PERFORMANCE COMPARITIVE STUDY OF MACHINE LEARNING ALGORITHMS FOR PHISHING URL DETECTION

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Chinmaye, 121910318009
B Srestha, 121910318016
B Shyam Kumar, 121910318044
D S S P Venkat, 121910318047

Under the esteemed guidance of

Dr. K Nitalaksheswara Rao,

M.Tech, Ph.D,

Asst. Professor

DEPARTMENT OFCOMPUTER SCIENCE & ENGINEERING

GITAM

(Deemed to be University)

VISAKHAPATNAM

OCTOBER 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**PERFORMANCE COMPARITIVE STUDY OF MACHINE LEARNING ALGORITHMS FOR PHISHING URL DETECTION**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 31-10-2022

Registration No(s).	**Name(s)**	**Signature(s)**
121910318009	**Chinmaye**	
121910318016	**B Srestha**	
121910318044	**B Shyam Kumar**	
121910318047	**D S S P Venkat**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**PERFORMANCE COMPARITIVE STUDY OF MACHINE LEARNING ALGORITHMS FOR PHISHING URL DETECTION**" is a bonafide record of work carried out by **Chinmaye 121910318009, B Srestha 121910318016, B Shyam Kumar 121910318044, D S S P Venkat 121910318047** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Head of the Department**
Dr. K Nitalaksheswara Rao	**Dr. R Sireesha**
M. Tech, Ph. D, Asst. Professor	**M.S., Ph. D, Professor**

TABLE OF CONTENTS

1. ABSTRACT

Phishing attack is one of the simplest ways to obtain sensitive information from innocent users who are unaware. The main motive of the phishers is to acquire critical information like username, password and bank account details etc. using a fake link which actually looks like a genuine link from an authorized source. Users who have good technical knowledge might be able to identify these links but for naive users, this is going to be dangerous as it might lead to loss of privacy and assets. There are other techniques of spam detection based on the sender's information and content-based detection. In this paper we follow an efficient approach in which we can directly an URL if it is Phishing or not by its features using Machine Learning. The aim of this project is to build multiple Machine Learning models, compare their performances and narrow down to best model based on its performance and efficiency.

2. INTRODUCTION

Phishing attacks are limitedly defined as stealing of personal information from the users by a non-trusted source which is actually pretending to be an authorized source, however it is not always the case. A link or site is said to a phishing whenever it act as a trusted party, when it is actually not, with a motive to confuse the naive users and make them to perform an action which they would do only with the trust of the party.

The phishing link might grab our personal information and misuse it or there are cases in which the link might grab our bank account details or any authentication details of online assets to manipulate the accounts. There is also another possibility of damage with this phishing attacks where clicking the phishing link activates a worm into the user's device. This worm may give unnecessary permission to the cyber criminals to get over the total control and access of the device. Also, sometimes the worm may cause damage to software and hardware of the system.

Phishing is basically a social engineering activity. The cyber criminals who involve in making these phishing attacks are known as phishers. Although phishing is the simplest attack possible to obtain information illegally, it is totally dependent on human weakness. General people may refer these attacks as 'Hacking' but it's a misconception. This is just a trick being played by phishers to trap users.

Naive users get into these traps easily as the link seems to be almost real. In recent times, the number of phishing cases are increasing in India. The current spam detection techniques which are being used are based on sender's reputation and content-based identification, but the phishers are using trustful identities to send the links with generic content as the link is just enough to fulfil their motives. In this case, phishing can only be prevented by identification of the URL links.

URL means Uniform Resource Locator. Just like how we have address to any particular location, an URL is an address for location where the resources are stored on the internet. So, these URLs will contain a lot of information. The elements present in the URL, domain information and the content of the link plays a major role in describing an URL.

Classification is one of the Machine Learning concepts which can help in detecting Phishing attacks. The differentiation between a benign link and phishing link is possible. The URL link which is being sent by the phishers will be having certain features both internally and externally which will be used to decide whether the link is phishing link or a benign one. Initially, two classes named ‘Phishing’ and ‘Benign’ are considered. Based on the features of the URL, the prediction is made on which class the URL belongs to. This classification approach can be implemented by using various supervised learning algorithms suitable for classification.

3. LITERATURE REVIEW

While the use of social engineering began to rise in the world, Phishing has been a simple and convenient way to manipulate naive users and obtain their data maliciously. The first phishing attack was found to be happened in the mid 1990’s and targeted America Online users. The victims unknowingly provided their login details in the phishing links and the Phishers started using the victim’s accounts for spamming and adding likes[1].

In the year 2000, people used to receive mails with title ‘I LOVE YOU’ attached with a love letter. Users who clicked on the letter, it initiated a worm which obtained all the personal image files and sent it to all the contacts in the outlook[2].

In current times, particularly in India, people are receiving a lot of links pretending to be that they are from official banks or from government saying that the user need to immediately update his identity details like Aadhar no., PAN no. etc. Innocent users tend to click on those links and provide the information thinking that it is genuine. The government and network providers are not able to do anything more than warn the users to stay away from these links as there is no responsible source other than human weakness of the victim[3].

There are various naive methods to identify phishing attacks but most of them are inefficient. Also, sometimes it really doesn’t matter even if the detection worked after the attack has been committed as the loss has been happened already. So, there needs to be a better approach. The spam detection techniques which are currently in use, which can be seen in SMS applications and Mail applications are based on content based detection and sender’s behaviour[6][7].

Sometimes people tend to receive mails stating that they have won lotteries, lucky draws or any other gifts and asks the user to click there to claim their prize. These kind of mails can be detected by content based detection approaches.

We might see messages and mails getting thrown into spam based on the sender. The users reputation is based on his history, the type of messages or mails he was sending, number of reports on him, etc. Also, the way of style he maintains, like some patterns can be considered. In this way, the detection can be done based on the sender's behaviour and statistics.

The above approaches can be useful, but a more powerful methodology is required in order to protect the naive users from phishing. Rather than detecting the mail or message, detecting the URL would be a better way. As we have discussed earlier, the URL detection is totally based on its features, but what exact features? is the biggest question. According to past studies, the features are majorly classified into 3 types: Address-bar based features, HTML-JavaScript based features, Domain based features[4]. The address bar based features are directly taken from the URL string. The HTML-JavaScript based are taken by crawling through the source code. The domain based features are taken from the WHOIS database, DNS records, etc[4].

Based on different kinds of features, many studies have been evolved around the Machine Learning. Using different kinds of datasets, different sets of features, and different algorithms resulted in various outcomes. According to Machine Learning, the suitable concept used for these kind of problems is Classification[8]. Generally, a classifier is built in order to decide to which class a particular input belongs to[8]. Here, the URL, based on its features, need to be classified into a class which it belongs to, either Phishing URL or Benign URL. From [5], the proposed algorithm was SVM, Support Vector Machines, performing with 95.66% of accuracy.

4. PROBLEM IDENTIFICATION AND OBJECTIVES

PROBLEM STATEMENT

Phishing is a type of social engineering which involves sending of fraudulent communication that appear to come from a trusted source. Phishing attacks have become very common these days as it is an easy, cheap and effective approach to manipulate innocent users and steal their data. The cyber criminals who initiate Phishing attacks are known as Phishers. The main motive behind these attacks is to obtain sensitive information from users and use it for illegal activities. Hence, there needs to be a solution to prevent users from falling into this trap. Studies have shown that the Phishing URLs can be differentiated from the Genuine URLs based on some features, both externally and internally. By using various methods of classification from Machine Learning, we try to conclude the best approach according to the situation.

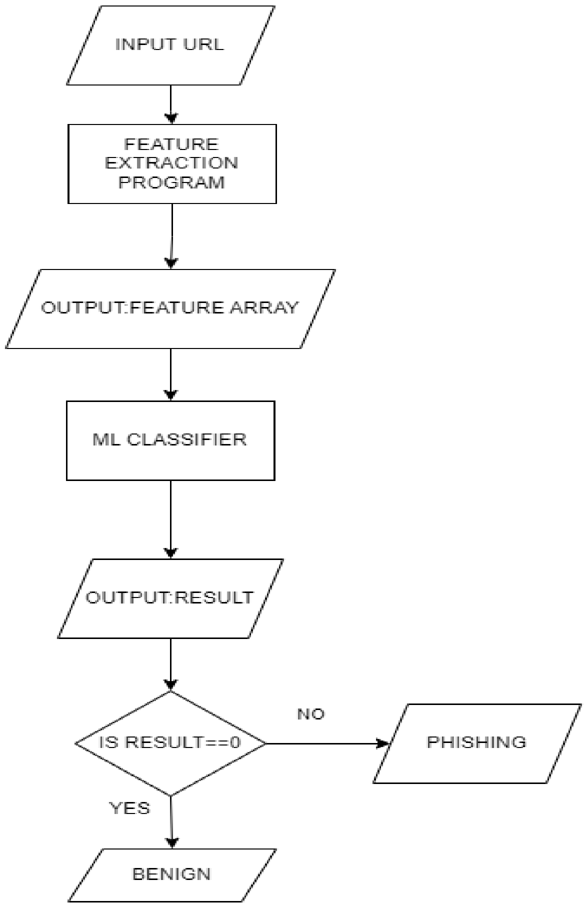
OBJECTIVES

- To obtain the important information from the URL through Web scraping.
- To determine and extract the best possible set of features.
- To decide the algorithms that should be used based on the situation.
- To understand the concept and mathematics behind the algorithms.
- To build multiple classification models/classifiers using the algorithms.
- To compare the performance of the models based on their accuracy.
- To conclude the best methodology for phishing detection.

5. METHODOLOGY

SYSTEM ARCHITECTURE

The Website link/URL received by the user needs to be identified whether it is a Phishing URL Link or a Benign one. For that, the URL will be passed into the Feature Extraction Program, which is a python program written using various string techniques to extract the address bar based features and also used various web scraping techniques to extract its content based features and domain based features. After successful execution of the program, the Feature Extraction Program gives an output as an Array. The array contains all the required and extracted features of the URL and hence it is called the Feature Array.



The Feature Array which is a result of the feature extraction program is used as an input to the Classifier. The model makes a prediction for the feature array that has been provided and gives output as 0 or 1 as it is a binary class classification. '0' says that the URL is Benign and '1' says that the URL is Phishing.

DATASET

A dataset has been extracted from previously existing dataset and re-processed. According to the current scenario, only required columns have been taken into the new dataset. The dataset consists of 11,430 URLs in which 5,715 are Phishing and 5,715 are Benign. The Phishing URLs are labelled as '1' and the Benign URLs are labelled as '0'.

FEATURES

The features of the URL will be extracted by the Feature Extraction Program, written in python to extract the Address bar based features, Content based features (Internal) and Domain based features (External). The features are as follows:

1) <u>Length of the URL:</u> An URL is a simple string object. The length of the string is taken as a feature named 'Length of the URL'. Most of the phishing URLs seem to be long as there is a chance of hidden information in it.

2) <u>Presence of IP in the URL:</u> Most benign sites doesn't use IP in the URL to load the page. If IP present in URL, it is assumed to be malicious and marked as '1' else '0'.

3) <u>Number of dots (.) in the URL:</u> The number of dots present in the URL are counted and taken as a feature. Phishers try to trick uses by using reputated names in the domain part with multiple dots to make it look like a trusted link.

4) <u>Number of hyphens (-) in the URL:</u> The number of hyphens present in the URL are counted and taken as a feature. Phishers try to trick uses by using multiple hyphens with different combinations of words just like in the Number of dots, to make it look like a trusted link.

5) Number of ‘@’ in the URL: Phishers add “@” symbol in the URL make the browser to ignore everything preceding the “@” symbol and the real address is given after the “@” symbol. The count of “@” symbols is included as a feature.

6) Number of slashes (/) in the URL: The average number of slashes in a benign URL is found to be 5. So, a count of slashes present in an URL is considered as a feature.

7) Number of double slashes (//) in the URL: The presence of double slashes in the URL might lead to redirection into another page. A count of double slashes present in an URL is taken as a feature.

8) Presence of HTTP/HTTPS in domain part: Unlike the HTTP/HTTPS token before the domain part, phishers try to trick users by putting HTTP/HTTPS token in the domain part which might make users to feel the site is secure which is actually not. So, if the HTTP/HTTPS token is present in the domain part the feature is set to ‘1’, else set to ‘0’.

9) Ratio of digits in the URL: The ratio of number of digits present in the URL to the length of the URL is calculated and set as a feature. Benign URLs contains numbers but a lot of numbers might be suspicious.

10) Prefix-Suffix separated by dash symbol: Phishers add dash symbol to the domain name to confuse the and make them feel that they are dealing with a legitimate site. The dash symbol is not much used in benign URLs. If domain name separated by dash symbol then feature is set to ‘1’ else to ‘0’.

11) Using URL shortening services: There are a lot of URL shortening services like bit.ly, tinyurl, etc. which are used to hide a large URL. This is definitely a suspicious move as there might be a reason to hide the URL and also URL shortening services are not recommended to be safe. If the URL has URL shortening then it is set to ‘1’ else ‘0’.

12) Number of hyperlinks: The source code of the page is obtained in order to find the number of hyperlink tags. More number of links makes the site more suspicious. So, the number of hyperlinks is taken as a feature.

13) Using iframe tag: This feature can be extracted by crawling the source code of the URL. This tag is used to add another web page into existing webpage. Phishers can make use of the “iframe” tag and make it invisible, without frame borders. So, if there is an iframe it is set to ‘1’ else set to ‘0’.

14) Right click disabled: Phishers use JavaScript to disable the right-click function, so that users cannot view and save the webpage source code. This feature is considered and if right click is disabled, then the feature is set to ‘1’ else set to ‘0’.

15) Domain with copyright: The content of the page might contain copyright information of the domain for the benign sites. So, if the copyright information is not found, then the feature is set to ‘1’ else set to ‘0’.

16) WHOIS registered domain: Most of the benign URLs will be having records in the WHOIS database. As most phishing sites are temporary, they will not have WHOIS records. If the domain doesn’t have WHOIS records, then the feature is set to ‘1’ else set to ‘0’.

