	0	0	0	
2	3	1	3	144	0	0	0	0	2	0	2	1	0	41	
3	3	1	2	58	0	0	0	0	0	0	0	0	0	0	
4	3	1	6	79	1	0	0	0	0	0	0	0	0	0	
5	3	0	4	46	0	0	0	0	0	0	0	0	0	2	
6	3	1	1	42	1	0	0	0	0	0	0	0	0	0	
7	2	0	5	60	0	0	0	0	0	0	0	0	0	1	
8	1	0	3	30	0	0	0	0	0	0	0	0	0	3	
9	8	7	2	76	1	1	0	0	0	0	0	0	0	2	
10	2	0	2	46	0	0	0	0	0	0	0	0	0	0	
11	5	4	2	64	1	1	0	0	0	0	0	0	0	3	
12	2	0	2	47	0	0	0	0	0	0	0	0	0	1	
13	2	1	2	61	1	1	0	0	0	0	0	0	0	0	
14	2	1	3	35	0	0	0	0	0	0	0	0	0	0	
15	2	1	2	60	1	1	0	0	0	0	0	0	0	0	
16	3	0	4	73	0	0	0	0	0	0	0	0	0	0	
17	3	0	5	50	0	0	0	1	0	0	0	0	0	10	
18	3	1	2	59	1	1	0	0	0	0	0	0	0	7	
19	2	0	3	28	0	0	0	0	0	0	0	0	0	1	
20	1	0	4	59	0	0	0	0	0	0	0	0	0	22	
21	1	0	4	32	0	0	0	0	0	0	0	0	0	4	
22	5	1	2	52	0	0	0	0	0	0	0	0	0	1	
23	2	1	6	62	1	0	0	0	0	0	0	0	0	0	
24	1	0	10	105	2	0	0	0	0	0	0	0	0	24	
25	4	1	2	55	0	0	0	0	0	0	0	0	0	0	
26	5	0	3	134	3	0	0	0	0	0	0	0	0	1	
27	2	0	3	43	0	0	0	0	0	0	0	0	0	0	
28	6	0	3	210	2	0	0	0	7	0	5	4	0	24	
29	2	0	7	135	2	0	0	0	3	0	2	1	0	1	

Data Analysis:

A URL is a protocol used to indicate the location of data on a network. The URL comprises the protocol, subdomain, primary domain, top-level domain (TLD), and path domain. The subdomain, primary domain, and TLD are collectively referred to as the domain.

URL Features:

Sr. No	Feature name	Description
1	IP address	Whether domain is in the form of an IP address
2	Length of URL	Length of URL
3	Suspicious character	Whether URL has _@‘, _//‘
4	Prefix and suffix	Whether URL has _-‘
5	Length of sub domain	Length of sub domain
6	Number of _/‘	Number of _/‘ in URL
7	HTTPS protocol	Whether URL use https.

Features 1 to 4 is associated with suspicious URL patterns and characters. Characters such as _@' and _//' rarely appear in a URL. Feature 5 is defined for identifying newly created phishing sites. Currently, to prevent a user from recognizing that a site is not legitimate, phishing sites typically hide the primary domain; the URLs of these sites have unusually long subdomains. This feature can be additionally used to identify phishing sites that target vulnerabilities of smartphones, which have small displays that make it difficult to see the full URL.

Data Modelling and Training:

Ensemble methods are broadly categorized as Bagging, Boosting, and Stacking. Bootstrapping and Aggregating is referred to as bagging. Through the random selection of data, overfitting is intended to be avoided. Boosting is an ensemble strategy that tries to reduce bias and turn weak classifier into strong classifiers. Stacking combines the predictions of various weak learners using a meta-classifier.

Random Forest Classifier:

Random forest is a well-known bagging ensemble machine learning process that trains the model on several portions of the same training set using various deep decision trees, whose individual results are then averaged to produce the final classification. This method seeks to reduce variance. The Gini Index is used to determine whether nodes on a decision tree should branch during categorization.

