1) INTRODUCTION:-

(1.1) Project Review:-

Web spam can significantly deteriorate the quality of search engine results. Thus there is a large incentive for commercial search engines to detect spam pages efficiently and accurately. In this paper we present a spam detection system that combines link-based and content-based features, and uses the topology of the Web graph by exploiting the link dependencies among the Web pages. We find that linked hosts tend to belong to the same class: either both are spam or both are non-spam. We demonstrate three methods of incorporating the Web graph topology into the predictions obtained by our base classifier: (i) clustering the host graph, and assigning the label of all hosts in the cluster by majority vote, (ii) propagating the predicted labels to neighboring hosts, and (iii) using the predicted labels of neighboring hosts as new features and retraining the classifier. The result is an accurate system for detecting Web spam, tested on a large and public dataset, using algorithms that can be applied in practice to large-scale Web data.

(1.2) Purpose:-

Over the past few years, following the growth of communication networks, internet as the biggest has been widespread popular. Using anonymity provided by the internet, hustlers set out to deceive people with false offers and make themselves look legitimate in this medium (Arun et al., 2012). With increased terminals for access to information, internet banking creates the need for using reliable methods in order to control and use confidential and vital information. Today, financial crimes are transformed from direct attacks into indirect attacks. In other words, instead of bank robbery, criminals try to target bank's clients with a specific trick (Vrîncianu & Popa, 2010). Attacks on computer security are classified in three types: physical attacks, synthetic attacks, and semantic attacks (He et al., 2011). Phishing is one of the types of semantic attacks. In these types of attacks, vulnerabilities in the users are targeted; for example, the way users interpret computer messages (He et al., 2011), because most of the users read information sources without verifying them, and respond their demands.

(2) LITERATURE SURVEY:-

According to this paper we people are highly dependent on the internet. For performing online shopping and online activities like banking, mobile recharge and more activities are done only through internet. Here phishing is nothing but a type of website threat which illegally collects the original website information such as login id, password and credit card information. Here we will use an efficient machine learning based web phishing detection technique.

(2.1) Problem Identification:-

There are many users who purchase products through online platform and the payment is done through e-banking. There are some fake banking websites in which they collect the more sensitive information like username, password, credit card details etc , for illegal purpose. This type of websites are called phishing website. Here web phishing is one of the security threat to webservices on the internet.

(2.2)Problem Solution:-

To overcome the problem of phishing website whenever we are clicking on one website it must show an alert box like it is a secure website or it is not a secure website. Then another way is that we can scan the website in order to prevent our system or mobile from the phishing attack. Even though technologies are there we as the user have to be aware of the websites whether it is secure or not. We should not click any unwanted websites.

(2.3)REFERENCES:-

[1] Higashino, M., et al. An Anti-phishing Training System for Security Awareness and Education Considering Prevention of Information Leakage. in 2019 5th International Conference on Information Management (ICIM). 2019. [2] H. Bleau, Global Fraud and Cybercrime Forecast,. 2017. [3] Michel Lange, V., et al., Planning and production of grammatical and lexical verbs in multi-word messages. PloS one, 2017. 12(11): p. e0186685-e018668.

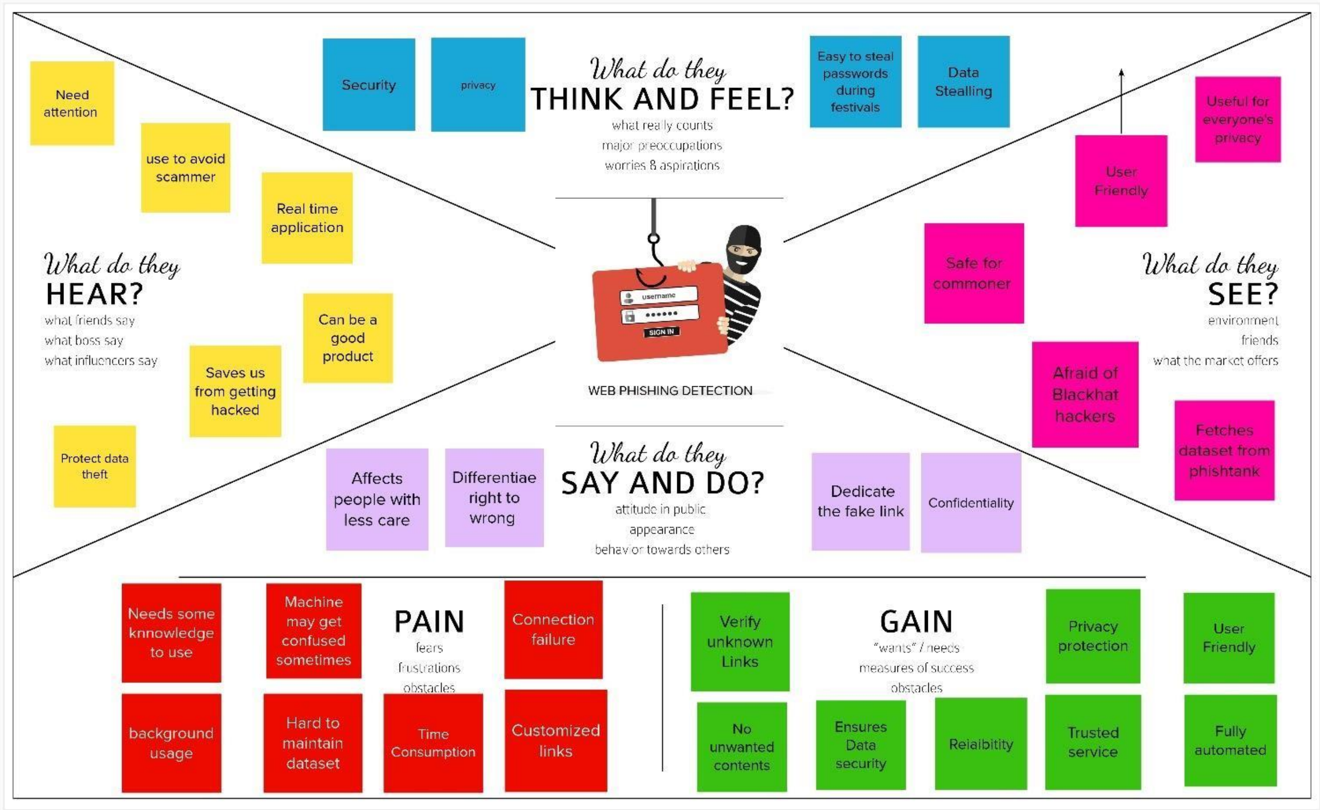
(2.4)CONCLUSION:-

This paper aims to enhance detection method to detect phishing website using machine learning technology. Also , classifiers generated by machine learning algorithms identify legitimate phishing websites.The proposed technique can detect new temporary phishing sites and reduce the damage caused by phishing attacks. The performance of the proposed technique based on machine learning is more effective that previous phishing detection technologies. In the future, it will be useful to investigate the impact of feature selection using various algorithms.