17) Domain Registration Length: The domain registration length is the number of days remaining for expiration from present. This is considered as a feature.

18) Domain Age: The age of the domain can be obtained from the WHOIS database. The age of the domain is the survival time of the domain, the time between its creation and present. The age of domain in days is taken as a feature.

19) Web traffic: This feature measures the popularity of a site based on its visitor count. Phishing sites are not much visited. So, the average count of visitors is taken as a feature.

20) DNS records: WHOIS database has all the DNS records. If no records found, then the feature is set to ‘1’ else ‘0’.

21) Google index: Google crawler ‘Googlebot’ visits all the existing sites and makes an index for it and it can be accessed through Google’s API. If the URL doesn’t have a Google index then the feature is set to ‘1’ else ‘0’.

22) Page rank: Every page is given an Open page rank on a scale of 1 to 10. The Open page rank is obtained from OPR API and set as a feature.

MACHINE LEARNING ALGORITHMS

KNN ALGORITHM

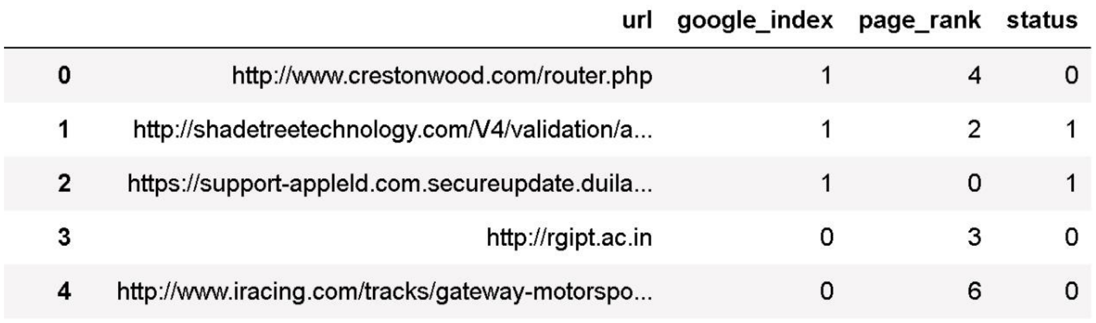
K-Nearest Neighbor is simply known as KNN algorithm. KNN is the simplest algorithm which can be used for both Regression and Classification. In the current scenario, we are using this algorithm for classification of URLs. KNN differentiate between the classes on the basis of distance between the datapoints. The metrics that are used to calculate the distance between the datapoints are Euclidean distance, Manhattan's distance, etc.

WORKING OF KNN ALGORITHM FOR CLASSIFICATION:

1) The training data will be given to fit the model.
2) A K-value is taken in order to proceed further, basically the K-value is taken as square root of n, where n is the size of training data. Also, the K-value is recommended to be a value due to its effect on classification.
3) When a new data point is need to be predicted or classified, then the distance between new data point and each point from the training data is calculated.
4) Based on the distance, the K nearest data points are taken. For example, if k=5, then the closest 5 data points are considered according to the distance.
5) From the resultant data points, the label of class of the majority data points is considered to be the class of the new data point.

MATHEMATICAL EXPLAINATION:

In order to get a clear understanding of the working of the algorithm, the mathematics behind the algorithm should be known. A small piece of data is taken from the original dataset which is being used and classification is made for a new data point manually.



	url	google_index	page_rank	status
0	http://www.crestonwood.com/router.php	1	4	0
1	http://shadetreetechnology.com/V4/validation/a...	1	2	1
2	https://support-appleld.com.secureupdate.duila...	1	0	1
3	http://rgipt.ac.in	0	3	0
4	http://www.iracing.com/tracks/gateway-motorspo...	0	6	0

The above instances of data is considered for training.

Note: This might look vague as we are dealing with a sample for the sake of understanding the concept and mathematics behind the algorithm, but it is more effective while dealing with the real and huge data.

Now, when a new data point is need to be classified then the distance between new data point and each data point from the above data will be calculated as shown below.

URL	Google Index	Page Rank	Status	Co-ordinates	Distance
1	1	4	0	(1,4)	1
2	1	2	1	(1,2)	2.2360
3	1	0	1	(1,0)	4.123
4	0	3	0	(0,3)	2.2360
5	0	6	0	(0,6)	2.828
6	2	4	? (0)	(2,4)	

As representation of multiple dimensions is practically not possible, we have taken 2D vector space with 2 features as an example. The features are taken into vectors or co-ordinates and inserted in the column named ‘Co-ordinates’.

In the above table, let URL 6 be the new URL that needs to be classified. The distance from the URL 6 and the remaining URLs is calculated.

Here, the metric used to calculate distance is Euclidean Distance. It can be calculated as,

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}\,.$$

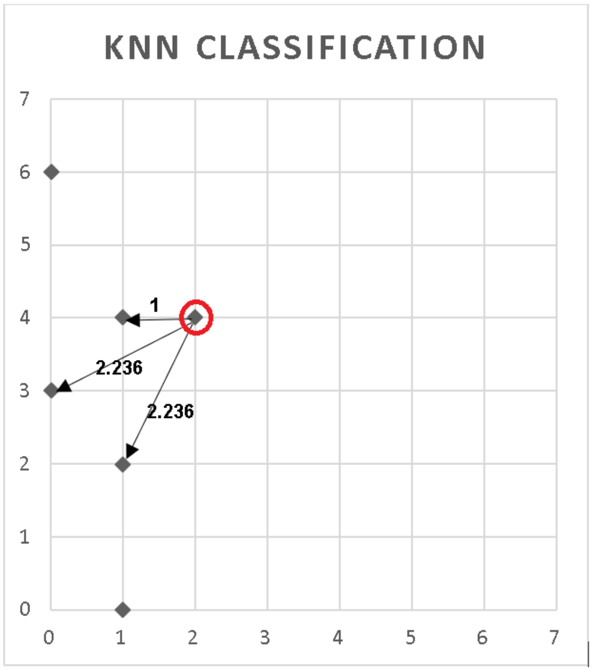
Where, n = number of dimensions/ n space,

x, y = two data points in the vector space.

The distance between two data points in a 2-Dimensional vector space can be calculated by,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

By using the above formula, the distance between the new data point and each data point in the training set is calculated and inserted in the column named ‘Distance’.



Assuming that K=3, we can observe the closest 3 data points to the new data point based on their distance.

Now, if we see the class labelled for that data points, the ‘Status’ column, two of them were marked 0, which means Benign and one of them was marked 1, which means Phishing.

As the majority of the neighbors were classified as 0, the new data point will also be classified as 0, which means the URL is not a Phishing URL.

The motive behind using this algorithm is to start with a simple approach. Sometimes, complicated problems can be solved by simple approaches. So, we gave it a try and implemented the model.

RANDOM FOREST ALGORITHM

Random forest is a supervised machine learning algorithm. It is a type of bagging algorithm in Ensemble learning. It is an algorithm built on the idea of integrating various classifiers to solve complex issues and enhance model performance. It is pretty popular as it is very powerful and versatile.

In Random forest, instead of depending on one decision tree, multiple decision trees are used for making a prediction based on the majority value. This algorithm can be used under regression and classification problems.

Random forest has many benefits. It requires less training time when compared to other algorithms. It also gives a high accuracy even with a large dataset. Most importantly, it maintains accuracy when a large amount of the dataset is not present.

Random forest is based on decision trees. The reason decision trees are not used is because the trees are sensitive to the training dataset, which causes high variance. Therefore, to avoid that in random forest, multiple decision trees are used with various data sets. This is the reason it is called Random forest. It has two processes that are executed randomly.

WORKING OF RANDOM FOREST ALGORITHM FOR CLASSIFICATION:

1) Generate multiple number of datasets from the main dataset by bootstrapping. The datasets generated are generally called bags. The bags will be containing few sample of rows and few sample of columns from the main dataset. This is done by row sampling and column sampling with replacement.

2) Each bag of dataset is used to create its individual decision tree based on its features to get the output.

3) The output of each decision will be the label of class.

4) Considering the decision trees, the class which is majority output of the trees is considered as the final output of the random forest.

XG BOOST ALGORITHM

XG Boost Algorithm is a Supervised Machine Learning Algorithm. It was found by Tianqi Chen. XG Boost is a boosting type of algorithm in Ensemble Learning. Boosting is a type of Ensemble technique in machine learning where models are built in sequential manner, a model is built and then based on its performance the next model is built. Here every model will be a weak learner to its next model and this is going to stop when we get a model with good performance which will be a strong learner.

XG Boost means Extreme Gradient Boost. It is an advancement of Gradient Boosting algorithm where increased number of hyperparameters are involved which means there is more customization.

XG Boost is based on decision trees. A decision tree is a tree like structure which splits into multiple nodes based on a particular condition taken from the attributes. In XG Boost decision trees are built one after another, in a sequential manner. In XG Boost, the decision trees are only constructed as binary trees, having only 2 child nodes.

<u>WORKING OF XG BOOST ALGORITHM FOR CLASSIFICATION:</u>

1) Initially, a base model is created which is just going to predict the mean of the possible output classes and it is taken as prediction of 1st model.

2) Then the residuals of the model are calculated, which is the difference between the actual value and predicted value.

3) A decision tree is constructed, such that the root node consists of all the residual values of all the instances. A threshold binary condition is taken in order to derive the child nodes of the tree. The child nodes will also be containing the residual values according to the condition.

4) The similarity weight of both the child nodes and parent node is calculated and based on the similarity weights, the Gain of the tree is calculated. The similarity weight of a node can be calculated by,

Similarity Weight = $\Sigma[Res(i)]$^2/ $\Sigma(P(1-P))+ \lambda$.

5) The Gain of the tree determines the priority of the threshold and the tree. The tree having higher gain is considered to be first in the model. Gain can be calculated by,

Gain = Similarity weight(Left node) + Similarity weight(Right node) - Similarity weight(Root node).

6) In this way, we can obtain a certain path for each possibility and also the output for each leaf is calculated. Output of a leaf can be calculated by,

Output of leaf = $\Sigma Res(i)/ \Sigma(P(1-P))+ \lambda$.

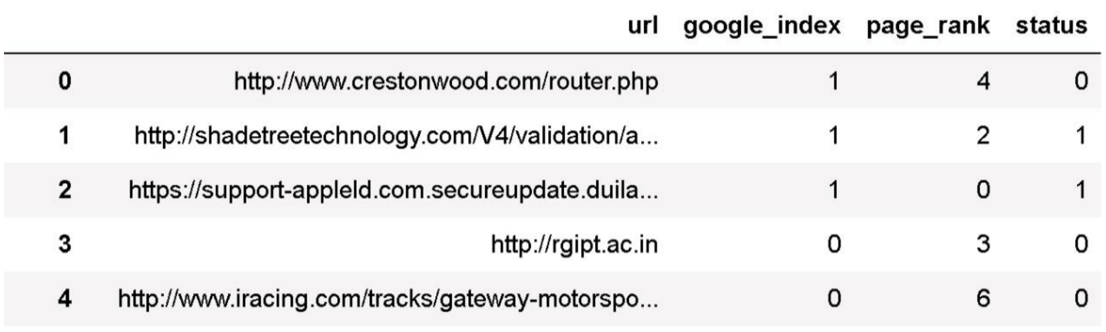
7) By considering the base prediction, learning rate and output of the leaf node, new predictions can be made by,

New prediction = σ[Base prediction output + Learning Rate*Output of the leaf].

8) In this way, a model is made and based on the new predictions we can obtain new residual values. Then we can repeat the process and build as many models as we can and obtain least residual values. The shows the improvement in the performance by model by model.

MATHEMATICAL EXPLAINATION:

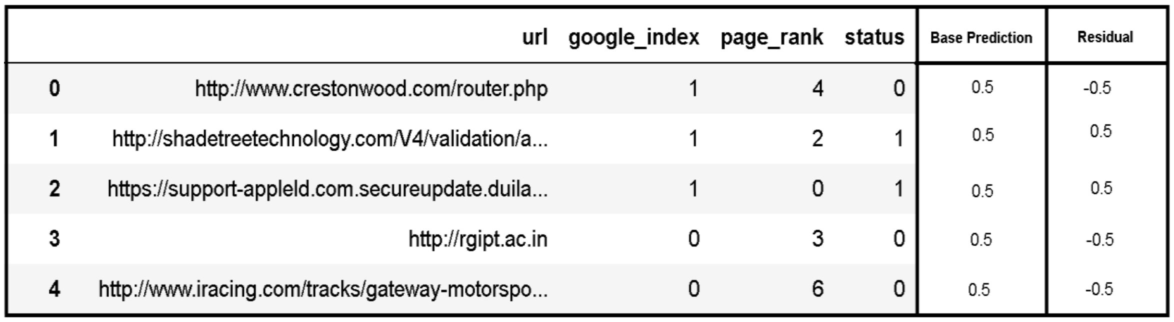
XG Boost is a very complicated algorithm to understand. Taking a sample from the dataset and solving it manually would be a better way to understand.



	url	google_index	page_rank	status
0	http://www.crestonwood.com/router.php	1	4	0
1	http://shadetreetechnology.com/V4/validation/a...	1	2	1
2	https://support-appleld.com.secureupdate.duila...	1	0	1
3	http://rgipt.ac.in	0	3	0
4	http://www.iracing.com/tracks/gateway-motorspo...	0	6	0

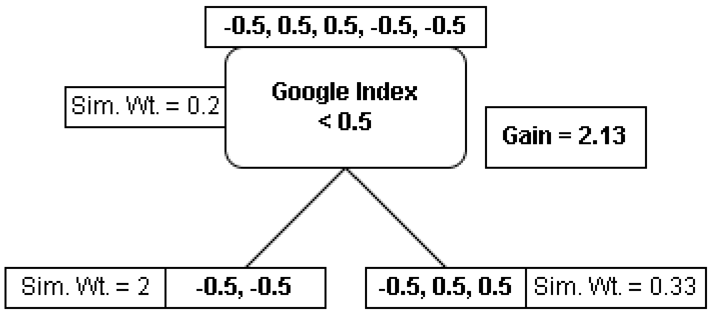
The above sample of data is considered for explaining the mathematical concept behind XG Boost. The sample contains 5 instances of data with 2 features of the URLs and the ‘status’ which is the class label.