To create a more reliable prediction model, one can combine several kinds of algorithms or use the same techniques more than once in ensemble learning. The random forest algorithm is used for classification and regression tasks; it mixes many algorithms of the same type, i.e., several decision trees, emerging in a forest of trees, hence the name "Random Forest." As the name implies, and the more trees there are, the more resilient the forest appears. Larger the number of trees in the forest, the higher the accuracy of the results produced by the random forest classifier.

$$Gini = 1 - \sum_{i=1}^{c} (P_i)^2$$

where c is the number of classes and Pi is the relative frequency of the observed class. By deducting the sum of the squared probabilities for each class from 1, the Gini Index is calculated. As a result, the class with the lowest Gini Index value receives the most priority because its likelihood of making a mistake is lower. Because it favors big partitions and has an easy implementation, the Gini Index is useful.

Data Modeling and Training Steps:

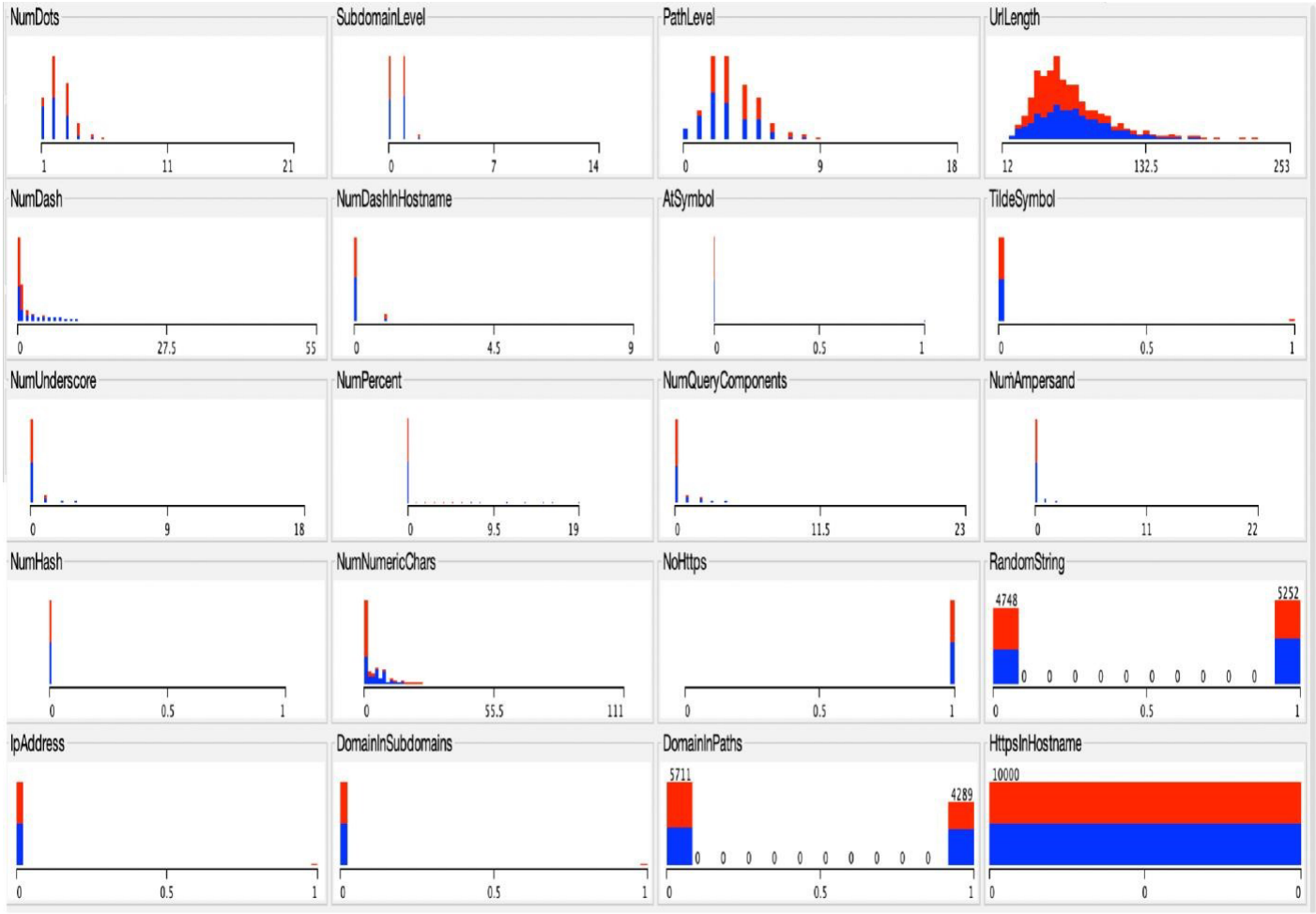
The basic steps in applying the random forest algorithm are as follows:

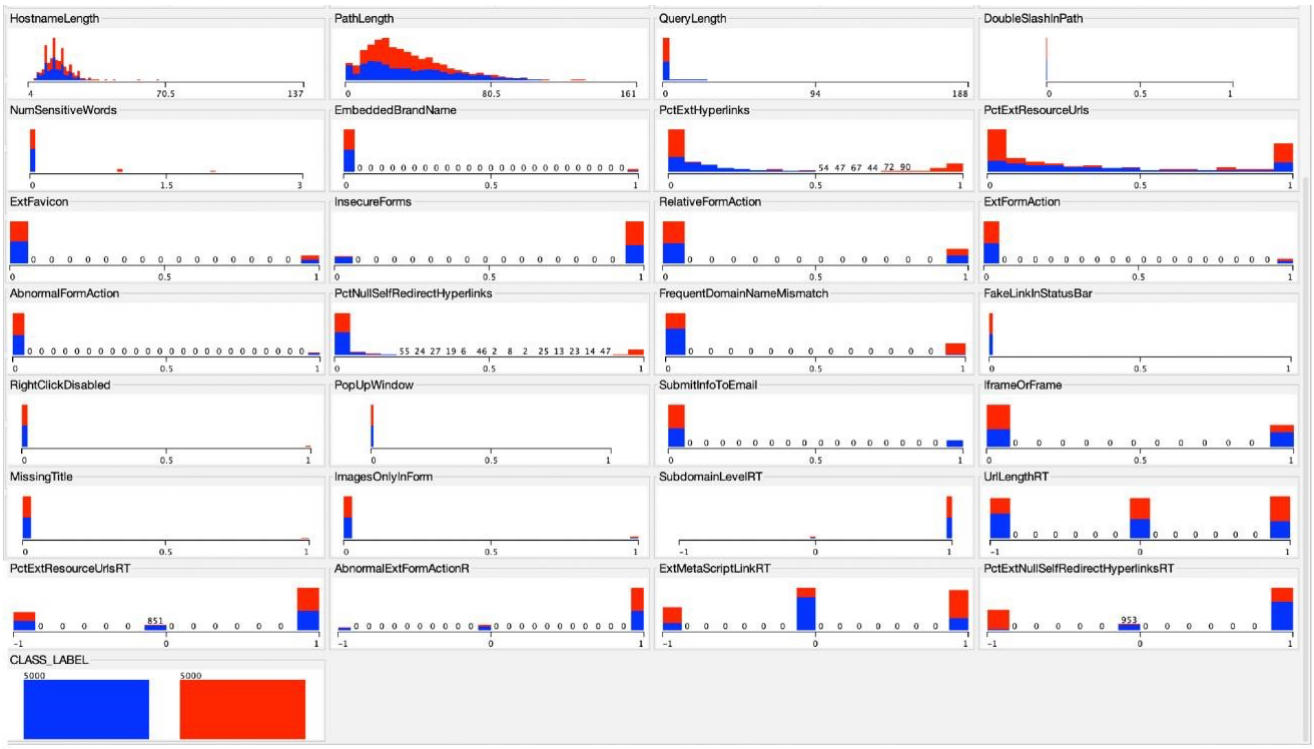
- Select N records at random from the dataset.
- Based on these N records, construct a decision tree.
- Repeat steps 1 and 2 till you have the required number of trees you want in your algorithm.

For RF, a model is constructed using a default set of K=10 estimators. The number of estimators represents the total number of forest trees. The maximum depth is unlimited for each tree. The training data is passed for model development once all the key parameters have been set, and the testing data is passed for performance evaluations. As for the splits, the min sample split parameter, which specifies the number of samples needed to split an internal node, is set to 2. Additionally, the minimum number of samples and weighted proportion of the total weights that must be present at these internal/leaf nodes are both set at 1 and 0, respectively.