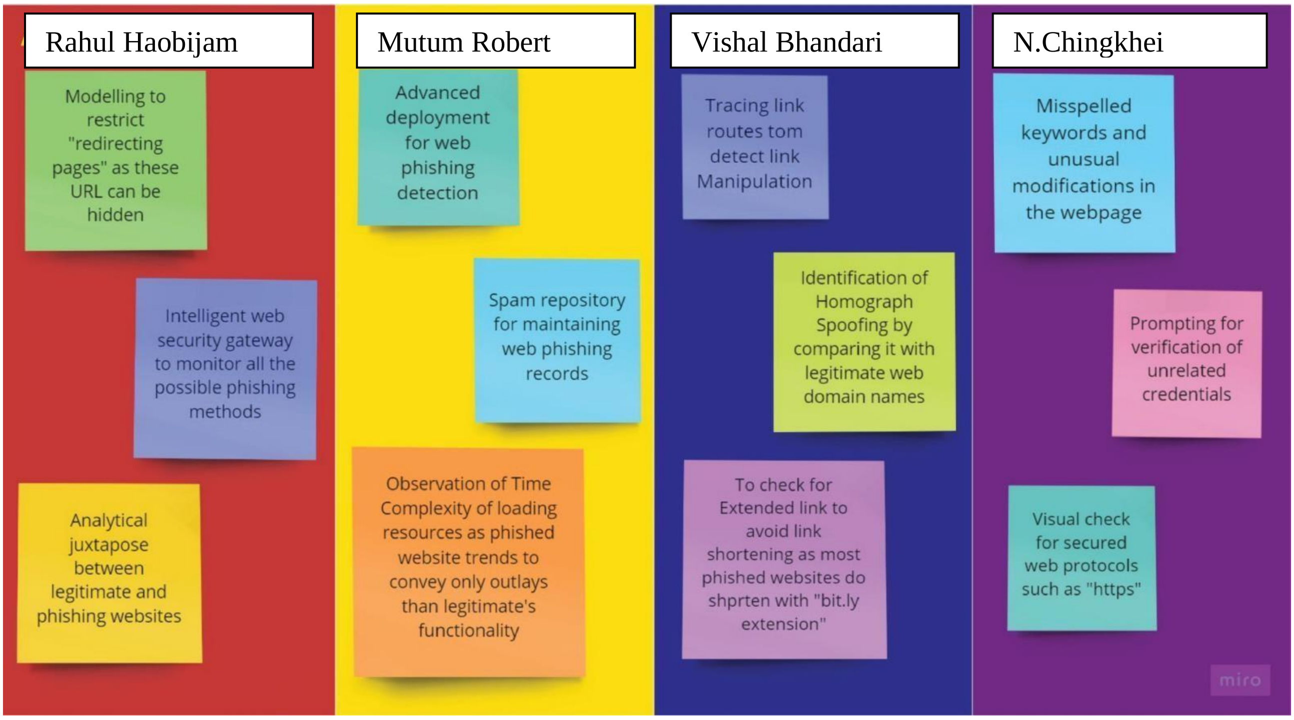
(3) IDEATION & PROPOSED SOLUTION :-

To make future development easy, we proposed a rule-based system by extracting the hidden knowledge from our classification model. - We provide an easy to use chrome extension from our proposed rule-based method to detect phishing attacks on internetbanking websites.

(3.1) Empathy Map:-



(3.2) Ideation & Brainstorming:-



(3.3) Proposed Solution :-

Project team shall fill the following information in proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Web phishing tends to steal a lots of informationfrom the user during online transaction like username, password, important documents that has been attached to that websites. There are Multiple Types of Attacks happens here every day, but there is no auto detection Process through Machine Learning is achieved
2.	Idea / Solution description	Through ML and data mining techniques like classification algorithm user can able to attain a warning signal to notify these phishing websites which helps the user to safeguard their identities and their login credentials etc. python is the language that helps to enable these techniques for the online users
3.	Novelty / Uniqueness	This project not only able to identify the malicious websites it also has the ability to automatically block these kind of websites completely in the future when it has been identified and also blocks some various mails /ads from these malicious websites
4.	Social Impact / Customer Satisfaction	This web phishing detection project attains thecustomer satisfaction by discarding various kindsof malicious websites to protect their privacy.This project is not only capable of using by ansingle individual ,a large social community and a organization can use this web phishing detectionto protect their privacy. This project helps toblock various malicious websites simultaneously.
5.	Business Model (Revenue Model)	This developed model can be used as an enterprise applications by organizations which handles sensitive information and also can be sold to government agencies to prevent the loss of potential important data.
6.	Scalability of the Solution	This project’s performance rate will be high and it also provide many capabilities to the user without reducing its efficieny to detect the malicious websites. thus scalability of this project will be high .

Problem Solution Fit:-

Project Title: Web Phishing Detection | **Project Design Phase-I - Solution Fit Template** | **Team ID: PNT2022TMID09696**

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

An internet user who is willing to shop products online.

An enterprise user surfing through the internet for some information.

6. CUSTOMER CONSTRAINTS CC

Customers have very little awareness on phishing websites.

They don't know what to do after losing data.

5. AVAILABLE SOLUTIONS AS

Which solutions are available

The already available solutions are blocking such phishing sites and by triggering a message to the customer about dangerous nature of the website.

But the blocking of phishing sites are not more affective as the attackers use a different/new site to steal potential data thus a AI/ML model can be used to prevent customers from these kinds of sites from stealing data

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

The phishing websites must be detected in a earlier stage .

The user can be blocked from entering such sites for the prevention of such issues.

9. PROBLEM ROOT CAUSE RC

The hackers use new ways to cheat the naïve users.

Very limited research is performed on this part of the internet.

7. BEHAVIOUR BE

The option to check the legitimacy of the Websites is provided.

Users get an idea what to do and more importantly what not to do.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

A trigger message can be popped warning the user about the site.

Phishing sites can be blocked by the ISP and can show a "site is blocked" or "phishing site detected" message.

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?

The customers feel lost and insecure to use the internet after facing such issues.

Unwanted panicking of the customers is felt after encounter loss of potential data to such sites.

10. YOUR SOLUTION SL

An option for the users to check the legitimacy of the websites is provided.

This increases the awareness among users and prevents misuse of data, data theft etc.,

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE

Customers tend to lose their data to phishing sites.

8.2 OFFLINE

Customers try to learn about the ways they get cheated from various resources viz., books, other people etc.,

Identify strong TR & EM

(4) REQUIREMENTS ANALYSIS:-

(4.1) Functional Requirements;-

First, we get real traffic flow from ISP. The data set includes traffic flow for 40 minutes and 24 hours. We construct the graph structure of traffic flow and analyze the characteristics of web phishing from the view of the graph.

Each piece of data contains the following fields.

AD: user node number.

IP: user IP address.

TS: access time.

URL: Uniform Resource Locator, access web address.

REF: request page source.

UA: user browser type.

DST: server address to access.

CKE: User Cookie.

(4.2) Non-Functional Requirements:-

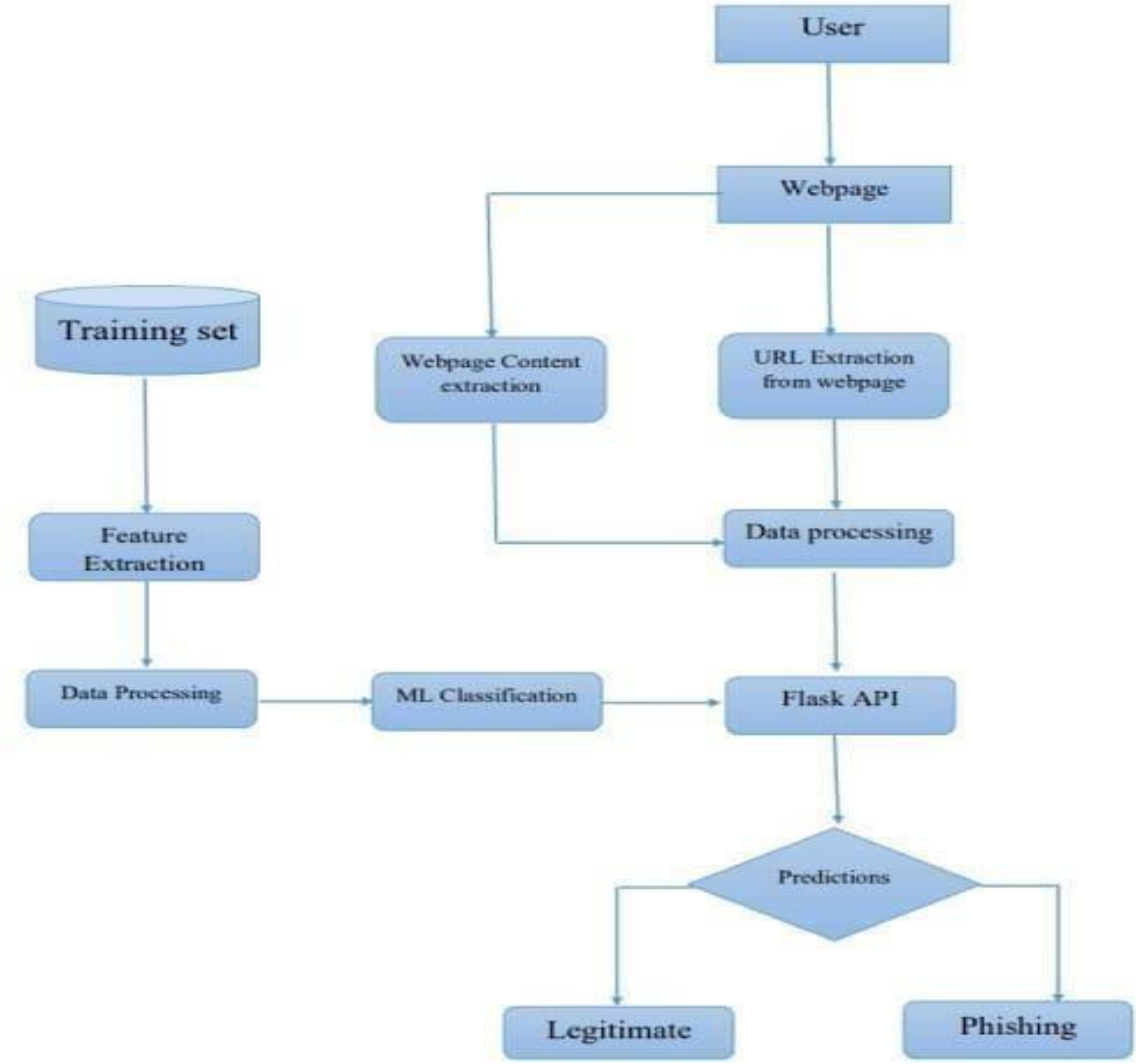
Researchers have conducted lot of work in security [12–18], including secure routing [19–21], intrusion detection [22–27], intrusion prevention [28], and smart grids security [29]. Different from research problems in wireless networks [30–60] and energy networks [61–64], web phishing is the attempt to acquire sensitive information such as usernames, passwords, and credit card details, often for malicious reasons, by masquerading as a trustworthy website on the Internet. Researchers present some solutions to detect web phishing as follows.

