“DETECTING PHISHING WEBSITE USING MACHINE LEARNING”

Project submitted in partial fulfillment of the requirements for the

award of the Degree of

Bachelor of Computer Applications

of

SASTRA DEEMED UNIVERSITY

Submitted by

“S. SIDDHARTHAN”

Register No: 21113070562

Under the Supervision and Guidance of

Prof. L. GOWRI

School of Computing
SASTRA Deemed University
(Under section 3 of the UGC Act, 1956)
Thanjavur – 613 401
January – 2024

School of Computing
SASTRA University
(Under section 3 of the UGC Act, 1956)
Thanjavur – 613 401

<u>Bonafide Certificate</u>

Certified that this project report entitled

"DETECTING PHISHING WEBSTE USING MACHINE LEARNING"

Is a bonafide record of work done by

"S.SIDDHARTHAN"
Register No. 21113070562

In partial fulfillment of the requirements for award of the
Degree of **Bachelor of Computer Applications**
During the year 2021 -2024

Project Guide: Prof. L. Gowri

Submitted for Project viva-voce examination held on --------

Examiner -I **Examiner-II**

Acknowledgements

I would like to thank our Honorable Chancellor Prof.R.Sethuraman for providing me with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

I would like to thank our Honorable Vice-Chancellor Dr.S.Vaidhyasubramaniam and Dr.S.Swaminathan, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

I extend my sincere thanks to Dr.R.Chandramouli, Registrar, SASTRA Deemed to be University forproviding the opportunity to pursue this project.

I extend my heartfelt thanks to Dr.V.S.Shankar Sriram, Dean, School of Computing, Dr.R.Muthaiah,Associate Dean-Research, School of Computing Dr.K.Ramkumar, Associate Dean-Academics, School of Computing Dr.D.Manivannan, Associate Dean-Infrastructure School of Computing and Dr.R.Alageswaran, Associate Dean-Student Welfare, School of Computing for their motivation and support offered in materializing this project.

My guide Prof. L. Gowri, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped me in making progress throughout our project work. I also thank the project review panel members for their valuable comments and insights which made this project better.

I would like to extend my gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped me in the completion of the project.

I gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. I thank you all for providing me an opportunity to showcase my skills through project.

ABSTRACT

This research delves into the realm of machine learning methodologies applied to the detection of phishing websites through an in-depth analysis of URL features. The primary focus lies in the meticulous examination and discrimination between legitimate and phishing sites, utilizing advanced techniques to bolster the accuracy of detection mechanisms. The evaluation encompasses the efficacy of decision trees, random forests, and deep learning algorithms, aiming to identify the most suitable approach for precise and reliable detection of phishing activities. By undertaking this investigation, the study significantly contributes to cybersecurity measures by enhancing the capabilities of phishing detection systems, thereby fortifying online security protocols and providing a robust defense against evolving cyber threats.

Table of Contents

Introduction

The escalating threat of phishing websites poses an ever-growing risk to the landscape of online security, necessitating robust and proactive measures to counteract the evolving tactics employed by malicious entities. Recognizing the pivotal role of machine learning algorithms in fortifying cybersecurity, this study is dedicated to exploring their indispensable application for the proactive identification and mitigation of phishing sites. The overarching objective is to implement advanced measures that leverage the power of machine learning for real-time detection, thereby significantly reducing the risks posed by phishing websites. Through a focused examination of cutting-edge technologies, the research aims to contribute to the development of a resilient defense mechanism that protects users from the constantly changing and increasingly sophisticated landscape of cyber threats.