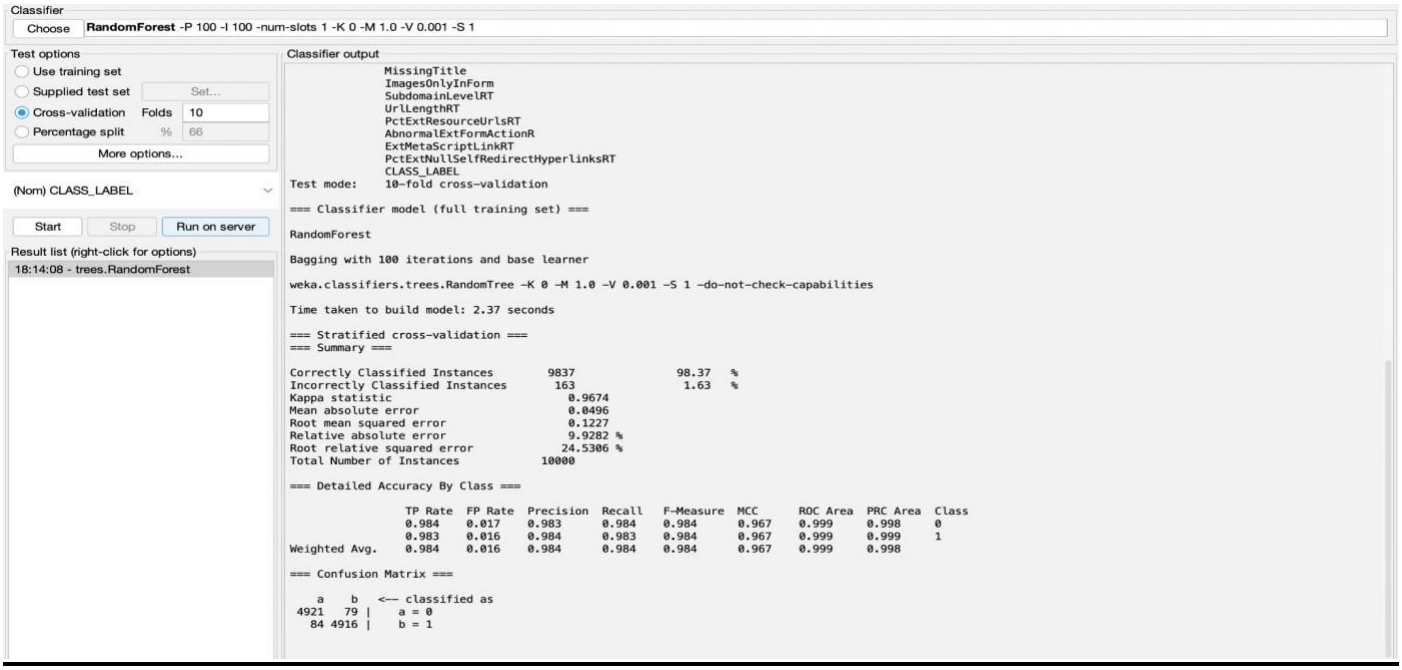
Results Obtained for visualization of RF in WEKA:

Tool used: **Weka Visualization of all Instances**


NumDots
1
11
21
SubdomainLevel
0
7
14
PathLevel
0
9
18
UrlLength
12
132.5
253
NumDash
0
27.5
55
NumDashInHostname
0
4.5
9
AtSymbol
0
0.5
1
TildeSymbol
0
0.5
1
NumUnderscore
0
9
18
NumPercent
0
9.5
19
NumQueryComponents
0
11.5
23
NumAmpersand
0
11
22
NumHash
0
0.5
1
NumNumericChars
0
55.5
111
NoHttps
0
0.5
1
RandomString
4748
5252
0 0 0 0 0 0 0 0 0 0
0
0.5
1
IpAddress
0
0.5
1
DomainInSubdomains
0
0.5
1
DomainInPaths
5711
4289
0 0 0 0 0 0 0 0 0 0
0
0.5
1
HttpsInHostname
10000
0
0
0