When we judge whether a specific website is web phishing, the direct way is to use a white list or black list. We may search the URL in some database and decide. Pawan Prakash *et al.* [10] presented two ways to detect phishing websites by the blacklist. The first way includes five heuristics to enumerate simple combinations of known phishing sites to discover new phishing URLs. The second way consists of an approximate matching algorithm that dissects a URL into multiple components that are matched individually against entries in the blacklist. Many well-known browser vendors such as Firefox [65] and Chrome [66] also used a self-built or third-party black-white list, to identify whether the URL is a phishing site. This method is very accurate, but its blacklist or whitelist usually relies on manual maintaining and reviewing. Obviously, these methods are not real time and may cost a lot of time and effort.

(5) PROJECT DESIGN:-

(5.1) Data Flow Diagram:-

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored



(5.2) User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1

		USN-2	As a user, I will receive confirmation emailonce I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Webuser)	User input	USN-1	As a user i can input the particular URL in therequired field and waiting for validation.	I can go access the websitewithout any problem	High	Sprint-1
Customer Care Executive	Feature extraction	USN-1	After i compare in case if none found on comparison then we can extract feature using heuristic and visual similarity approach.	As a User i can have comparison between websites for security.	High	Sprint-1
Administrator	Prediction	USN-1	Here the Model will predict the URL websites using Machine Learning algorithms such as Logistic Regression, KNN	In this i can have correct prediction on the particular algorithms	High	Sprint-1
	Classifier	USN-2	Here i will send all the model output to classifier inorder to produce final result.	I this i will find the correct classifier for producing the result	Medium	Sprint-2

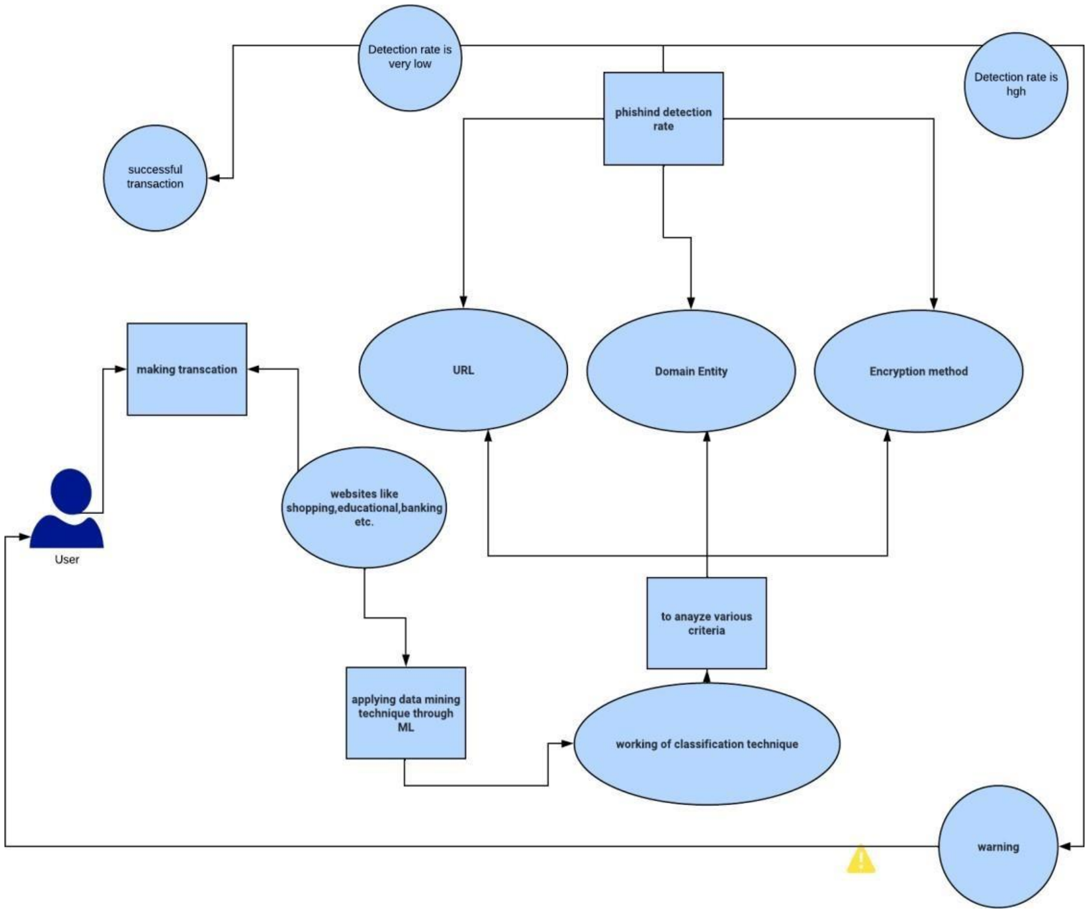
(5.3) Solution & Technical Architecture:-

Solution Architecture:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software toproject stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed,.

(5.4) Solution Architecture Diagram:



ARCHITECTURE DIAGRAM FOR WEB PHISHING DETECTION

(5.5) User Stories:-

STAGE	AWARENESS	CONSIDERATION	DECISION	SERVICE	LOYALTY
CUSTOMER ACTION	To become aware by watching some videosabout phishing websites.	Compare secure and insecurewebsites.	Customer decides to avoid the scam website in order to prevent virus attackfrom their computer.	Customer can contact customercare service.	They can share their experience about using the website.
TOUCH POINTS	Social media, Traditional media	Website Certifications	Website, Mobile app	Web Service	Review sites
CUSTOMER EXPERIENCE	Interested to get aware of phishing websites	Awareness of phishing websites	Plan to Detect Legal and Phishing websites to preventthe attacks.	Provides trustiness of thewebsite.	Satisfied, Excited
KPIS	They check the amount of people getting aware of thephishing attacks	They see the count of visits ofthe website.	They check the Conversional rateof visiting the websites.	It provides Less time in producingthe result of the website visitors.	Provides Customer satisfaction score.
BUSINESS GOALS	Provides an Increasein the awareness of the phishing website attacks.	Aims on detecting phishingwebsite with high accuracy.	It gives an Increasein the customer rateof visiting the websites.	It provides anIncrease in the customer satisfaction.	It Generates some positive reviews from the customer side.

(6) Project Planning And Scheduling:-

(6.1) Sprint Planning & Estimation:-

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Homepage	USN-1	As a user, I can explore the resources of thehomepage for the functioning	10	Low	Rahul haobijam
Sprint-1		USN-2	As a user, I can learn about the various sides of the web phishing and be aware of the scams	5	High	M.Rob ert
Sprint-2	Final page	USN-3	As a user, I can explore the resources of thefinal page for the functioning	15	Low	Vishal,

Sprint-3	Prediction	USN-4	As a user, I can predict the URL easily for detecting whether the website is legitimate ornot	10	High	N.Chingkhei
	Dashboard					
Sprint-4	Chat	USN-5	As a user, I can share the experience or contactthe admin for the support	10	High	Rahul,Robert
Sprint-1	Homepage	USN-6	As a admin, we can design interface and maintain the functioning of the website	5	High	Vishal
Sprint-2	Final page	USN-7	As a admin, we can design the complexity ofthe website for making it user-friendly	5	Medium	N.Chingkhei
Sprint-3	Prediction	USN-8	As a admin, we can use various ML classifier model for the accurate result for the detection ofURL	10	High	Rahul,Vishal
	Dashboard					
Sprint-4		USN-9	As a admin, we can response to the user message for improvement of the website	10	Medium	Robert

(6.2)Project Tracker, Velocity & Burndown Char (4Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date(Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	12 Nov 2022

Velocity:

Imagine we have a 10-day sprint duration, and thevelocity of the team is 20 (points per sprint). Let’s calculate the team’s average velocity (AV) per iteration unit (story points per day).