The base model is created, with an initial probability of 0.5(default value=0.5, can be changed based on the training data) and the residuals will be calculated as shown below.



	url	google_index	page_rank	status	Base Prediction	Residual
0	http://www.crestonwood.com/router.php	1	4	0	0.5	-0.5
1	http://shadetreetechnology.com/V4/validation/a...	1	2	1	0.5	0.5
2	https://support-appleld.com.secureupdate.duila...	1	0	1	0.5	0.5
3	http://rgipt.ac.in	0	3	0	0.5	-0.5
4	http://www.iracing.com/tracks/gateway-motorspo...	0	6	0	0.5	-0.5

Based on the above table, the first tree is created with a threshold. As the column ‘Google Index’ contains just 2 discrete values, the threshold is common and <0.5 can be considered.



If the column ‘Google Index’ contains values <0.5 then they are kept into the left node else on the right node.

The similarity weight of each node is calculated and show above.

$$\text{Similarity Score} = \frac{\left(\sum \text{Residuals}\right)^2}{\sum^{N}[P\ (1 - P)] + \lambda}$$

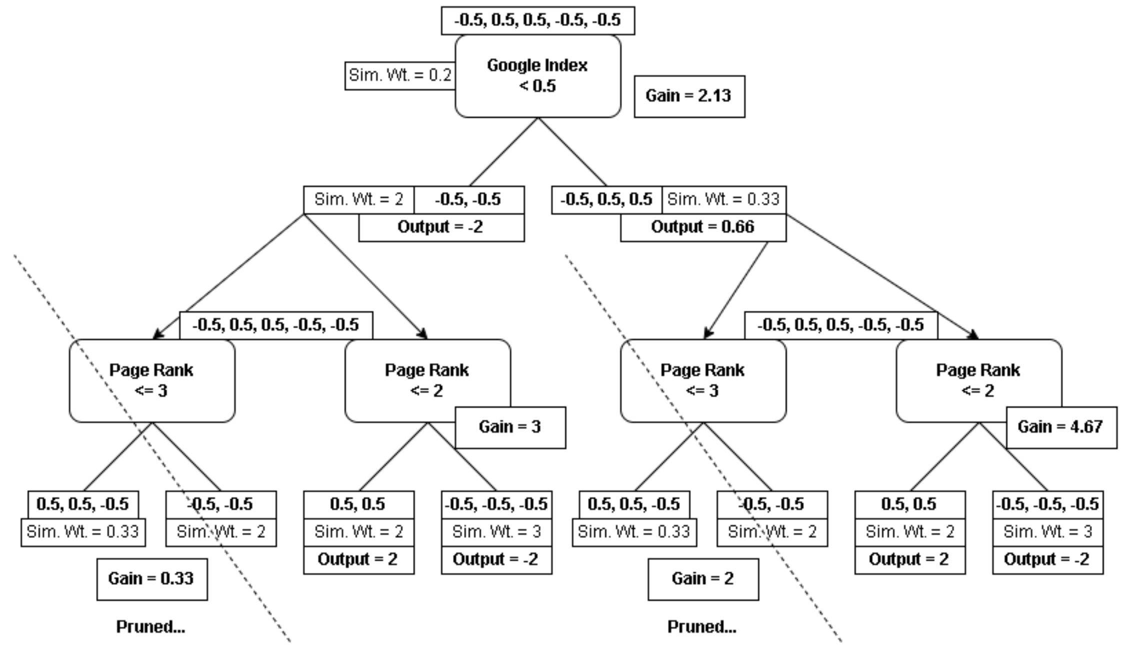
P = Probability

Assume λ=0 for now.

Based on the above similarity scores, the gain value of the tree is calculated.

$$\text{Gain} = SS_{(\text{Left node})} + SS_{(\text{Right node})} - SS_{(\text{Root node})}$$

Now, continuing this tree the next branches are created based of another condition.



In the same way that we have done before, but a bit differently we created trees with different possibilities as we took different thresholds. And then we calculated the Similarity scores and Gain values as shown in the above figure.

Here, we prioritized the trees with higher gain values and pruned the other trees.

Note: In this example, We've taken the trees according to my imagination for the sake of simplicity but in real the XG Boost consider all possibilities and same attribute can be divided into sub-trees for multiple number of times.

The outputs of all the leaf nodes are calculated as shown in the picture.

$$\frac{\left(\sum \text{Residual}_i\right)}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] + \lambda}$$

Now for a new prediction, the output based on the probability is calculated by log(odds),

$$\log(\text{odds}) = \text{logit}(P) = \ln\left(\frac{P}{1-P}\right)$$

Now for a new prediction,

$$\textbf{New prediction} = \sigma[\textbf{Base prediction output} + \alpha * \textbf{Output of the leaf})]$$

Here, the base prediction output is logit(P) = logit(0.5) = 0.

Since, the initial probability is 0.5.

α = Learning rate, assume it to be 0.3 for now.

Output of the leaf is calculated above according to the feature.

After the first iteration the new output would look like,

	url	google_index	page_rank	status	New Prediction	New Residuals
0	http://www.crestonwood.com/router.php	1	4	0	0.35	-0.35
1	http://shadetreetechnology.com/V4/validation/a...	1	2	1	0.65	0.35
2	https://support-appleld.com.secureupdate.duila...	1	0	1	0.65	0.35
3	http://rgipt.ac.in	0	3	0	0.35	-0.35
4	http://www.iracing.com/tracks/gateway-motorspo...	0	6	0	0.35	-0.35

From the above table, we can see the improvement in the results and the residuals have been reduced. In the same way multiple iterations will be done to get precise output.

HYPERPARAMETERS:

Regularization hyperparameter (λ): This parameter is responsible for change in Similarity score of the nodes, which later going to impact in gain and deciding the trees. By default set as 1.

Learning Rate (α): This parameter determines how fast the model is learning from previous model, determines the variation in output. By default set as 0.3.

Gamma (γ): The difference between gain and gamma should be positive, else the would be pruned. By default set to 0.

XG Boost is known for its speed and performance. Many Machine Learning engineers love to use this algorithm in first place. As, we are dealing with large amount of data and various features. We thought this would be a suitable approach for the current problem.

RNN ALGORITHM

Recurrent Neural Networks (RNN) are a type of Neural Network where the output from previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. It is a deep learning technique that was originally evolved from feed forward neural network. In feed forward neural network the nodes do not form any circular connections. This reduces the chances of any improvement. But, in RNN this problem is evicted as there is a feedback loop and nodes have circular connections. Hence there is a backpropagation due to the feedback loop.

Here, the data is given as an input in a sequential input with the time constraint that marks the relation of the present input by using already stored previously input sequences to find the target value.

What is a neural network?

A Neural network as the name suggests is the process of artificially mimicing the working of the human neural network to process the information that is fed in during the time of decision making.

Neural network is consists of many nodes similar to neurons.

Basically, there are three node layers:

Input layers - It takes in the input by assigning each input-feature that is responsible for characterization, a node of its own.

Here the input a is given as a(t) a constraint of time.

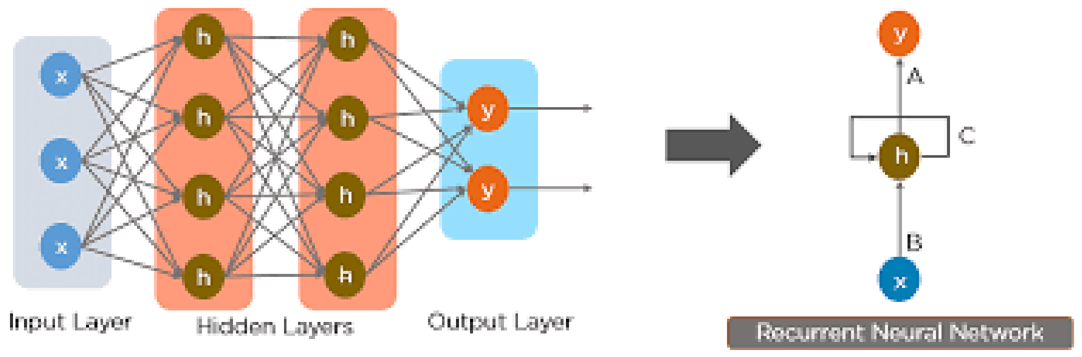
Hidden layer – Here, the main decision making process happens through multiple layers of nodes paired with weights along with bias which forms a non-linear function.

This is passed through an activation function. Here, it is decided whether it will be passed on to other layers that is whether to activate a neuron or not.

$$s(t) = g(U\, a(t) + W\, s(t-1)) \quad \text{(Summation function)}$$

Here, it can be seen that the previous hidden states are to be considered. Passing it through activation function such as ReLU, Sigmoid etc. So as to get the output in the binary form where the output 0 would mean not active while 1 would mean active.

Output layers -o(t) it is the final output for the hidden layer during time constraint t.



Why this algorithm?

Every feature input node has memory that stores previous input values and predicts the possibility of next input based on all the previous inputs based on the weight and bias associated to each of them. They have feedback loops that improves the efficiency with time.

Thus, for the process of finding the phished links we can consider this algorithm. As the speciality of the algorithm lies in prediction. It is dependent on time constraint for prediction.

<u>WORKING OF RNN FOR CLASSIFICATION:</u>

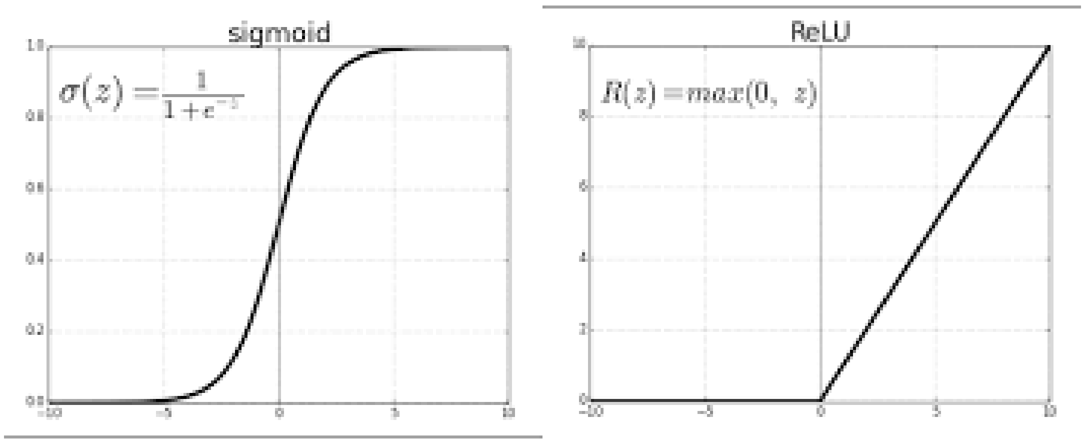
In this project we have considered a dataset with 22 features that decides the possibilty of the website being Phished.

The field "Status" determines ultimately as to whether its phished or not.

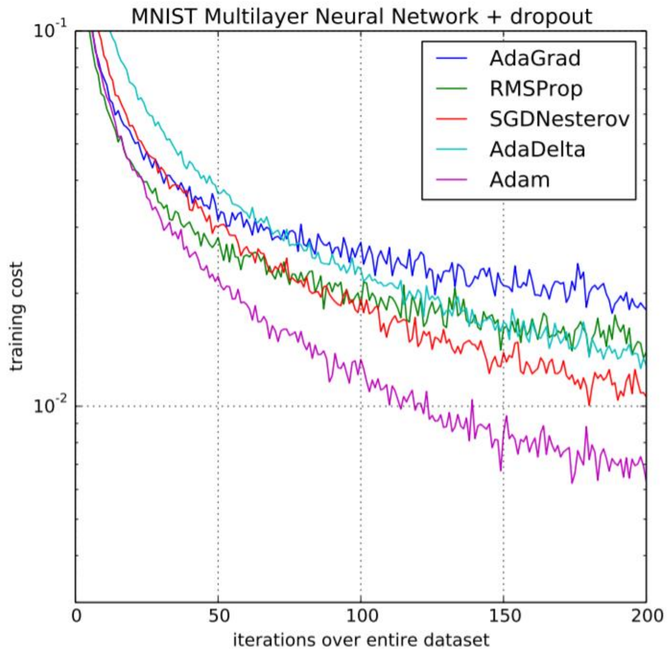
We are working with Pandas, Numpy and sklearn in this project.

In order to implement RNN MLPClassifier is imported from sklearn library.

Here ReLU activation function is used as it relatively better than other activation functions because it reduces the number of vanishing gradient problem.



The Solver we use here is "ADAMS". It is combination of gradient descent with momentum algorithm and the RMSP algorithm.



It requires less memory and can work with large dataset.

6. OVERVIEW OF TECHNOLOGIES

WEB SCRAPING AND FEATURE EXTRACTION

Feature extraction is the initial stage in the system. An URL is a string like object which represents a location of resources that can be accessed through internet. The outer features such as length of the URL, presence of certain characters, etc. can be directly extracted from the URL string. There are certain features that are needed to be extracted internally and externally. Certain features like presence of hyperlinks, disabling of right click, etc. can be extracted by crawling through the source code. Also, certain domain based features like page rank, WHOIS records, etc. can be extracted by requesting the APIs which are an open source.

The URL will be having internal and external features. The features are classified into three types, they are:

1) Address bar based Features: These features are simply extracted from the URL string based on the differentiation between phishing and benign URLs.

2) Content based Features: These features are extracted by crawling through the source code of the website.

3) Domain based Features: There are various APIs which provide the domain information of websites.

To obtain the Content based features and Domain based features, there is a technique called Web Scraping. Web scraping is an automated technique to obtain huge amount of data from websites. The data can be obtained by crawling though the source code, the information that can be found in HTML and JavaScript. There are other ways to obtain the domain based information by making requests to various APIs. Many companies like Google, are providing free open-APIs where the huge amount of information about websites is stored. There are also various trusted third-parties like WHOIS, OPR which can provide domain based information of websites.