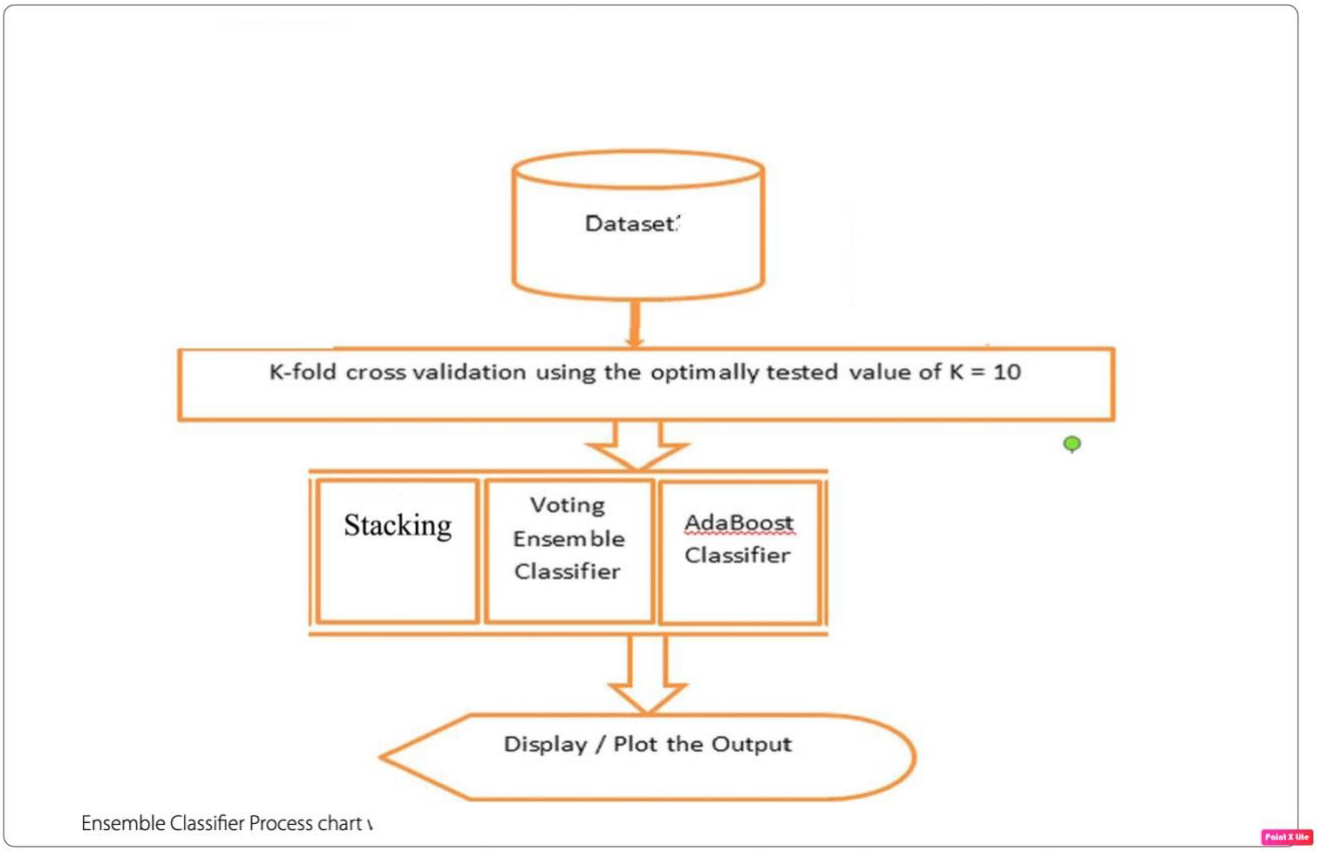
The Random Forest Classifier trained result



From the results that we have obtained from the Weka Visualization and Random Forest Classification trained model, we have 98% accurately classified data.

Further Analysis to Compare the Results with other Classifiers:

Once the model has been trained using the Random Forest classifier on the dataset, it is essential to compare our Base classifier with the different Ensemble models to verify the model's accuracy and the dataset's predicted values.



Procedure:

Step 1: The underlying classifier, Random Forest, is first trained using the dataset that will provide the output.

Step 2: The Voting, AdaBoost, and Stacking classifiers are fed the output produced in Step 1 as input.

Step 3: The voting algorithm determines the outcome with the highest number of votes anticipated to determine the final decision. The AdaBoost method is run concurrently to carry out sequential iterations until the model is predicted with the fewest errors and with the greatest accuracy possible. The Random Forest, Ada Boost, and Voting classifier outputs are now considered by the stacking algorithm, which accepts the heterogeneous output, and performs simultaneous predictions. These predictions are then provided as input to the meta-model for training to produce optimum results.