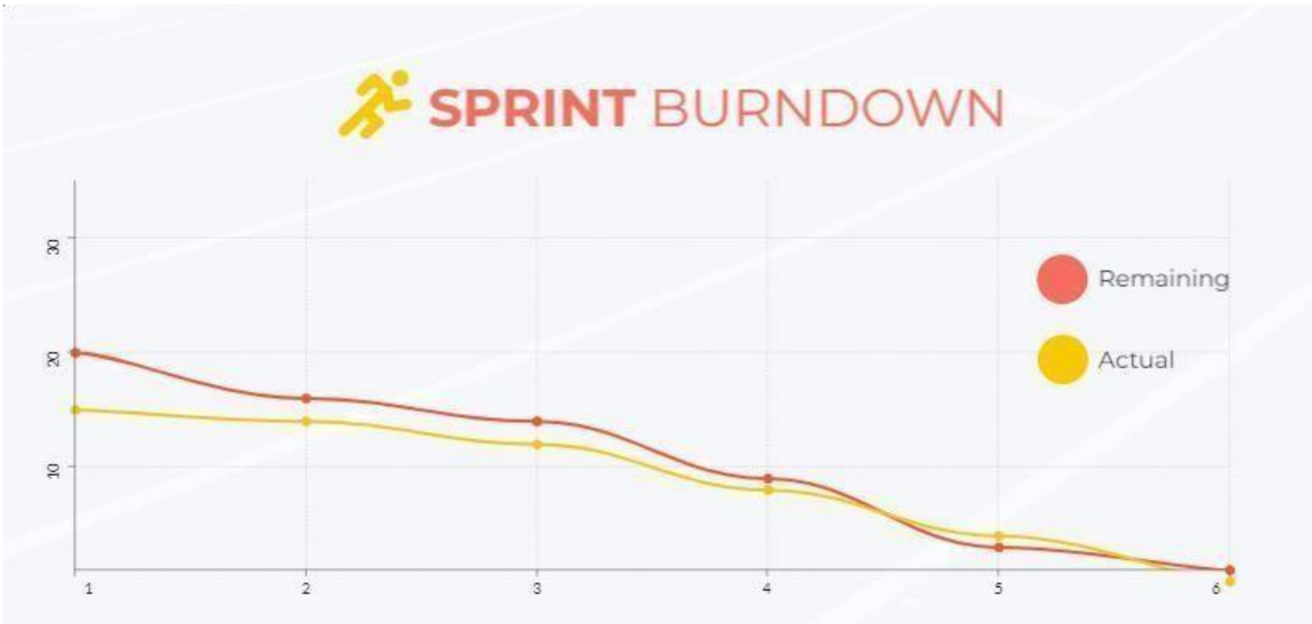
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

We have a 6-day sprint duration, and the velocity of the team is 20 (points per sprint). So our team's average velocity (AV) per iteration unit (storypoints per day)

AV = (Sprint Duration / Velocity) = 20 /6 = 3.33

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum.However, burn down charts can be applied to any project containing measurable progress over time.



Reference:

https:/[/www.visual-paradigm.com/scrum/scrum-burndown-chart/](https://www.visual-paradigm.com/scrum/scrum-burndown-chart/)
https:/[/www.visme.co/templates/charts/sprint-burndown-chart-1425285230/](https://www.visme.co/templates/charts/sprint-burndown-chart-1425285230/)

(7) CODING & SOLUTIONING

Phishing Website Detection by Machine Learning Techniques

Final project of AI & Cybersecurity Course

1. Objective:

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. The objective of this project is to train machine learning models and deep neural nets on the dataset created to predict phishing websites. Both phishing and benign URLs of websites are gathered to form a dataset and from them required URL and website content-based features are extracted. The performance level of each model is measures and compared.

This project is worked on Google Collaboratory.
The required packages for this notebook are imported when needed.

2. Loading Data:

The features are extracted and store in the csv file. The working of this can be seen in the 'Phishing Website Detection_Feature Extraction.ipynb' file.

The reulted csv file is uploaded to this notebook and stored in the dataframe.

In [0]:

```
#importing basic packages
import pandas as pd
import numpy  as  np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [0]:

```
#Loading the data
data0 = pd.read_csv('5.urldata.csv')
data0.head()
```

Out[0]:

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	graphicrive	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
	r.net																	
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	0
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0

3. Familiarizing with Data

In this step, few dataframe methods are used to look into the data and its features.

In [0]:

```
#Checking the shape of the dataset
data0.shape
```

Out[0]:

```
(10000, 18)
```

In [0]:

```
#Listing the features of the dataset
data0.columns
```

Out[0]:

```
Index(['Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
       'Redirection', 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Reco
rd',
       'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over',
       'Right_Click', 'Web_Forwards', 'Label'],
      dtype='object')
```

In [0]:

```
#Information about the dataset
data0.info()
```

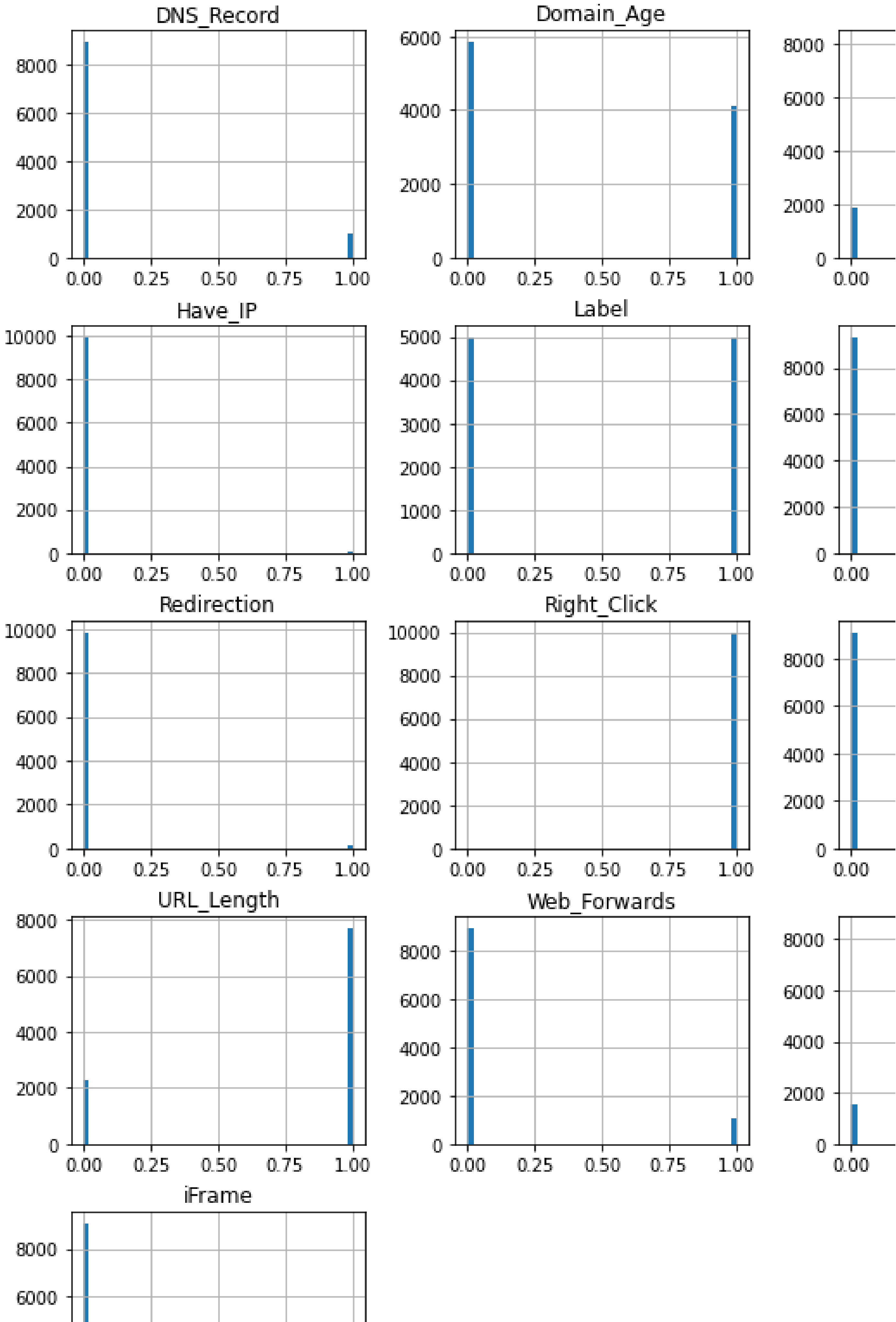
```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
-----  ---------       --------------  --------
 0   Domain          10000 non-null  object
 1   Have_IP         10000 non-null  int64
 2   Have_At         10000 non-null  int64
 3   URL_Length      10000 non-null  int64
 4   URL_Depth       10000 non-null  int64
 5   Redirection     10000 non-null  int64
 6   https_Domain    10000 non-null  int64
 7   TinyURL         10000 non-null  int64
 8   Prefix/Suffix   10000 non-null  int64
 9   DNS_Record      10000 non-null  int64
 10  Web_Traffic     10000 non-null  int64
 11  Domain_Age      10000 non-null  int64
 12  Domain_End      10000 non-null  int64
 13  iFrame          10000 non-null  int64
 14  Mouse_Over      10000 non-null  int64
 15  Right_Click     10000 non-null  int64
 16  Web_Forwards    10000 non-null  int64
 17  Label           10000 non-null int64
dtypes: int64(17), object(1)
memory usage: 1.4+ MB
```

4. Visualizing the data

Few plots and graphs are displayed to find how the data is distributed and the how features are related to each other.

In [0]:

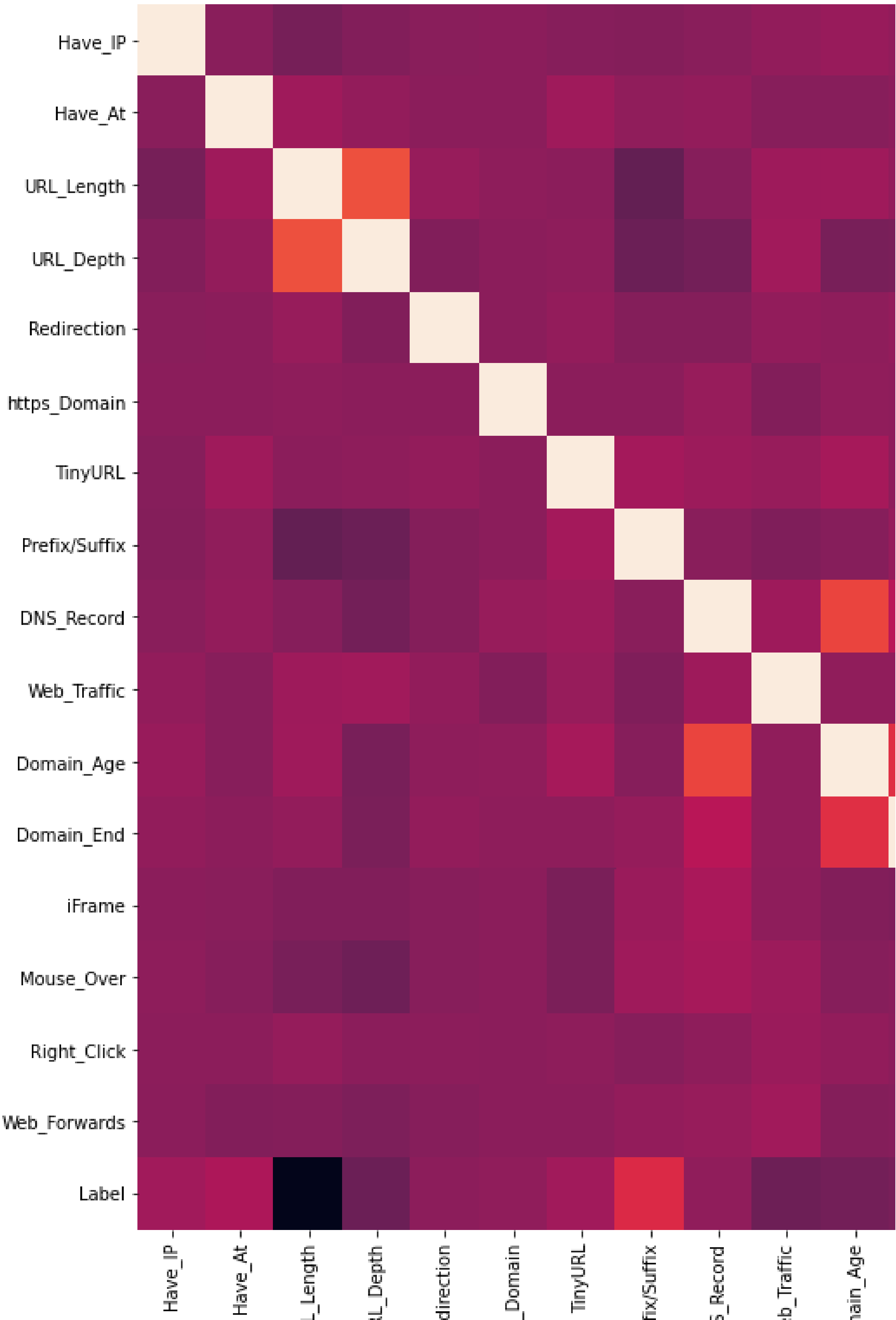
```
#Plotting the data distribution
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



DNS_Record
0
2000
4000
6000
8000
0.00
0.25
0.50
0.75
1.00
Domain_Age
0
2000
4000
6000
Have_IP
10000
Label
1000
3000
5000
Redirection
Right_Click
URL_Length
Web_Forwards
iFrame

In [0]:

```
#Correlation heatmap

plt.figure(figsize=(15,13))
sns.heatmap(data0.corr())
plt.show()
```



Have_IP
Have_At
URL_Length
URL_Depth
Redirection
https_Domain
TinyURL
Prefix/Suffix
DNS_Record
Web_Traffic
Domain_Age
Domain_End
iFrame
Mouse_Over
Right_Click
Web_Forwards
Label

5. Data Preprocessing & EDA

Here, we clean the data by applying data preprocesssing techniques and transform the data to use it in the models.

In [0]:

```
data0.describe()
```

Out[0]:

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.00000	10000.00000	10000.000000	10000.000000
mean	0.005500	0.022600	0.773400	3.072000	0.013500	0.000200	0.090300	0.093200	0.100800	0.845700	0.413700	0.8099	0.090900	0.06660	0.99930	0.105300	0.500000
std	0.073961	0.148632	0.418653	2.128631	0.115408	0.014141	0.286625	0.290727	0.301079	0.361254	0.492521	0.3924	0.287481	0.24934	0.02645	0.306955	0.500025
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000	0.000000	0.00000	0.00000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.0000	0.000000	0.00000	1.00000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.0000	0.000000	0.00000	1.00000	0.000000	0.500000
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.0000	0.000000	0.00000	1.00000	0.000000	1.000000

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000	1.000000	1.00000	1.00000	1.000000	1.000000

The above obtained result shows that the most of the data is made of 0's & 1's except 'Domain' & 'URL_Depth' columns. The Domain column doesnt have any significance to the machine learning model training. So dropping the *'Domain'* column from the dataset.

In [0]:

```
#Dropping the Domain column
data = data0.drop(['Domain'], axis = 1).copy()
```

This leaves us with 16 features & a target column. The *'URL_Depth'* maximum value is 20. According to my understanding, there is no necessity to change this column.

In [0]:

```
#checking the data for null or missing values
data.isnull().sum()
```

Out[0]:

```
Have_IP          0
Have_At          0
URL_Length       0
URL_Depth        0
Redirection      0
https_Domain     0
TinyURL          0
Prefix/Suffix    0
DNS_Record       0
Web_Traffic      0
Domain_Age       0
Domain_End       0
iFrame           0
Mouse_Over       0
Right_Click      0
Web_Forwards     0
Label            0
dtype: int64
```

In the feature extraction file, the extracted features of legitmate & phishing url datasets are just concatenated without any shuffling. This resulted in top 5000 rows of legitimate url data & bottom 5000 of phishing url data.