PYTHON MODULES USED FOR WEB SCRAPING:

1) Beautiful Soup (*bs4*): This module is used for extracting content based features from the URL. It is used for parsing HTML and XML documents. The page content and source code can be extracted.

2) Requests (*requests*): The information from the API can be obtained by making HTTP requests to the API. This module helps in making HTTP requests to the APIs and receive data in the form of response. The domain based features can be extracted using this module.

3) WHOIS (*whois*): WHOIS is a database which stores all the information about a domain like owner, creation time, expiration time, etc. All the information can be accessed by using this module in python.

MACHINE LEARNING AND CLASSIFICATION

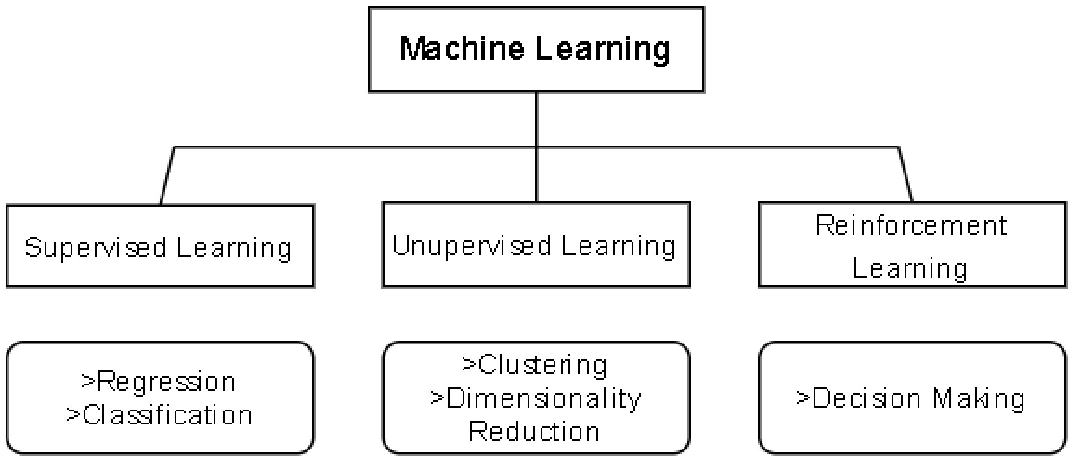
Machine Learning is an application of AI where the systems will learn and improve for past experiences without explicit programming.

Machine Learning is mainly classified into three types, they are:

1) Supervised Learning

2) Unsupervised Learning

3) Reinforcement Learning



In Supervised Learning, as the name says there is supervision of variables. The independent variable (if one) or the independent variables (if more than one) will be determining the dependent variable which makes the prediction. Regression and Classification are two types of Supervised Learning algorithms where the Regression is used to predict the dependent variables in values whereas the Classification is used to predict to which class a particular instance belong to out of the existing classes.

The general steps followed while building Supervised Learning models are:

1) Preparing the data

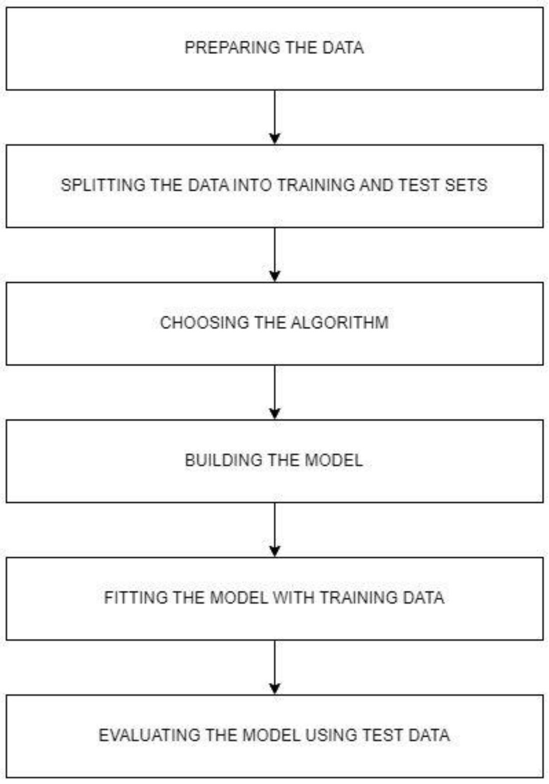
2) Splitting the data into training and test sets

3) Choosing the algorithm

4) Building the model

5) Fitting the model with training data

6) Evaluating the model using test data


PREPARING THE DATA
SPLITTING THE DATA INTO TRAINING AND TEST SETS
CHOOSING THE ALGORITHM
BUILDING THE MODEL
FITTING THE MODEL WITH TRAINING DATA
EVALUATING THE MODEL USING TEST DATA

Classification from Supervised Learning is the proposed technique for the current problem. There will be two classes, Phishing and Benign and we will be further implementing the Classifiers to classify an URL either Phishing or Benign based on its features.

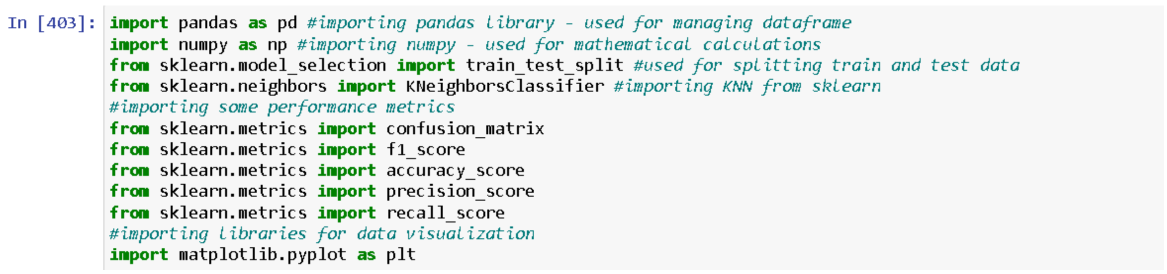
PYTHON LIBRARIES USED FOR MACHINE LEARNING:

1) Scikit-learn(*sklearn*): ‘sklearn’ is a python library which provides a lot of required tools for machine learning and statistics. Many algorithms and methods were defined in this library, which can just be imported and used.
2) Pandas(*pandas*): ‘pandas’ is a python library which is used to manage the data. This is mostly used for data analysis.
3) Seaborn(*seaborn*) and Matplotlib(*matplotlib*): These are python libraries used for data-visualization. Data visualization helps in better understanding of the data.

7. IMPLEMENTATION

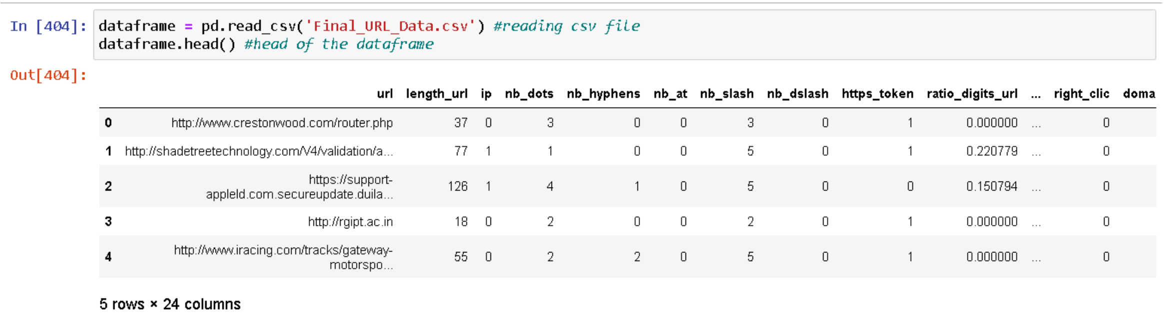
PYTHON IMPLEMENTATION OF CLASSIFIER USING KNN ALGORITHM

Step-1: Importing required libraries.



```
In [403]: import pandas as pd #importing pandas library - used for managing dataframe
          import numpy as np #importing numpy - used for mathematical calculations
          from sklearn.model_selection import train_test_split #used for splitting train and test data
          from sklearn.neighbors import KNeighborsClassifier #importing KNN from sklearn
          #importing some performance metrics
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import f1_score
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import precision_score
          from sklearn.metrics import recall_score
          #importing libraries for data visualization
          import matplotlib.pyplot as plt
```

Step-2: Importing the dataset, the .csv file which contains the URL data.

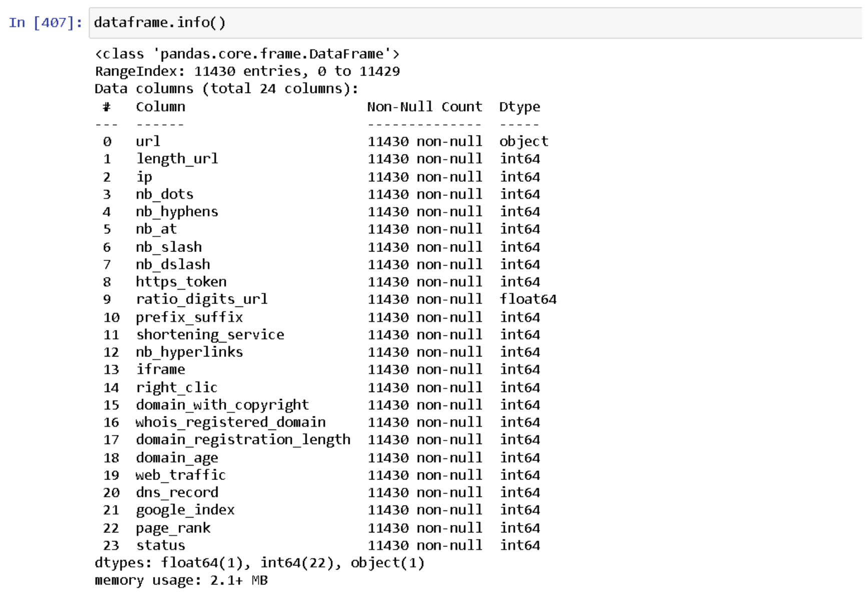


```
In [404]: dataframe = pd.read_csv('Final_URL_Data.csv') #reading csv file
          dataframe.head() #head of the dataframe

Out[404]:
```

	url	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	...	right_clic	doma
0	http://www.crestonwood.com/router.php	37	0	3	0	0	3	0	1	0.000000	...	0	
1	http://shadetreetechnology.com/V4/validation/a...	77	1	1	0	0	5	0	1	0.220779	...	0	
2	https://support-appleld.com.secureupdate.duila...	126	1	4	1	0	5	0	0	0.150794	...	0	
3	http://rgipt.ac.in	18	0	2	0	0	2	0	1	0.000000	...	0	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	0	2	2	0	5	0	1	0.000000	...	0	

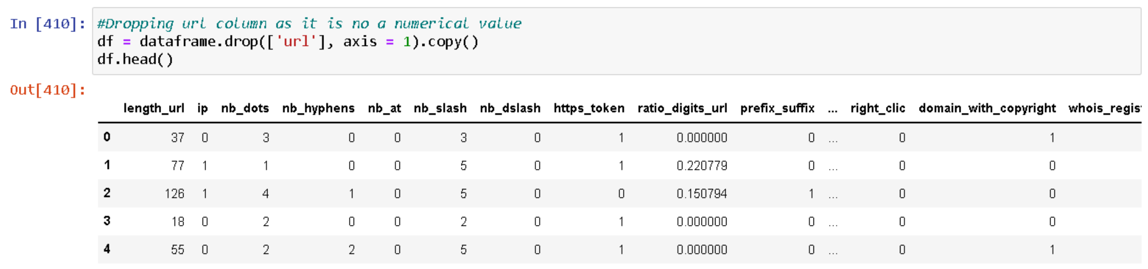
5 rows × 24 columns

Step-3: Getting familiar with the data.



```
In [407]: dataframe.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11430 entries, 0 to 11429
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   url                         11430 non-null  object
 1   length_url                  11430 non-null  int64
 2   ip                          11430 non-null  int64
 3   nb_dots                     11430 non-null  int64
 4   nb_hyphens                  11430 non-null  int64
 5   nb_at                       11430 non-null  int64
 6   nb_slash                    11430 non-null  int64
 7   nb_dslash                   11430 non-null  int64
 8   https_token                 11430 non-null  int64
 9   ratio_digits_url            11430 non-null  float64
 10  prefix_suffix               11430 non-null  int64
 11  shortening_service          11430 non-null  int64
 12  nb_hyperlinks               11430 non-null  int64
 13  iframe                      11430 non-null  int64
 14  right_clic                  11430 non-null  int64
 15  domain_with_copyright       11430 non-null  int64
 16  whois_registered_domain     11430 non-null  int64
 17  domain_registration_length  11430 non-null  int64
 18  domain_age                  11430 non-null  int64
 19  web_traffic                 11430 non-null  int64
 20  dns_record                  11430 non-null  int64
 21  google_index                11430 non-null  int64
 22  page_rank                   11430 non-null  int64
 23  status                      11430 non-null  int64
dtypes: float64(1), int64(22), object(1)
memory usage: 2.1+ MB
```

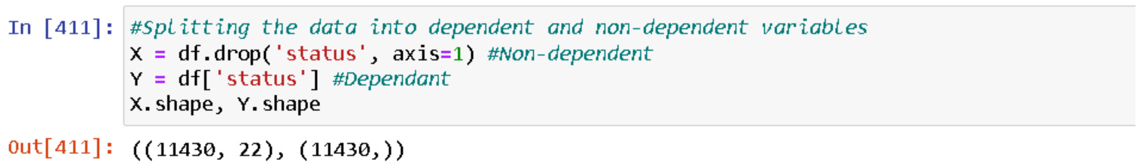
Step-4: Dropping the ‘url’ column as we don’t need it in the algorithm.



```
In [410]: #Dropping url column as it is no a numerical value
          df = dataframe.drop(['url'], axis = 1).copy()
          df.head()
Out[410]:
```

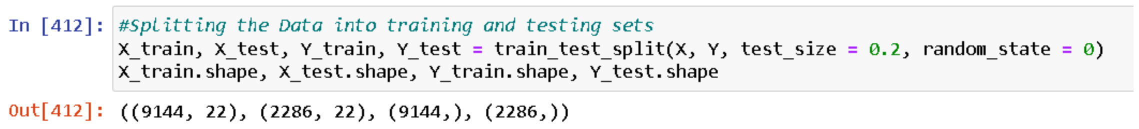
	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	right_clic	domain_with_copyright	whois_regis
0	37	0	3	0	0	3	0	1	0.000000	0	...	0	1	
1	77	1	1	0	0	5	0	1	0.220779	0	...	0	0	
2	126	1	4	1	0	5	0	0	0.150794	1	...	0	0	
3	18	0	2	0	0	2	0	1	0.000000	0	...	0	0	
4	55	0	2	2	0	5	0	1	0.000000	0	...	0	1	

Step-5: Now, we need to split the data into dependent and independent variables.