Step 4: The results from the Random Forest, AdaBoost, Voting, and Stacking classifiers are then plotted using the ROC curve, and their individual F1 Scores, Accuracy, and Runtimes are collected in a tabular format for comparison.

Boosting: A base model depends on its predecessors, which were learnt sequentially and deterministically by boosting, and frequently takes homogenous weak learners into account. The fundamental principle of boosting approaches is that we build a second model to correct any errors in the first one after building a model using the training dataset. Up till there are less errors and the dataset can be forecast reliably, this process is repeated.

Working of AdaBoost Algorithm:

When the random forest is used, the algorithm produces 'n' trees. With a start node and numerous leaf nodes, it produces n trees. A random forest has no predetermined depth, despite the possibility of larger or smaller trees. AdaBoost's method, on the other hand, only produces the Stump node with two leaves. These stumps are favored by boosting techniques due to their subpar learning capacities. The arrangement of the stumps is crucial in AdaBoost. The inaccuracy in the first stump influences how the others are built.

Stacking: Stacking is a method that regularly considers diverse weak learners, trains them concurrently, and then combines them by training a meta-model to provide a prediction based on the output of the diverse weak models.

Working of Stacking Algorithm:

Forecasts from other learned algorithms are incorporated into the Random Forest model. When using a weighted average ensemble, each member's contribution is given equal weight based on how confident they are in their capacity to provide the most accurate forecasts. The average model ensemble performs worse than the weighted average ensemble.

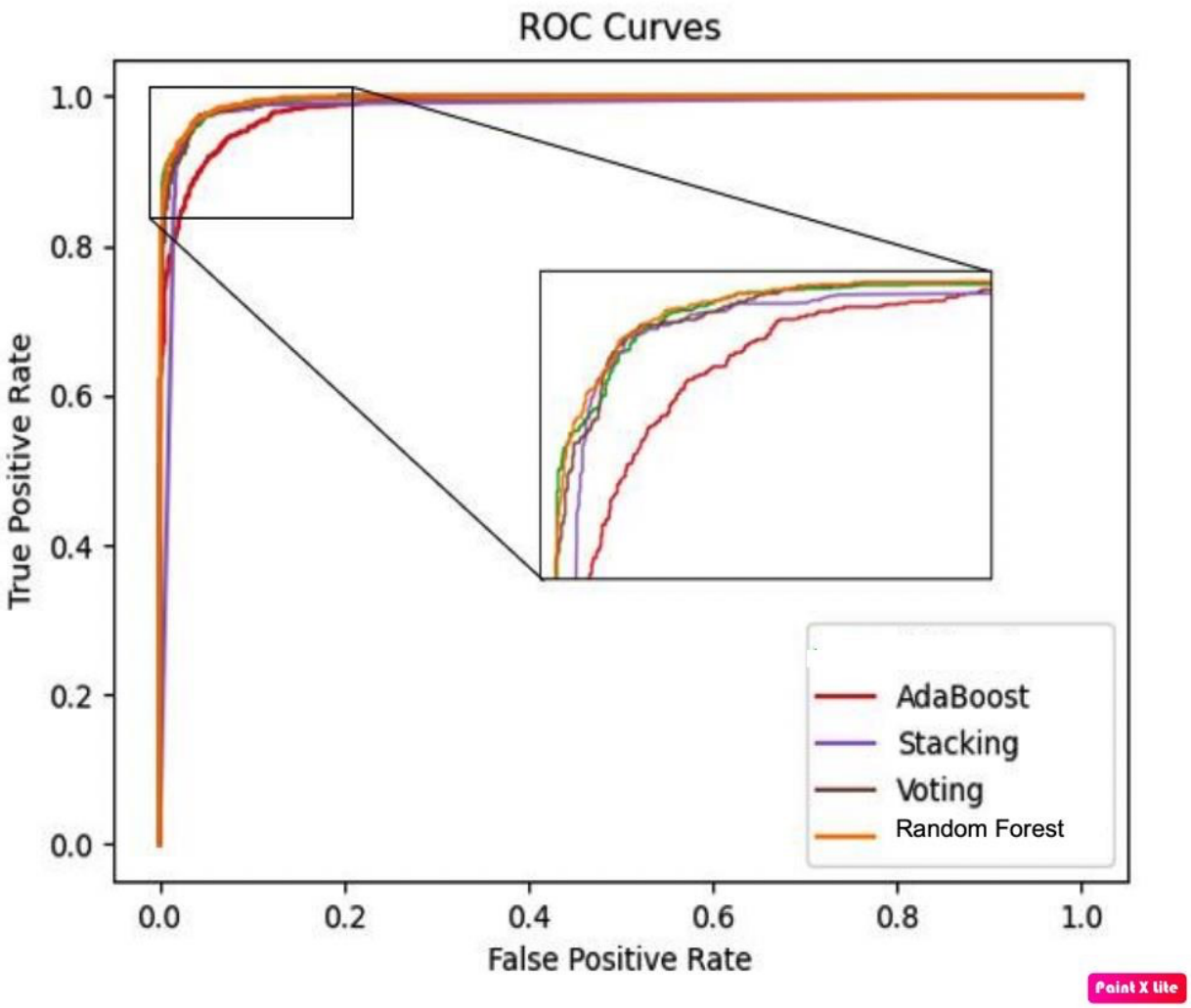
Voting: A voting classifier is a machine learning technique that creates multiple base models or estimators and forecasts based on summing their outcomes. The aggregating criteria can be combined with voting for each estimator output.