To even out the distribution while splitting the data into training & testing sets, we need to shuffle it. This even evades the case of overfitting while model training.

In [0]:

```
# shuffling the rows in the dataset so that when splitting the train and test set are equally distributed
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Out[0]:

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	0	0	1	2	0	0	0	1	0	1	0	1	0	0	1	0	1
1	0	0	1	5	0	0	0	0	0	1	0	1	0	0	1	1	0
2	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	0
3	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1	0	0
4	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1

From the above execution, it is clear that the data doesnot have any missing values.

By this, the data is throughly preprocessed & is ready for training.

6. Splitting the Data

In [0]:

```
# Sepratating & assigning features and target columns to X & y
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape
```

Out[0]:

```
((10000, 16), (10000,))
```

In [0]:

```
# Splitting the dataset into train and test sets: 80-20 split
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 = 12)
X_train.shape, X_test.shape
```

Out[0]:

```
((8000, 16), (2000, 16))
```

7. Machine Learning Models & Training

From the dataset above, it is clear that this is a supervised machine learning task. There are two major types of supervised machine learning problems, called classification and regression.

This data set comes under classification problem, as the input URL is classified as phishing (1) or legitimate (0). The supervised machine learning models (classification) considered to train the dataset in this notebook are:

- Decision Tree
- Random Forest
- Multilayer Perceptrons
- XGBoost
- Autoencoder Neural Network
- Support Vector Machines

In [0]:

```
#importing packages
from sklearn.metrics import accuracy_score
```

In [0]:

```
# Creating holders to store the model performance results
ML_Model = []
acc_train = []
acc_test = []

#function to call for storing the results
def storeResults(model, a,b):
  ML_Model.append(model)
  acc_train.append(round(a, 3))
  acc_test.append(round(b, 3))
```

7.1. Decision Tree Classifier

Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.

In the machine learning setting, these questions are called tests (not to be confused with the test set, which is the data we use to test to see how generalizable our model is). To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.

In [0]:

```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
```

Out[0]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [0]:

```
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
```

Performance Evaluation:

In [0]:

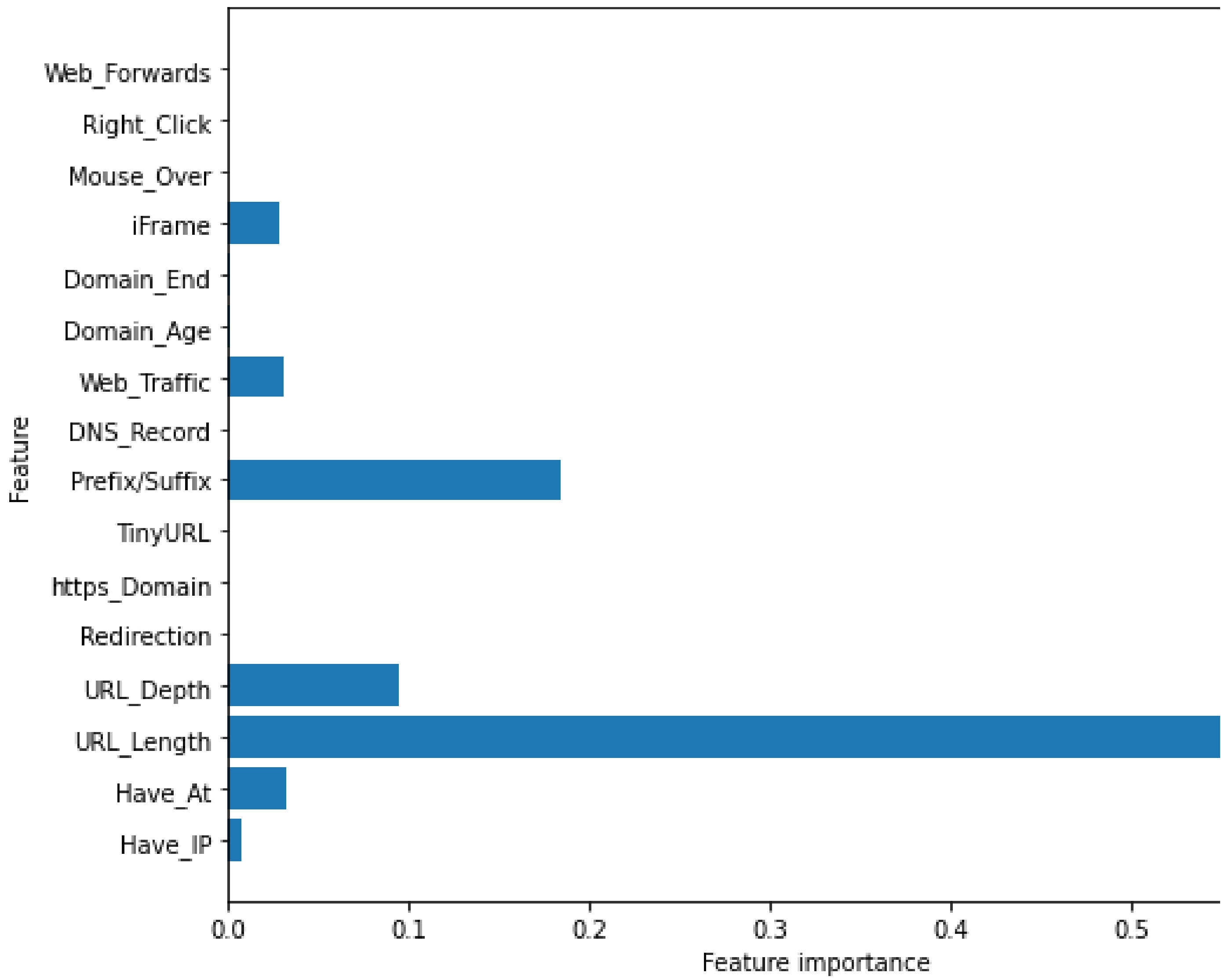
```
#computing the accuracy of the model performance
```

```
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)

print("Decision Tree: Accuracy on training Data:
{:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))
```

```
Decision Tree: Accuracy on training Data: 0.810
Decision Tree: Accuracy on test Data: 0.826
```

In [0]:

```
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), tree.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is important.
#Caution: Execute only once to avoid duplications.
storeResults('Decision Tree', acc_train_tree, acc_test_tree)
```

7.2. Random Forest Classifier

Random forests for regression and classification are currently among the most widely used machine learning methods.A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.

If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. To build a random forest model, you need to decide on the number of trees to build (the n_estimators parameter of RandomForestRegressor or RandomForestClassifier). They are very powerful, often work well without heavy tuning of the parameters, and don't require scaling of the data.

In [0]:

```
# 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[0]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [0]:

```
#predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)
```

Performance Evaluation:

In [0]:

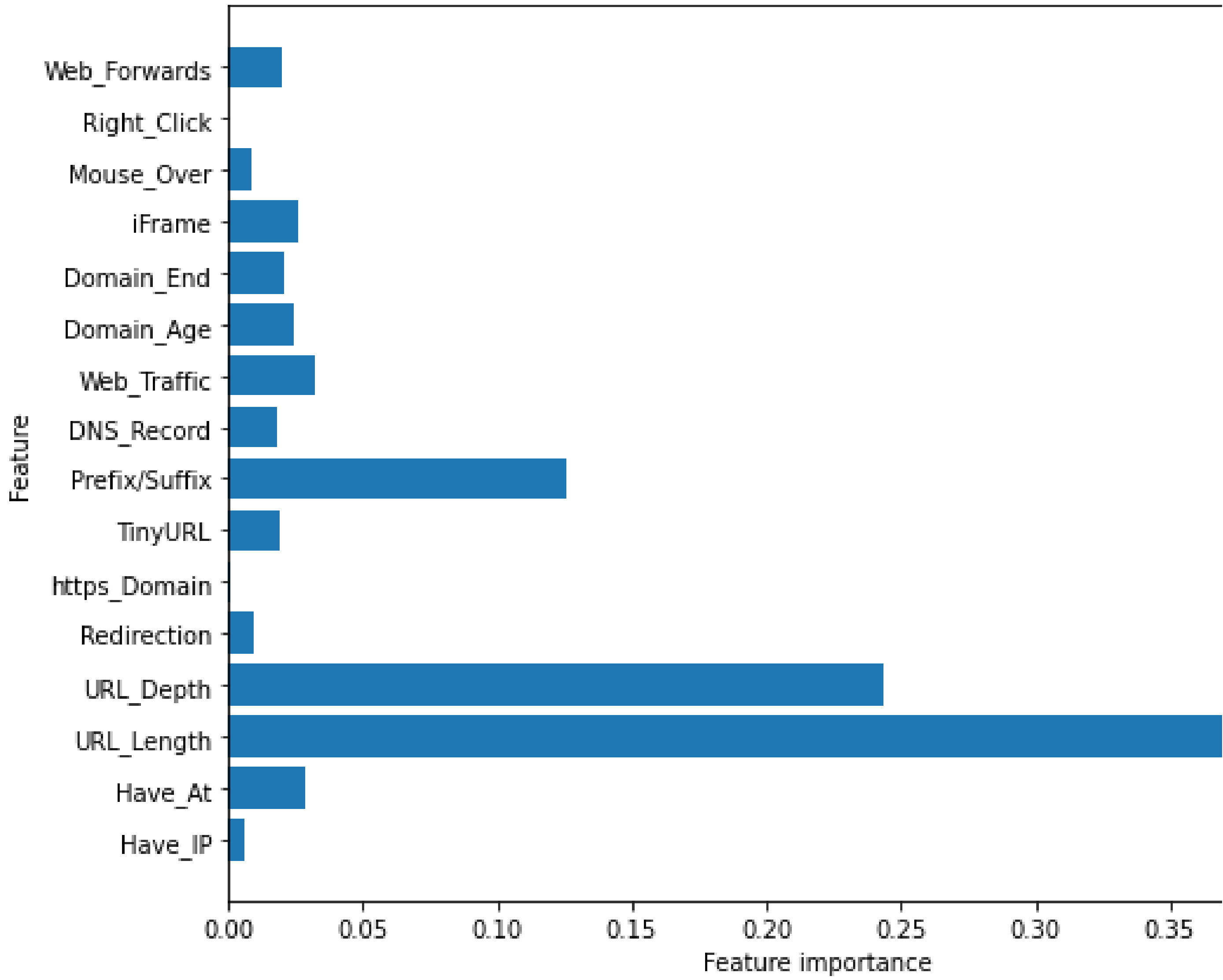
```
#computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data:
{:.3f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data:
{:.3f}".format(acc_test_forest))
```

```
Random forest: Accuracy on training Data: 0.814
Random forest: Accuracy on test Data: 0.834
```

In [0]:

```
#checking the feature improtance in the model
```

```
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

7.3. Multilayer Perceptrons (MLPs): Deep Learning

Multilayer perceptrons (MLPs) are also known as (vanilla) feed-forward neural networks, or sometimes just neural networks. Multilayer perceptrons can be applied for both classification and regression problems.

MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision.

In [0]:

```
# Multilayer Perceptrons model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))

# fit the model
mlp.fit(X_train, y_train)
```

Out[0]:

```
MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9
,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=[100, 100, 100], learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

In [0]:

```
#predicting the target value from the model for the samples
y_test_mlp = mlp.predict(X_test)
y_train_mlp = mlp.predict(X_train)
```

Performance Evaluation:

In [0]:

```
#computing the accuracy of the model performance
acc_train_mlp = accuracy_score(y_train,y_train_mlp)
acc_test_mlp = accuracy_score(y_test,y_test_mlp)

print("Multilayer Perceptrons: Accuracy on training Data:
{:.3f}".format(acc_train_mlp))
print("Multilayer Perceptrons: Accuracy on test Data:
{:.3f}".format(acc_test_mlp))
```

```
Multilayer Perceptrons: Accuracy on training Data: 0.859
Multilayer Perceptrons: Accuracy on test Data: 0.863
```

Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('Multilayer Perceptrons', acc_train_mlp, acc_test_mlp)
```

7.4. XGBoost Classifier

XGBoost is one of the most popular machine learning algorithms these days. XGBoost stands for eXtreme Gradient Boosting. Regardless of the type of prediction task at hand; regression or classification. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

In [0]:

```
#XGBoost Classification model
```

```
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)
```

Out[0]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.4, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [0]:

```
#predicting the target value from the model for the samples
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)
```

Performance Evaluation:

In [0]:

```
#computing the accuracy of the model performance
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)

print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
```

```
XGBoost: Accuracy on training Data: 0.866
XGBoost : Accuracy on test Data: 0.864
```

Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('XGBoost', acc_train_xgb, acc_test_xgb)
```

7.5. Autoencoder Neural Network

An auto encoder is a neural network that has the same number of input neurons as it does outputs. The hidden layers of the neural network will have fewer neurons than the input/output neurons. Because there are fewer neurons, the auto-encoder must learn to encode the input to the fewer hidden neurons. The predictors (x) and output (y) are exactly the same in an auto encoder.

In [0]:

```
#importing required packages
import keras
from keras.layers import Input, Dense
from keras import regularizers
import tensorflow as tf
from keras.models import Model
from sklearn import metrics
```

```
Using TensorFlow backend.
```

In [0]:

```
#building autoencoder model
```

```python
input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
                activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim), activation="relu")(encoder)

encoder = Dense(int(encoding_dim-2), activation="relu")(encoder)
code = Dense(int(encoding_dim-4), activation='relu')(encoder)
decoder = Dense(int(encoding_dim-2), activation='relu')(code)

decoder = Dense(int(encoding_dim), activation='relu')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)          (None, 16)                0
_________________________________________________________________
dense_1 (Dense)               (None, 16)                272
_________________________________________________________________
dense_2 (Dense)               (None, 16)                272
_________________________________________________________________
dense_3 (Dense)               (None, 14)                238
_________________________________________________________________
dense_6 (Dense)               (None, 16)                240
_________________________________________________________________
dense_7 (Dense)               (None, 16)                272
=================================================================
Total params: 1,294
Trainable params: 1,294
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```python
#compiling the model
autoencoder.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64,
shuffle=True, validation_split=0.2)
```

```
Train on 6400 samples, validate on 1600 samples
Epoch 1/10
6400/6400 [==============================] - 0s 51us/step - loss: 1.3997 -
accuracy: 0.7132 - val_loss: -0.3941 - val_accuracy: 0.7890
Epoch 2/10
6400/6400 [==============================] - 0s 24us/step - loss: -0.4269 -
accuracy: 0.7821 - val_loss: -0.5190 - val_accuracy: 0.7812
Epoch 3/10
6400/6400 [==============================] - 0s 24us/step - loss: -1.0514 -
accuracy: 0.7908 - val_loss: -1.3147 - val_accuracy: 0.8149
```

```
Epoch 4/10
6400/6400 [==============================] - 0s 24us/step - loss: -1.3118 -
accuracy: 0.8200 - val_loss: -1.3532 - val_accuracy: 0.8128
Epoch 5/10
6400/6400 [==============================] - 0s 25us/step - loss: -1.3789 -
accuracy: 0.8168 - val_loss: -1.4710 - val_accuracy: 0.8190
Epoch 6/10
6400/6400 [==============================] - 0s 25us/step - loss: -1.4435 -
accuracy: 0.8187 - val_loss: -1.5160 - val_accuracy: 0.8204
Epoch 7/10
6400/6400 [==============================] - 0s 25us/step - loss: -1.4951 -
accuracy: 0.8215 - val_loss: -1.5601 - val_accuracy: 0.8240
Epoch 8/10
6400/6400 [==============================] - 0s 23us/step - loss: -1.5208 -
accuracy: 0.8192 - val_loss: -1.5912 - val_accuracy: 0.8236
Epoch 9/10
6400/6400 [==============================] - 0s 25us/step - loss: -1.5044 -
accuracy: 0.8140 - val_loss: -1.5868 - val_accuracy: 0.8191
Epoch 10/10
6400/6400 [==============================] - 0s 25us/step - loss: -1.5554 -
accuracy: 0.8214 - val_loss: -1.6153 - val_accuracy: 0.8205
```

Performance Evaluation:

In [0]:

```
acc_train_auto = autoencoder.evaluate(X_train, X_train)[1]
acc_test_auto = autoencoder.evaluate(X_test, X_test)[1]

print('\nAutoencoder: Accuracy on training Data: {:.3f}'
.format(acc_train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}' .format(acc_test_auto))
```

```
8000/8000 [==============================] - 0s 18us/step
2000/2000 [==============================] - 0s 20us/step

Autoencoder: Accuracy on training Data: 0.819
Autoencoder: Accuracy on test Data: 0.818
```

Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('AutoEncoder', acc_train_auto, acc_test_auto)
```

7.6. Support Vector Machines

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

In [0]:

```
#Support vector machine model
from sklearn.svm import SVC

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)
```

```
#fit the model
svm.fit(X_train, y_train)
```

Out[0]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear'
,
    max_iter=-1, probability=False, random_state=12, shrinking=True, tol=0.
001,
    verbose=False)
```

In [0]:

```
#predicting the target value from the model for the samples
y_test_svm = svm.predict(X_test)
y_train_svm = svm.predict(X_train)
```

Performance Evaluation:

In [0]:

```
#computing the accuracy of the model performance
acc_train_svm = accuracy_score(y_train,y_train_svm)
acc_test_svm = accuracy_score(y_test,y_test_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))
```

```
SVM: Accuracy on training Data: 0.798
SVM : Accuracy on test Data: 0.818
```

Storing the results:

In [0]:

```
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('SVM', acc_train_svm, acc_test_svm)
```

8. Comparision of Models

To compare the models performance, a dataframe is created. The columns of this dataframe are the lists created to store the results of the model.