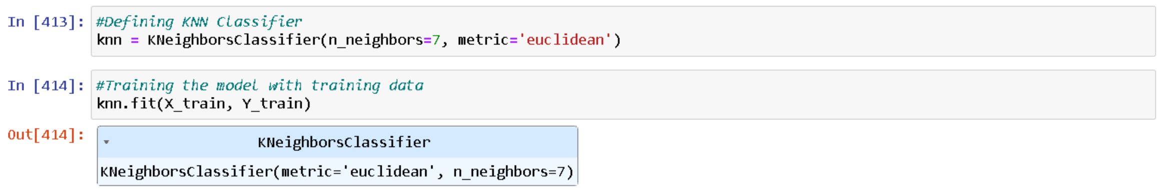
```
In [411]: #Splitting the data into dependent and non-dependent variables
          X = df.drop('status', axis=1) #Non-dependent
          Y = df['status'] #Dependant
          X.shape, Y.shape

Out[411]: ((11430, 22), (11430,))
```

Step-6: Splitting the data into training and testing sets.



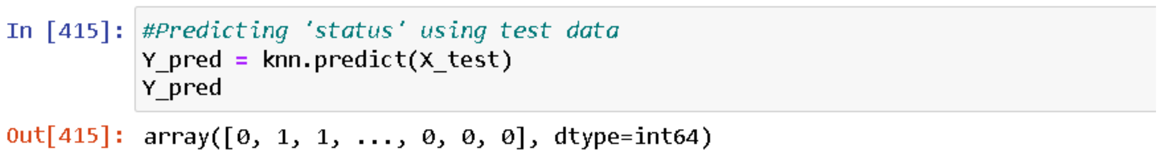
```
In [412]: #Splitting the Data into training and testing sets
          X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
          X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

Out[412]: ((9144, 22), (2286, 22), (9144,), (2286,))
```

Step-7: Building the classifier and fitting it with the training set of data.



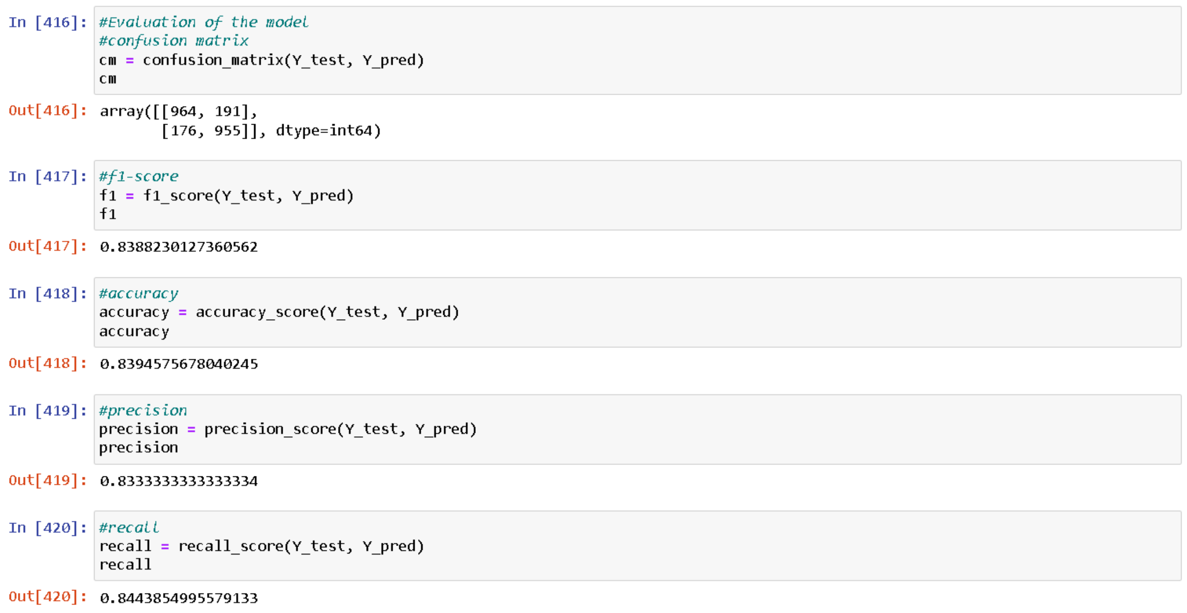
```
In [413]: #Defining KNN Classifier
          knn = KNeighborsClassifier(n_neighbors=7, metric='euclidean')

In [414]: #Training the model with training data
          knn.fit(X_train, Y_train)

Out[414]:
          ▾ KNeighborsClassifier
          KNeighborsClassifier(metric='euclidean', n_neighbors=7)
```

Step-8: The model is ready to make predictions. The test set of data is given into model to make predictions.



```
In [415]: #Predicting 'status' using test data
          Y_pred = knn.predict(X_test)
          Y_pred

Out[415]: array([0, 1, 1, ..., 0, 0, 0], dtype=int64)
```

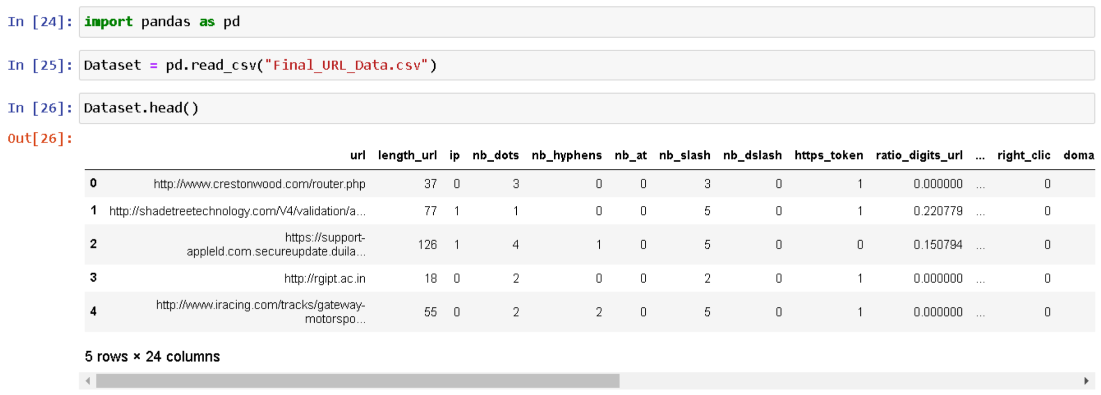
Step-9: To evaluate the performance of the model based on the prediction with respect to the test data, the confusion matrix, f1-score, accuracy, precision and recall are calculated.



```
In [416]: #Evaluation of the model
          #confusion matrix
          cm = confusion_matrix(Y_test, Y_pred)
          cm

Out[416]: array([[964, 191],
                 [176, 955]], dtype=int64)

In [417]: #f1-score
          f1 = f1_score(Y_test, Y_pred)
          f1

Out[417]: 0.8388230127360562

In [418]: #accuracy
          accuracy = accuracy_score(Y_test, Y_pred)
          accuracy

Out[418]: 0.8394575678040245

In [419]: #precision
          precision = precision_score(Y_test, Y_pred)
          precision

Out[419]: 0.8333333333333334

In [420]: #recall
          recall = recall_score(Y_test, Y_pred)
          recall

Out[420]: 0.8443854995579133
```

PYTHON IMPLEMENTATION OF CLASSIFIER USING RANDOM FOREST ALGORITHM

Step-1: Import the dataset.

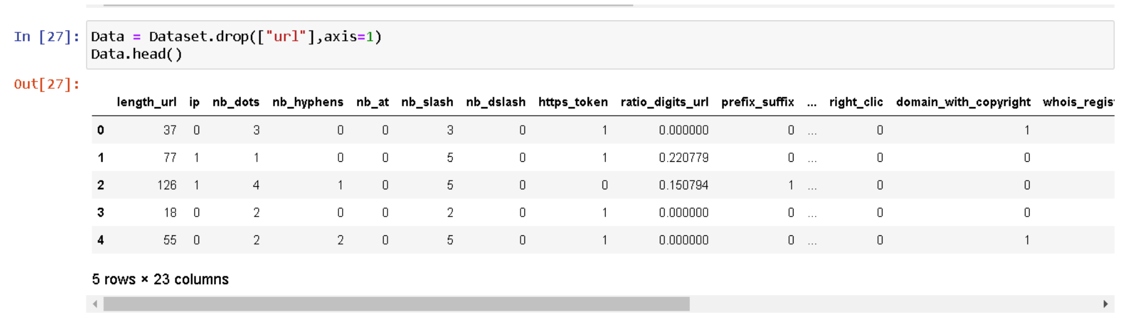


```
In [24]: import pandas as pd

In [25]: Dataset = pd.read_csv("Final_URL_Data.csv")

In [26]: Dataset.head()

Out[26]:
```

	url	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	...	right_clic	doma
0	http://www.crestonwood.com/router.php	37	0	3	0	0	3	0	1	0.000000	...	0	
1	http://shadetreetechnology.com/V4/validation/a...	77	1	1	0	0	5	0	1	0.220779	...	0	
2	https://support-appleld.com.secureupdate.duila...	126	1	4	1	0	5	0	0	0.150794	...	0	
3	http://rgipt.ac.in	18	0	2	0	0	2	0	1	0.000000	...	0	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	0	2	2	0	5	0	1	0.000000	...	0	

5 rows × 24 columns

Step-2: Dropping the ‘url’ column.

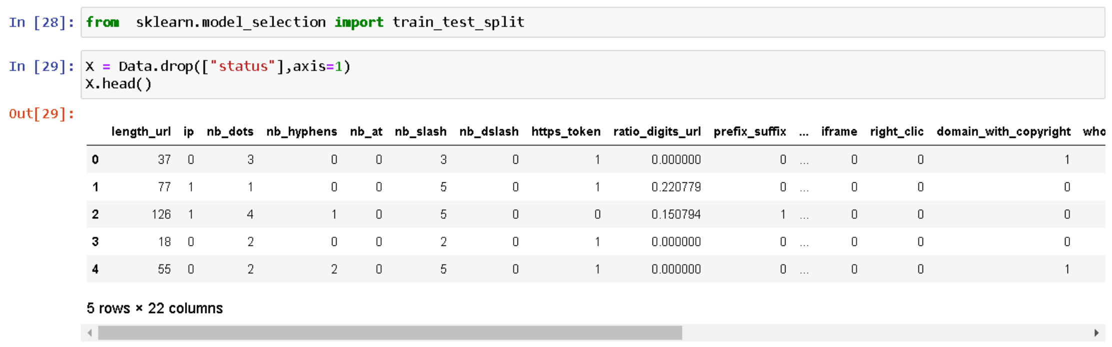


```
In [27]: Data = Dataset.drop(["url"],axis=1)
         Data.head()
Out[27]:
```

	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	right_clic	domain_with_copyright	whois_regis
0	37	0	3	0	0	3	0	1	0.000000	0	...	0	1	
1	77	1	1	0	0	5	0	1	0.220779	0	...	0	0	
2	126	1	4	1	0	5	0	0	0.150794	1	...	0	0	
3	18	0	2	0	0	2	0	1	0.000000	0	...	0	0	
4	55	0	2	2	0	5	0	1	0.000000	0	...	0	1	

5 rows × 23 columns

Step-3: Defining the independent variable.

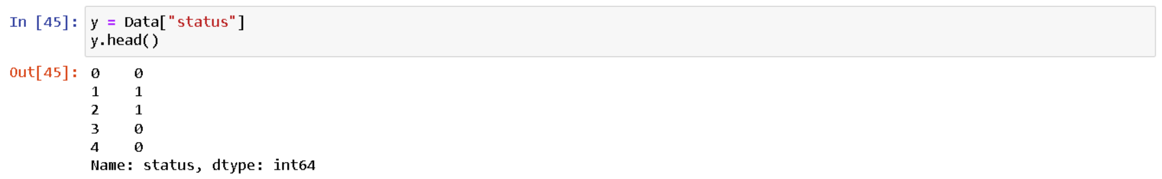


```
In [28]: from  sklearn.model_selection import train_test_split

In [29]: X = Data.drop(["status"],axis=1)
         X.head()
Out[29]:
```

	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	iframe	right_clic	domain_with_copyright	who
0	37	0	3	0	0	3	0	1	0.000000	0	...	0	0	1	
1	77	1	1	0	0	5	0	1	0.220779	0	...	0	0	0	
2	126	1	4	1	0	5	0	0	0.150794	1	...	0	0	0	
3	18	0	2	0	0	2	0	1	0.000000	0	...	0	0	0	
4	55	0	2	2	0	5	0	1	0.000000	0	...	0	0	1	

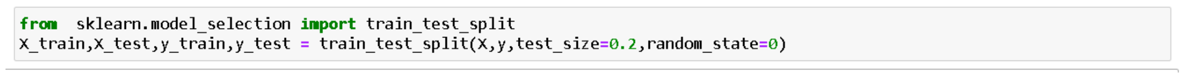
5 rows × 22 columns

Step-4: Defining the dependent variable.