Working of Voting Algorithm:

Step 1: The dataset is sampled at random in step one.
Step 2: Individual sample decision trees are built. The prediction outcome will then be obtained by the algorithm from the created decision trees.
Step 3: Voting will be conducted for each expected outcome.
Step 4: The outcome that receives most votes will be the outcome of the forecast.

Performance Measuring and Results

The performance of RF model is calculated by analyzing the ROC curve, which helps us to understand accuracy, F1 score, and time taken to execute the model.
The ratio of correct predictions to all the model's other predictions is used to measure accuracy.
The number of accurate predictions the model made is known as precision.
The precision and recall mean are known as the F-score. Below is the ROC curve generated based on the evaluations performed using python for the dataset.



ROC can also be applied to cost-benefit calculations when making decisions. It is used to plot various specificity cutoff points, often known as false-positive rates. These points stand for a pair of sensitivity and specificity in relation to a certain choice. A ROC curve that is very nearly in the upper left corner indicates greater accuracy and around 100% sensitivity and 100% specificity.

It is clear from Figure above that none of the AUC Scores for the ROC Curves generated by the baseline models outperform the AUC Score from the proposed RF model. More specifically, the AUC Score obtained from the parameter-tuned RF and the AUC Score derived from the curve of the proposed model are very close. However, the results from the Stacking and Voting models' curves are a little bit poorer than the results from the proposed model's curve. Furthermore, the poorer score of the AdaBoost model is evident through comparison. The recommended RF strategy thus outperforms all pertinent baseline methods when the results are considered.

The Results for all the base classifiers are shown below:

	Stacking	Adaboost	Random Forest	Voting
F1 Score	0.9605	0.9331	0.9643	0.9643
Accuracy	0.9605	0.9331	0.9643	0.9643
Runtime	6.0123s	0.3421s	0.2840s	8.3052s

Conclusion:

In this project, we evaluated site URLs using ensemble classifiers such as Random Forest, Adaboost, Stacking, and Voting. These methods are all based on specific parameter tweaks that vastly enhance forecast accuracy. The training and testing data sets are also somewhat feature-rich and resemble real-world phishing scenarios. The experiment shows that the suggested strategy outperforms all other approaches in terms of projected accuracy (96.44%) and the overall learning and classification process runtime dimensions (0.2841 seconds).

Challenges Faced:

- Although supporting data formats were inserted into the WEKA tool, the evaluations of the selected datasets needed to be carried out accurately. It was a usual struggle to develop new classification techniques and carry out the jobs to acquire the applicable findings.
- The subsequent challenge for us was to put the data into Weka because it did not support that format (CSV). Although Weka usually supports CSV, we are still determining why it was not working with that file. So, we attempted a few more methods until we could accurately insert the file.

References:

[1]:https://www.researchgate.net/publication/340695779_Detecting_Phishing_Website_Using_Machine_Learning

[2]: https://en.wikipedia.org/wiki/Phishing

[3]: https://phishtank.com/index.php

[4]: https://towardsdatascience.com/getting-started-with-weka-3-machine-learning-ongui-7c58ab684513

[5]: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[6]: https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/