In [0]:

```
#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
    'Train Accuracy': acc_train,
    'Test Accuracy': acc_test})
results
```

Out[0]:

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.810	0.826
1	Random Forest	0.814	0.834

	ML Model	Train Accuracy	Test Accuracy
2	Multilayer Perceptrons	0.858	0.863
3	XGBoost	0.866	0.864
4	AutoEncoder	0.819	0.818
5	SVM	0.798	0.818

In [0]:

```
#Sorting the datafram on accuracy
results.sort_values(by=['Test Accuracy', 'Train Accuracy'],
ascending=False)
```

Out[0]:

	ML Model	Train Accuracy	Test Accuracy
3	XGBoost	0.866	0.864
2	Multilayer Perceptrons	0.858	0.863
1	Random Forest	0.814	0.834
0	Decision Tree	0.810	0.826
4	AutoEncoder	0.819	0.818
5	SVM	0.798	0.818

For the above comparision, it is clear that the XGBoost Classifier works well with this dataset.

So, saving the model for future use.

In [0]:

```
# save XGBoost model to file
import pickle
pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

Testing the saved model:

In [0]:

```
# load model from file
loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
loaded_model
```

Out[0]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
```

```
learning_rate=0.4, max_delta_step=0, max_depth=7,
min_child_weight=1, missing=nan, n_estimators=100, n_jobs=1,
nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=None, subsample=1, verbosity=1)
```

9. References

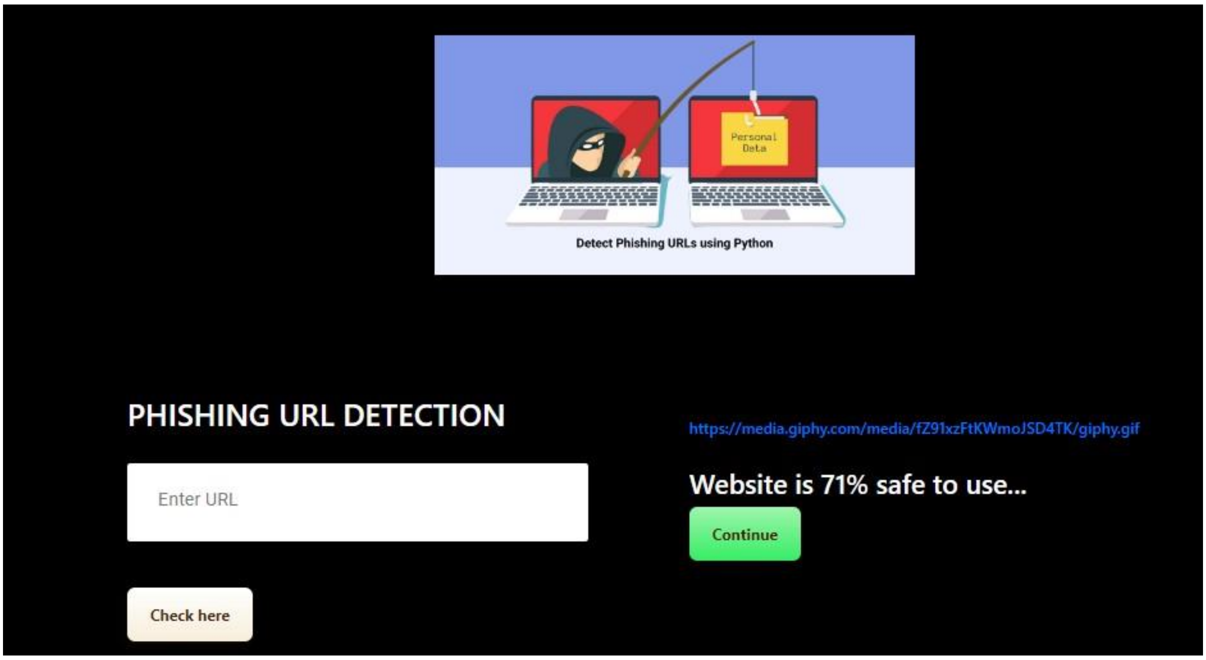
- https://blog.keras.io/building-autoencoders-in-keras.html
- https://en.wikipedia.org/wiki/Autoencoder
- https://mc.ai/a-beginners-guide-to-build-stacked-autoencoder-and-tying-weights-with-it/
- https://github.com/shreyagopal/t81_558_deep_learning/blob/master/t81_558_class_14_03_anomaly.ipynb
- https://machinelearningmastery.com/save-gradient-boosting-models-xgboost-python/

Testing: -

Phishing is a form of fraudulent attack where the attacker tries to gain sensitive information by posing as a reputable source. In a typical phishing attack, a victim opens a compromised link that poses as a credible website. The victim is then asked to enter their credentials, but since it is a "fake" website, the sensitive information is routed to the hacker and the victim gets "hacked."
Phishing is popular since it is a low effort, high reward attack. Most modern web browsers, antivirus software and email clients are pretty good at detecting phishing websites at the source, helping to prevent attacks. To understand how they work, this blog post will walk you through a tutorial that shows you how to build your own phishing URL detector using Python and machine learning:

1. **Identify the criteria** that can recognize fake URLs
2. **Build a decision tree** that can iterate through the criteria
3. **Train our model** to recognize fake vs real URLs
4. **Evaluate our model** to see how it performs
5. **Check for false positives/negatives**

(9) RESULTS:-



(10) Advantages:-

There is some Advantages of Web Phishing Detection

- Eliminate the cyber threat risk level.
- Measure the degrees of corporate and employee vulnerability.
- Increase user alertness to phishing risks.
- Install a cyber security culture and create cyber security heroes.

(10) Disadvantages:-

As increasingly-sophisticated phishing attacks, such as BEC, become more difficult to detect, even by trained security personnel. Thus there is an urgent need for the channel to provide customers with technology that not only strives to prevent intrusion, but can also help users after an attack has passed through the secure email gateway.

A mailbox-level anti-phishing solution offers an additional layer of protection by analyzing account information and understanding users' communication habits. This delivers an enhanced level of phishing protection to detect attacks faster, alert users and remediate threats as quickly as possible. Machine learning scores sender reputation enabling a baseline for what "normal communications" with a user should look like. It can then compare correspondence and incoming messages with multiple data points to identify and learn from anomalies.

(11) Conclusion:-

Given wide range of the researches carried out as well as different ways provided to detect phishing attacks, the existing techniques are still not able to precisely identify these attacks and in many cases provide no accurate results. In this paper, we proposed two feature sets to improve the performance of detecting phishing attacks and preventing data loss in internet banking webpages. Our proposed feature sets, determine the relationship between the content and the URL of a page.

(12) Future Scope:-

1) Phishing is a considerable problem differs from the other security threats such as intrusions and Malware which are based on the technical security holes of the network systems. The weakness point of any network system is its Users.
2) Phishing attacks are targeting these users depending on the trikes of social engineering.
3) Therefore, building a specific limited scope detection system will not provide complete protection from the wide phishing attack vectors.
4) Additionally, Anti-phishing solutions can be positioned at different levels of attack flow where most researchers are focusing on client side solutions which turn to add more processing overhead at the client side and lead to losing the trust and satisfaction of the users.

(13) Appendix:-

Source Code:-

IBM-EPBL/IBM-Project-22340-1659849590: Web Phishing Detection (github.com)

GitHub Link:-

IBM-EPBL/IBM-Project-22340-1659849590: Web Phishing Detection (github.com)