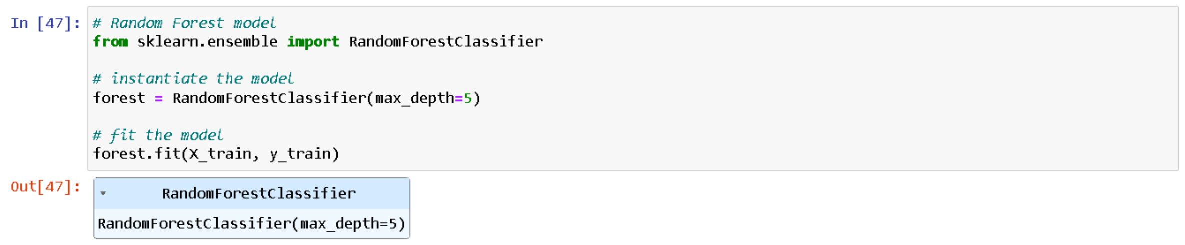
```
In [45]: y = Data["status"]
         y.head()
Out[45]: 0    0
         1    1
         2    1
         3    0
         4    0
         Name: status, dtype: int64
```

Step-5: Splitting both the dependent and independent variables into training and test sets.



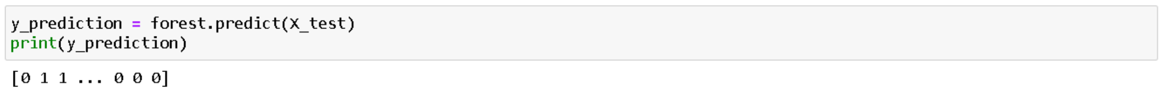
```
from  sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

Step-6: Initializing the model and fitting with training data.



```
In [47]: # Random Forest model
         from sklearn.ensemble import RandomForestClassifier

         # instantiate the model
         forest = RandomForestClassifier(max_depth=5)

         # fit the model
         forest.fit(X_train, y_train)
Out[47]: RandomForestClassifier
         RandomForestClassifier(max_depth=5)
```

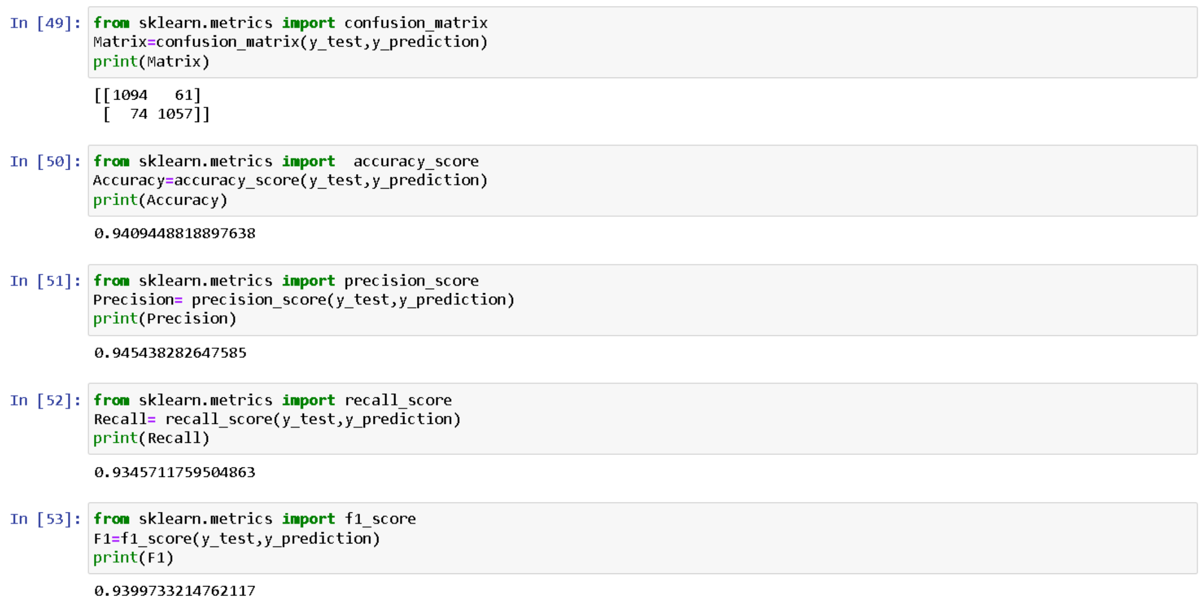
Step-6: Making the prediction.



```
y_prediction = forest.predict(X_test)
print(y_prediction)
```

```
[0 1 1 ... 0 0 0]
```

Step-7: Evaluating the model.



```
In [49]: from sklearn.metrics import confusion_matrix
         Matrix=confusion_matrix(y_test,y_prediction)
         print(Matrix)
```

```
[[1094   61]
 [  74 1057]]
```

```
In [50]: from sklearn.metrics import  accuracy_score
         Accuracy=accuracy_score(y_test,y_prediction)
         print(Accuracy)
```

```
0.9409448818897638
```

```
In [51]: from sklearn.metrics import precision_score
         Precision= precision_score(y_test,y_prediction)
         print(Precision)
```

```
0.945438282647585
```

```
In [52]: from sklearn.metrics import recall_score
         Recall= recall_score(y_test,y_prediction)
         print(Recall)
```

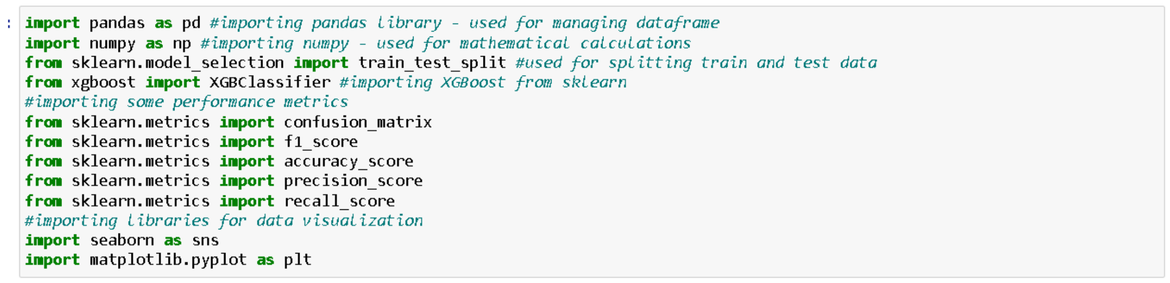
```
0.9345711759504863
```

```
In [53]: from sklearn.metrics import f1_score
         F1=f1_score(y_test,y_prediction)
         print(F1)
```

```
0.9399733214762117
```

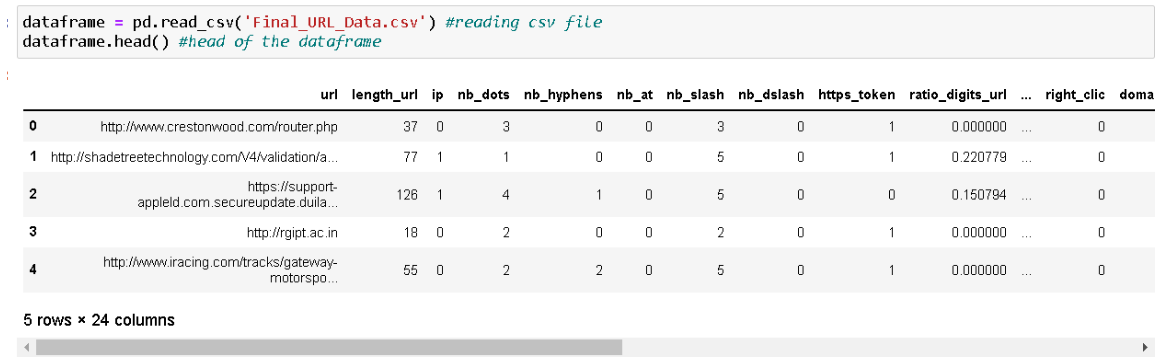
PYTHON IMPLEMENTATION OF CLASSIFIER USING XG BOOST ALGORITHM

Step-1: Importing the required libraries.



```
: import pandas as pd #importing pandas library - used for managing dataframe
  import numpy as np #importing numpy - used for mathematical calculations
  from sklearn.model_selection import train_test_split #used for splitting train and test data
  from xgboost import XGBClassifier #importing XGBoost from sklearn
  #importing some performance metrics
  from sklearn.metrics import confusion_matrix
  from sklearn.metrics import f1_score
  from sklearn.metrics import accuracy_score
  from sklearn.metrics import precision_score
  from sklearn.metrics import recall_score
  #importing libraries for data visualization
  import seaborn as sns
  import matplotlib.pyplot as plt
```

Step-2: Importing the dataset.

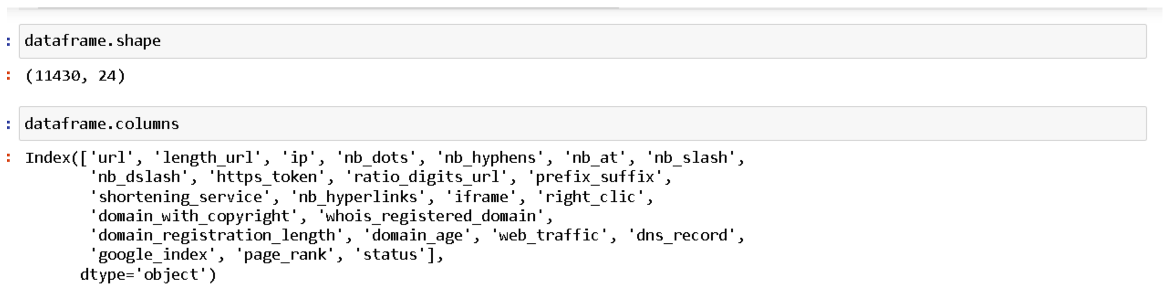


```
dataframe = pd.read_csv('Final_URL_Data.csv') #reading csv file
dataframe.head() #head of the dataframe
```

	url	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	...	right_clic	doma
0	http://www.crestonwood.com/router.php	37	0	3	0	0	3	0	1	0.000000	...	0	
1	http://shadetreetechnology.com/V4/validation/a...	77	1	1	0	0	5	0	1	0.220779	...	0	
2	https://support-appleld.com.secureupdate.duila...	126	1	4	1	0	5	0	0	0.150794	...	0	
3	http://rgipt.ac.in	18	0	2	0	0	2	0	1	0.000000	...	0	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	0	2	2	0	5	0	1	0.000000	...	0	

5 rows × 24 columns

Step-3: Getting familiar with the data.

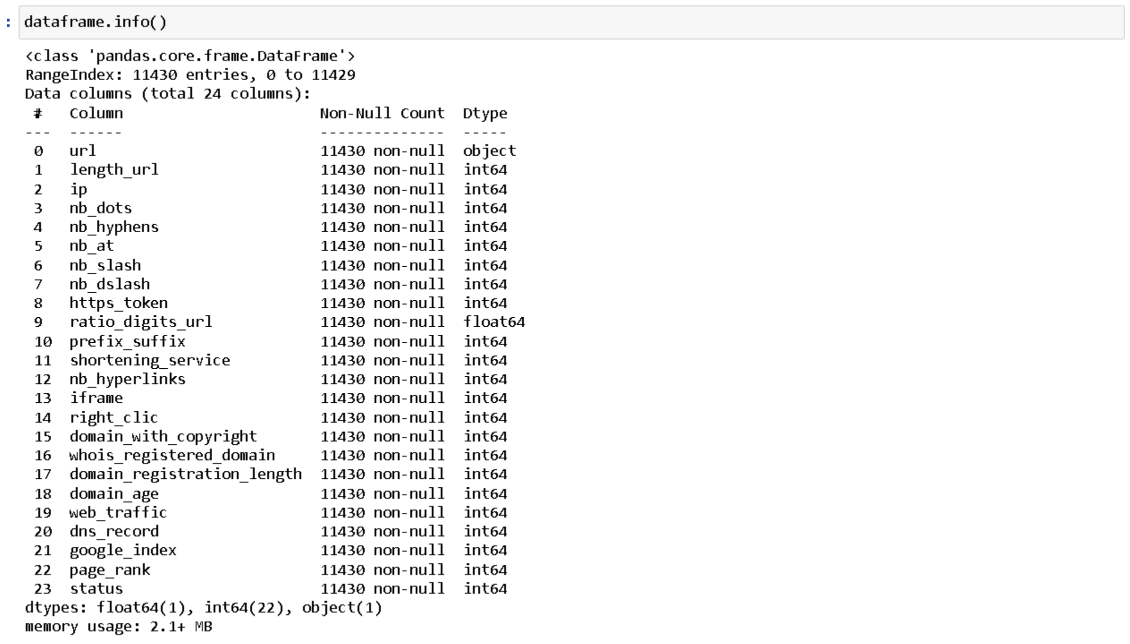


```
dataframe.shape
```

```
(11430, 24)
```

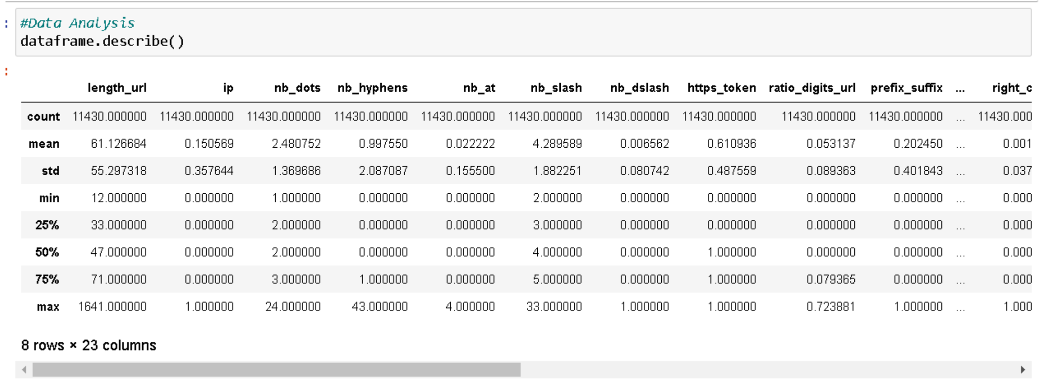
```
dataframe.columns
```

```
Index(['url', 'length_url', 'ip', 'nb_dots', 'nb_hyphens', 'nb_at', 'nb_slash',
       'nb_dslash', 'https_token', 'ratio_digits_url', 'prefix_suffix',
       'shortening_service', 'nb_hyperlinks', 'iframe', 'right_clic',
       'domain_with_copyright', 'whois_registered_domain',
       'domain_registration_length', 'domain_age', 'web_traffic', 'dns_record',
       'google_index', 'page_rank', 'status'],
      dtype='object')
```



```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11430 entries, 0 to 11429
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   url                         11430 non-null  object
 1   length_url                  11430 non-null  int64
 2   ip                          11430 non-null  int64
 3   nb_dots                     11430 non-null  int64
 4   nb_hyphens                  11430 non-null  int64
 5   nb_at                       11430 non-null  int64
 6   nb_slash                    11430 non-null  int64
 7   nb_dslash                   11430 non-null  int64
 8   https_token                 11430 non-null  int64
 9   ratio_digits_url            11430 non-null  float64
 10  prefix_suffix               11430 non-null  int64
 11  shortening_service          11430 non-null  int64
 12  nb_hyperlinks               11430 non-null  int64
 13  iframe                      11430 non-null  int64
 14  right_clic                  11430 non-null  int64
 15  domain_with_copyright       11430 non-null  int64
 16  whois_registered_domain     11430 non-null  int64
 17  domain_registration_length  11430 non-null  int64
 18  domain_age                  11430 non-null  int64
 19  web_traffic                 11430 non-null  int64
 20  dns_record                  11430 non-null  int64
 21  google_index                11430 non-null  int64
 22  page_rank                   11430 non-null  int64
 23  status                      11430 non-null  int64
dtypes: float64(1), int64(22), object(1)
memory usage: 2.1+ MB
```

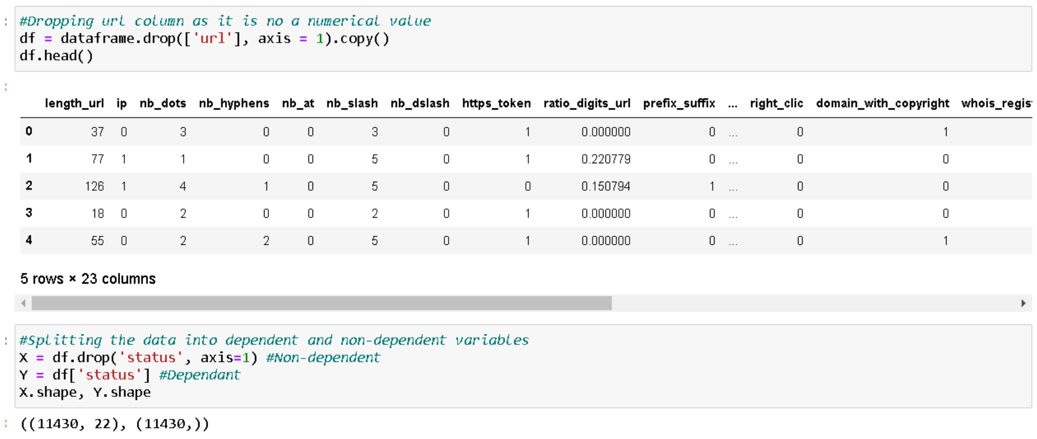


```
#Data Analysis
dataframe.describe()
```

	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	right_c
count	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	11430.000000	...	11430.000
mean	61.126684	0.150569	2.480752	0.997550	0.022222	4.289589	0.006562	0.610936	0.053137	0.202450	...	0.001
std	55.297318	0.357644	1.369686	2.087087	0.155500	1.882251	0.080742	0.487559	0.089363	0.401843	...	0.037
min	12.000000	0.000000	1.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	...	0.000
25%	33.000000	0.000000	2.000000	0.000000	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000	...	0.000
50%	47.000000	0.000000	2.000000	0.000000	0.000000	4.000000	0.000000	1.000000	0.000000	0.000000	...	0.000
75%	71.000000	0.000000	3.000000	1.000000	0.000000	5.000000	0.000000	1.000000	0.079365	0.000000	...	0.000
max	1641.000000	1.000000	24.000000	43.000000	4.000000	33.000000	1.000000	1.000000	0.723881	1.000000	...	1.000

8 rows × 23 columns

Step-4: Dropping the URL column as we don't need it in the training process and splitting the dataset into Dependent and Independent Variables.



```
#Dropping url column as it is no a numerical value
df = dataframe.drop(['url'], axis = 1).copy()
df.head()
```

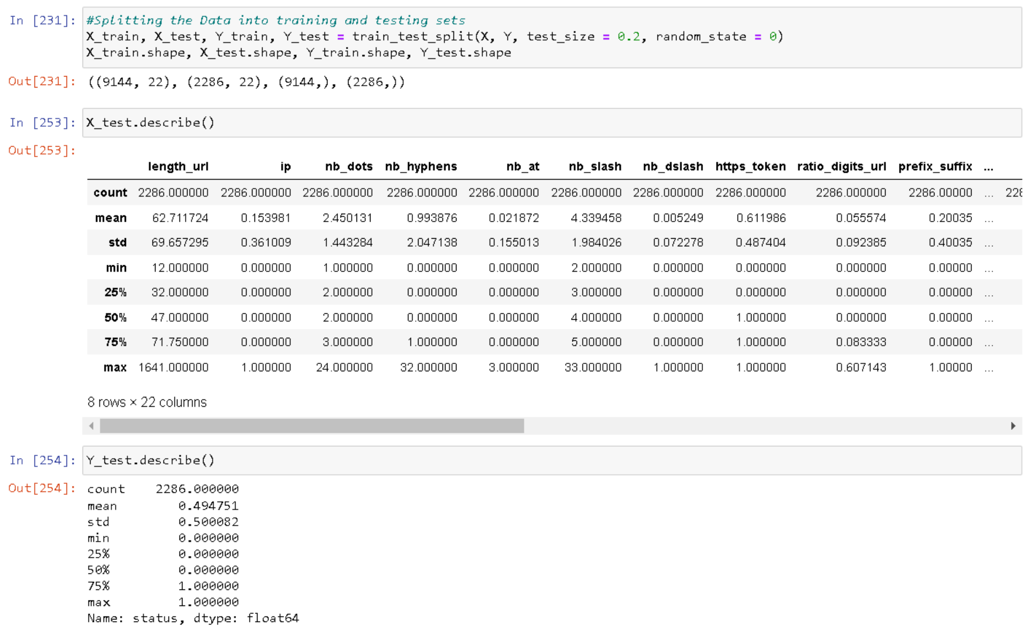
	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	right_clic	domain_with_copyright	whois_regis
0	37	0	3	0	0	3	0	1	0.000000	0	...	0	1	
1	77	1	1	0	0	5	0	1	0.220779	0	...	0	0	
2	126	1	4	1	0	5	0	0	0.150794	1	...	0	0	
3	18	0	2	0	0	2	0	1	0.000000	0	...	0	0	
4	55	0	2	2	0	5	0	1	0.000000	0	...	0	1	

5 rows × 23 columns

```
#Splitting the data into dependent and non-dependent variables
X = df.drop('status', axis=1) #Non-dependent
Y = df['status'] #Dependant
X.shape, Y.shape
```

```
((11430, 22), (11430,))
```

Step-5: Splitting both the dependent and independent variables into training and test sets. Also, checking how the split happened.



```
In [231]: #Splitting the Data into training and testing sets
          X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
          X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

Out[231]: ((9144, 22), (2286, 22), (9144,), (2286,))

In [253]: X_test.describe()

Out[253]:
```

	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	prefix_suffix	...	
count	2286.000000	2286.000000	2286.000000	2286.000000	2286.000000	2286.000000	2286.000000	2286.000000	2286.000000	2286.00000	...	228
mean	62.711724	0.153981	2.450131	0.993876	0.021872	4.339458	0.005249	0.611986	0.055574	0.20035	...	
std	69.657295	0.361009	1.443284	2.047138	0.155013	1.984026	0.072278	0.487404	0.092385	0.40035	...	
min	12.000000	0.000000	1.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.00000	...	
25%	32.000000	0.000000	2.000000	0.000000	0.000000	3.000000	0.000000	0.000000	0.000000	0.00000	...	
50%	47.000000	0.000000	2.000000	0.000000	0.000000	4.000000	0.000000	1.000000	0.000000	0.00000	...	
75%	71.750000	0.000000	3.000000	1.000000	0.000000	5.000000	0.000000	1.000000	0.083333	0.00000	...	
max	1641.000000	1.000000	24.000000	32.000000	3.000000	33.000000	1.000000	1.000000	0.607143	1.00000	...	

8 rows × 22 columns

```
In [254]: Y_test.describe()

Out[254]: count    2286.000000
          mean        0.494751
          std         0.500082
          min         0.000000
          25%         0.000000
          50%         0.000000
          75%         1.000000
          max         1.000000
          Name: status, dtype: float64
```

Step-6: Initializing the model and fitting it with the training set.

```
In [232]: #initializing the model
          xgboost = XGBClassifier(learning_rate=0.3,max_depth=7)

In [233]: #fitting the model with training data
          xgboost.fit(X_train, Y_train)

Out[233]:                    XGBClassifier
          XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                        colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                        early_stopping_rounds=None, enable_categorical=False,
                        eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
                        importance_type=None, interaction_constraints='',
                        learning_rate=0.3, max_bin=256, max_cat_to_onehot=4,
                        max_delta_step=0, max_depth=7, max_leaves=0, min_child_weight=1,
                        missing=nan, monotone_constraints='()', n_estimators=100,
                        n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
                        reg_alpha=0, reg_lambda=1, ...)
```

Step-7: Making prediction using test set.

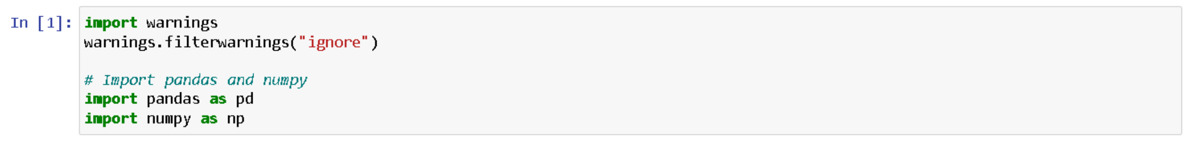
```
In [234]: Y_pred = xgboost.predict(X_test)
          Y_pred

Out[234]: array([0, 1, 1, ..., 0, 1, 0])
```

Step-8: Evaluating the performance of the model comparing the predictions and test set.

```
In [235]: #Evaluation of the model
          #confusion matrix
          cm = confusion_matrix(Y_test, Y_pred)
          cm

Out[235]: array([[1115,   40],
                 [  37, 1094]], dtype=int64)

In [236]: #f1-score
          f1 = f1_score(Y_test, Y_pred)
          f1

Out[236]: 0.9660044150110375

In [237]: #accuracy
          accuracy = accuracy_score(Y_test, Y_pred)
          accuracy

Out[237]: 0.9663167104111986

In [238]: #precision
          precision = precision_score(Y_test, Y_pred)
          precision

Out[238]: 0.9647266313932981

In [239]: #recall
          recall = recall_score(Y_test, Y_pred)
          recall

Out[239]: 0.9672855879752431
```

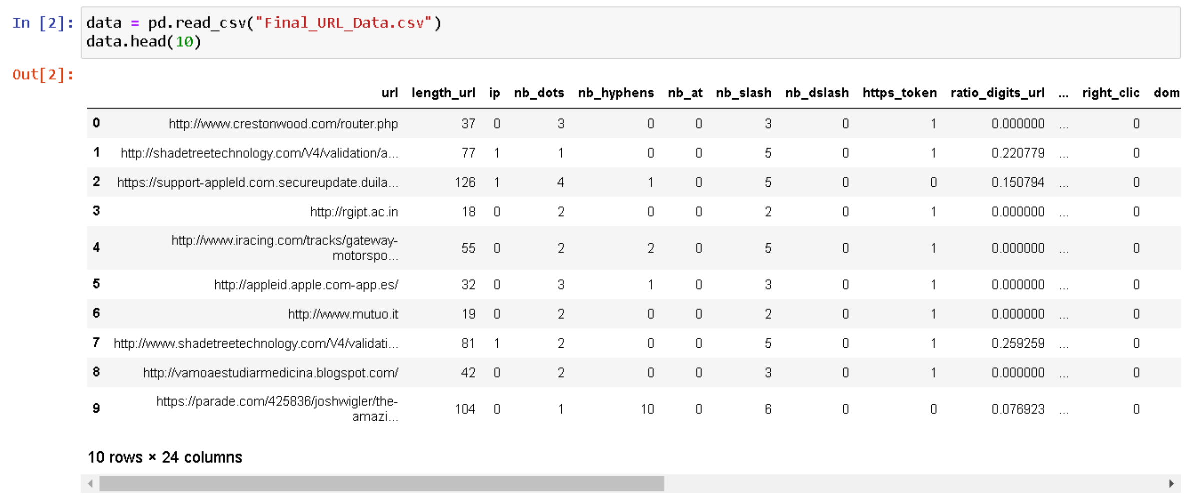
PYTHON IMPLEMENTATION OF CLASSIFIER USING RNN ALGORITHM

STEP 1: Pandas and Numpy being imported.



```
In [1]: import warnings
        warnings.filterwarnings("ignore")

        # Import pandas and numpy
        import pandas as pd
        import numpy as np
```

STEP 2: We are importing the dataset and assigning it to the dataframe.



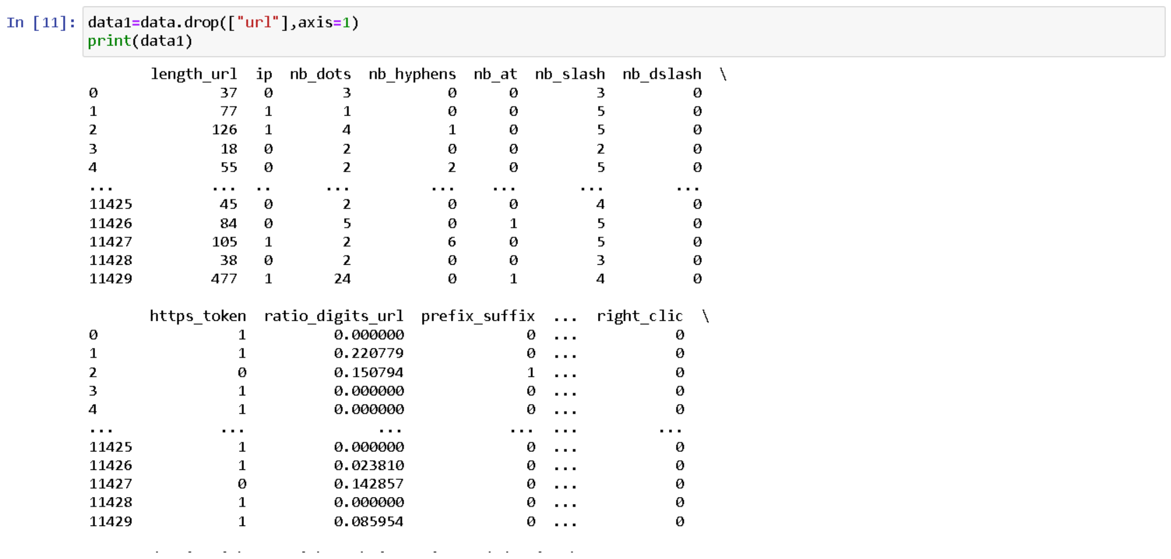
```
In [2]: data = pd.read_csv("Final_URL_Data.csv")
        data.head(10)
```

Out[2]:

	url	length_url	ip	nb_dots	nb_hyphens	nb_at	nb_slash	nb_dslash	https_token	ratio_digits_url	...	right_clic	dom
0	http://www.crestonwood.com/router.php	37	0	3	0	0	3	0	1	0.000000	...	0	
1	http://shadetreetechnology.com/V4/validation/a...	77	1	1	0	0	5	0	1	0.220779	...	0	
2	https://support-appleld.com.secureupdate.duila...	126	1	4	1	0	5	0	0	0.150794	...	0	
3	http://rgipt.ac.in	18	0	2	0	0	2	0	1	0.000000	...	0	
4	http://www.iracing.com/tracks/gateway-motorspo...	55	0	2	2	0	5	0	1	0.000000	...	0	
5	http://appleid.apple.com-app.es/	32	0	3	1	0	3	0	1	0.000000	...	0	
6	http://www.mutuo.it	19	0	2	0	0	2	0	1	0.000000	...	0	
7	http://www.shadetreetechnology.com/V4/validati...	81	1	2	0	0	5	0	1	0.259259	...	0	
8	http://vamoaestudiarmedicina.blogspot.com/	42	0	2	0	0	3	0	1	0.000000	...	0	
9	https://parade.com/425836/joshwigler/the-amazi...	104	0	1	10	0	6	0	0	0.076923	...	0	

10 rows × 24 columns

STEP 3: Dropping the column named "url" as its not needed in training.



```
In [11]: data1=data.drop(["url"],axis=1)
         print(data1)
```

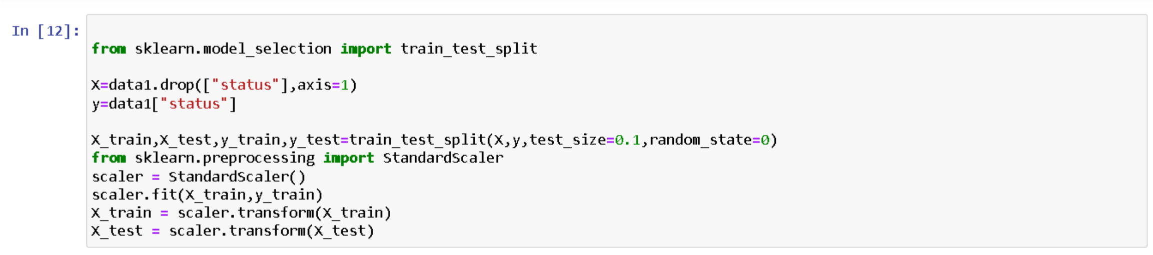
```
       length_url  ip  nb_dots  nb_hyphens  nb_at  nb_slash  nb_dslash  \
0              37   0        3           0      0         3          0
1              77   1        1           0      0         5          0
2             126   1        4           1      0         5          0
3              18   0        2           0      0         2          0
4              55   0        2           2      0         5          0
...           ...  ..      ...         ...    ...       ...        ...
11425          45   0        2           0      0         4          0
11426          84   0        5           0      1         5          0
11427         105   1        2           6      0         5          0
11428          38   0        2           0      0         3          0
11429         477   1       24           0      1         4          0

       https_token  ratio_digits_url  prefix_suffix  ...  right_clic  \
0                1          0.000000              0  ...           0
1                1          0.220779              0  ...           0
2                0          0.150794              1  ...           0
3                1          0.000000              0  ...           0
4                1          0.000000              0  ...           0
...            ...               ...            ...  ...         ...
11425            1          0.000000              0  ...           0
11426            1          0.023810              0  ...           0
11427            0          0.142857              0  ...           0
11428            1          0.000000              0  ...           0
11429            1          0.085954              0  ...           0
```

STEP 4: Dividing the data into testing and training set using sklearn library to split the data into testing and training set.

We remove the status value from testing set.

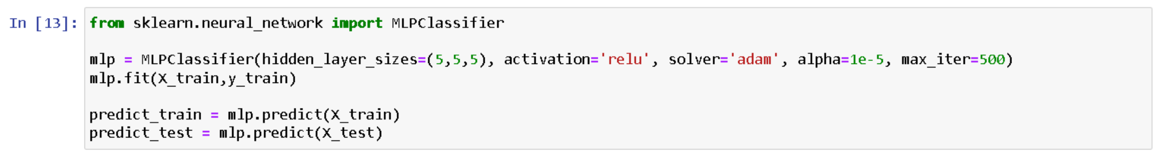
It is required to use feature scaling for the data given as it is highly sensitive to it.

Hence here we need to import Standard scaler to Scale features from [0,1].



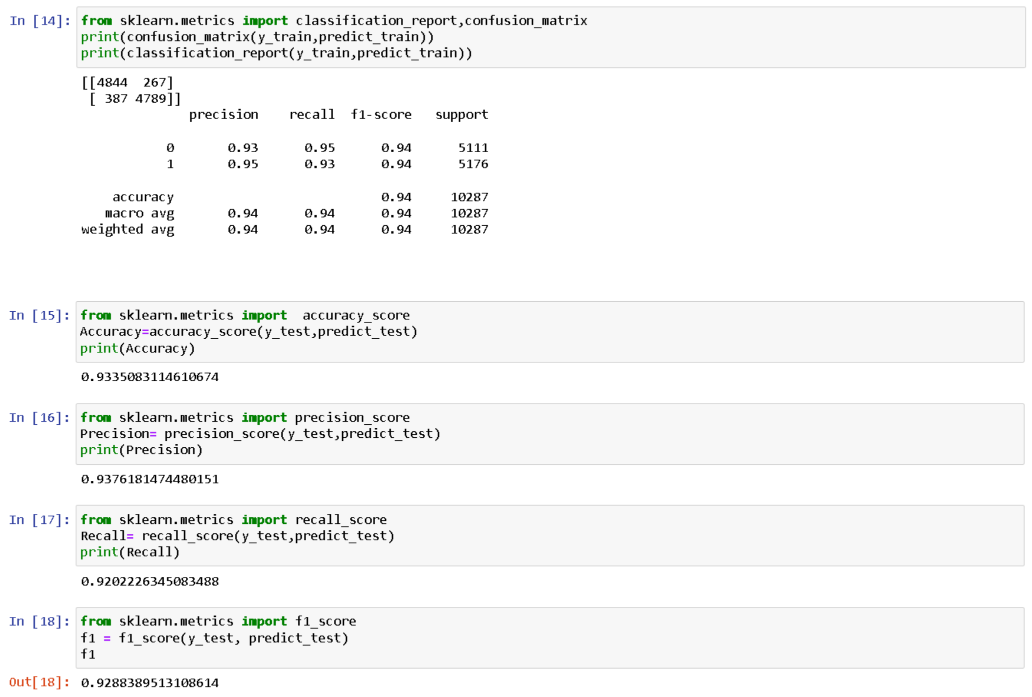
```
In [12]:
from sklearn.model_selection import train_test_split

X=data1.drop(["status"],axis=1)
y=data1["status"]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=0)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train,y_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

STEP 5: Importing MLPClassifier to apply RNN by giving Activation function as ReLU and solver as Adam number of iterations 500.



```
In [13]: from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(5,5,5), activation='relu', solver='adam', alpha=1e-5, max_iter=500)
mlp.fit(X_train,y_train)

predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)
```

STEP 6: Creating a confusion matrix to know accuracy, f1-score , precision value , Recall value.



```
In [14]: from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_train,predict_train))
print(classification_report(y_train,predict_train))

[[4844  267]
 [ 387 4789]]
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      5111
           1       0.95      0.93      0.94      5176

    accuracy                           0.94     10287
   macro avg       0.94      0.94      0.94     10287
weighted avg       0.94      0.94      0.94     10287


In [15]: from sklearn.metrics import  accuracy_score
Accuracy=accuracy_score(y_test,predict_test)
print(Accuracy)

0.9335083114610674

In [16]: from sklearn.metrics import precision_score
Precision= precision_score(y_test,predict_test)
print(Precision)

0.9376181474480151

In [17]: from sklearn.metrics import recall_score
Recall= recall_score(y_test,predict_test)
print(Recall)

0.9202226345083488

In [18]: from sklearn.metrics import f1_score
f1 = f1_score(y_test, predict_test)
f1

Out[18]: 0.9288389513108614
```

8. RESULTS AND DISCUSSIONS

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
KNN Classifier	83.94%	83.33%	84.43%	83.88%
Random Forest Classifier	94.09%	94.54%	93.45%	93.99%
XG BOOST Classifier	96.63%	96.47%	96.72%	96.60%
RNN Classifier	93.35%	93.76%	92.02%	92.88%

The above table shows the performance measures of the models that we have built for classification. Accuracy is the most important measure which says how good the model performs. Accuracy is the ratio of number of correct predictions to the total number of predictions. The XG BOOST Classifier is having the accuracy of 96.63 percent, and then Random Forest, RNN and KNN classifiers with 94.09%, 93,35% and 83.94% of accuracies respectively.

Precision measures the performance of the model in making positive predictions in a categorical manner. It is defined as the ratio of true positive predictions to the sum of true positive and false positive predictions. The XG BOOST Classifier is having a precision of 96.47% which is the highest, Random Forest with 94.54%, RNN with 93.76% and KNN with 83.33%.

Recall is the ratio of number of true positive predictions to the total number of actual positive predictions. This describes the ability of the model to detect positive predictions. The XG BOOST Classifier is having the Recall of 96.72 percent, and then Random Forest, RNN and KNN classifiers with 93.45%, 92.02% and 84.43% respectively.

F1-score is defined as the harmonic mean of precision and recall. F1-score describes the accuracy of the model based on the dataset. It describes the balance of classes in the dataset for classification. The highest F1-score is given by XG BOOST classifier which is 96.60%. The Random forest, RNN and KNN classifiers are having F1-scores of 93.99%, 92.88% and 83.88% respectively.

9. CONCLUSION AND FUTURE SCOPE

This project aims in improving the Phishing URL detection by using various methods in Machine Learning. The best possible approaches were attempted for collecting the best set of features of the URLs and finding the suitable Machine Learning approaches. In this process, four classifiers were built based on supervised machine learning, ensemble learning and deep learning. The KNN Classifier, using a simple algorithm, gave an accuracy of 84% which is good but not the best. The Random Forest Classifier gave an accuracy of 94% which is pretty good. The reason behind this improvement of the classifier is, Random forest is based on bagging technique in ensemble learning. It uses multiple models to decide the final outcome. The XG Boost Classifier gave an highest accuracy of 97%. XG Boost is based on boosting technique from ensemble learning, where models were built in sequential manner. Every model learns from its previous model to become a better performer. The technique along with the best set of features resulted in the performance of XG Boost. At last, the RNN classifier gave an accuracy of 93% which is also pretty good. As we are dealing with a normal dataset the RNN didn’t work with its full potential. It would have performed much better if there was huge amount of data. Comparing the results, it can be concluded that the XG Boost is the best performing model in the current scenario. Also, XG Boost is known for its speed, efficiency and reliability. So, XG Boost can also be used in other sectors for solving Classification and Regression kind of problems.

In the future, this methodology can be used to detect phishing websites accurately and reduce the major social engineering problem in the society. This methodology can be further integrated into various technologies and improve safety of the users.

References

1. “History of Phishing”, https://cofense.com/knowledge-center/history-of-phishing/

2. “Love bug virus creates worldwide chaos”, https://www.theguardian.com/world/2000/may/05/jamesmeek

3. “A new phishing attack lurking to scam banking customers: Advisory”, https://timesofindia.indiatimes.com/business/india-business/a-new-phishing-attack-lurking-to-scam-banking-customers-advisory/articleshow/85236685.cms

4. Rami M. Mohammad, Fadi Thabtah, Lee McCluskey, “Phishing Website Features”, http://eprints.hud.ac.uk/id/eprint/24330/6/MohammadPhishing14July2015.pdf

5. Junaid Rashid, Toqeer Mahmood, Muhammad Wasif Nisar, Tahira Nazir, “Phishing Detection Using Machine Learning Technique”, *First International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, INSPEC Accession Number: 20278066, DOI: 10.1109/SMART-TECH49988.2020.00026

6. Uur Ozker, Ozgur Koray Sahingoz, "Content Based Phishing Detection with Machine Learning", *2020 International Conference on Electrical Engineering (ICEE)*, INSPEC Accession Number: 20154203, DOI: 10.1109/ICEE49691.2020.9249892

7. S. Naksomboon, C. Charnsripinyo, N. Wattanapongsakorn, "Considering Behavior of Sender in Spam Mail Detection", *INC2010: 6th International Conference on Networked Computing*, INSPEC Accession Number: 11372876

8. Shovan Chowdhury , Marco P. Schoen, "Classification using Supervised Machine Learning Techniques", *2020 Intermountain Engineering, Technology and Computing (IETC)*, INSPEC Accession Number: 20154043, DOI:10.1109/IETC47856.2020.9249211