Problem Statement

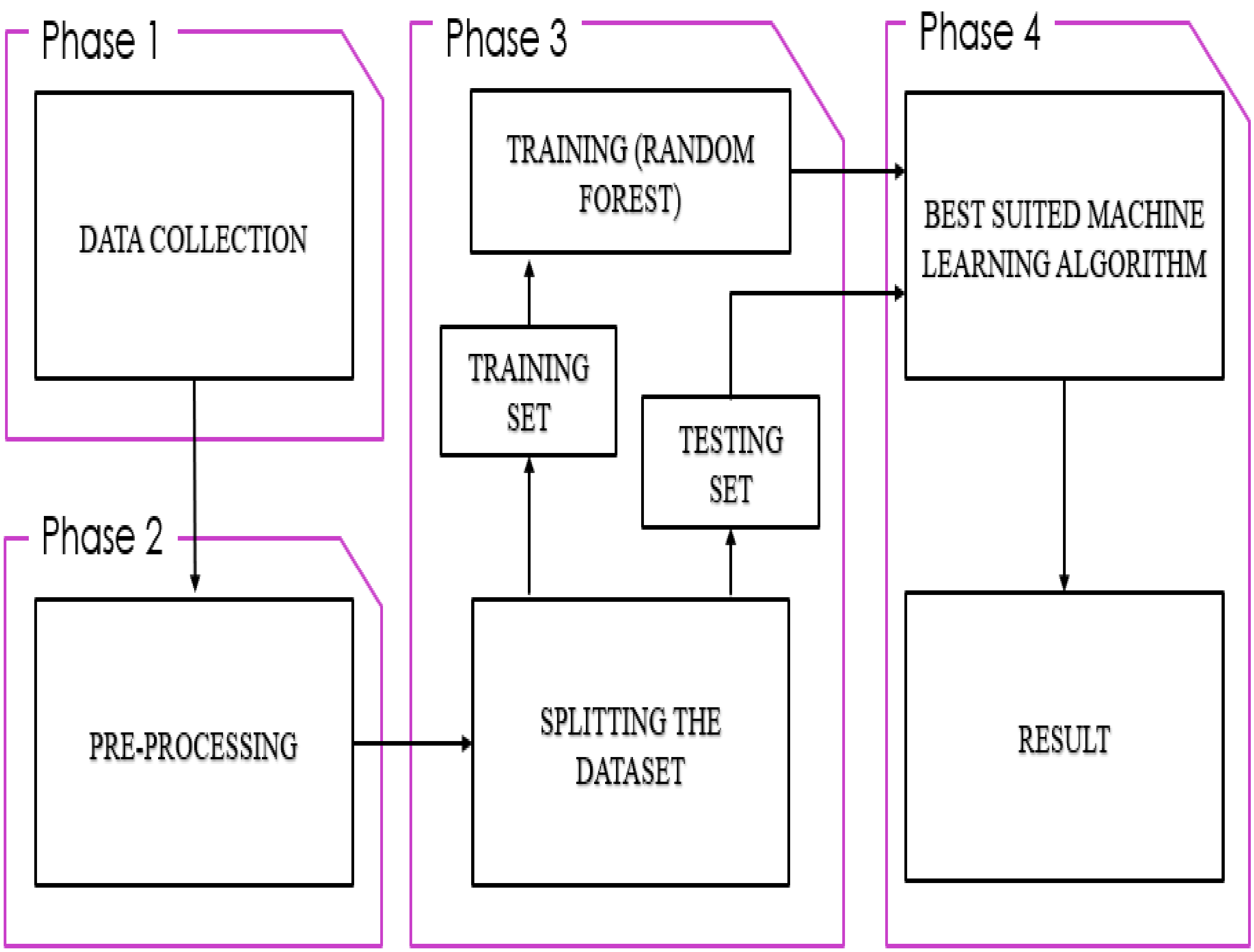
The relentless growth of phishing threats has surpassed the efficacy of traditional identification methods, demanding an urgent and proactive response to safeguard online security. In this context, the research takes a pioneering approach by focusing on the strategic utilization of machine learning algorithms for a meticulous and robust analysis of URL features. The primary goal is to develop an advanced model that goes beyond merely identifying phishing threats but excels in effectively distinguishing between legitimate and malicious websites. By integrating state-of-the-art methodologies, the study aims to address the dynamic nature of phishing tactics, providing a resilient and adaptive solution that significantly elevates cybersecurity standards. Furthermore, the research emphasizes the importance of real-time detection capabilities, aiming to create a system that not only reacts to known threats but also proactively identifies emerging phishing techniques. This multifaceted approach seeks to contribute to the ongoing evolution of cybersecurity practices, offering a comprehensive defense against the multifarious and sophisticated challenges posed by phishing threats in the contemporary digital landscape.

Objective

This research is centered around the comprehensive development and assessment of machine learning algorithms, specifically decision trees, random forests, and deep learning, with a targeted application in proactive phishing detection. The primary objective is to thoroughly evaluate the efficacy of these models in distinguishing between legitimate and malicious websites. The overarching goal is to significantly enhance online security, going beyond immediate applications to contribute valuable insights to the continually evolving field of cybersecurity. By delving into the nuances of machine learning algorithms, this study aims to advance the current state-of-the-art in proactive phishing detection, fostering a deeper understanding of the complex dynamics involved in discerning authentic from fraudulent online entities. The research aspires to fortify online security practices, ensuring a resilient defense against the ever-expanding landscape of phishing threats, and to contribute knowledge that informs effective cybersecurity strategies in the face of dynamic tactics employed by cyber adversaries.

2. SYSTEM DESIGN

Architecture Diagram



3. SYSTEM REQUIREMENT

3.1 HARDWARE REQUIREMENTS:

- System: Intel Pentium IV 2.80 GHz.
- Monitor: LED.
- Mouse: Logitech.
- Ram: 4.00 GB or above 4.00 GB
- Hard Disk: 250 GB

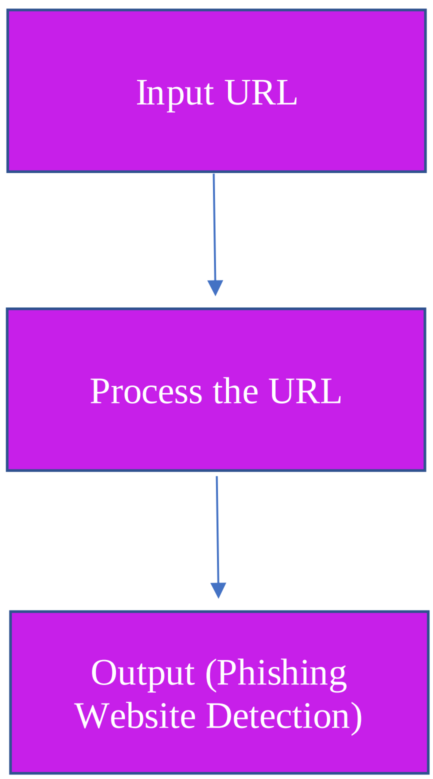
3.2 SOFTWARE REQUIREMENTS:

- Operating system: Windows 7, Ubuntu
- Language: Python 3

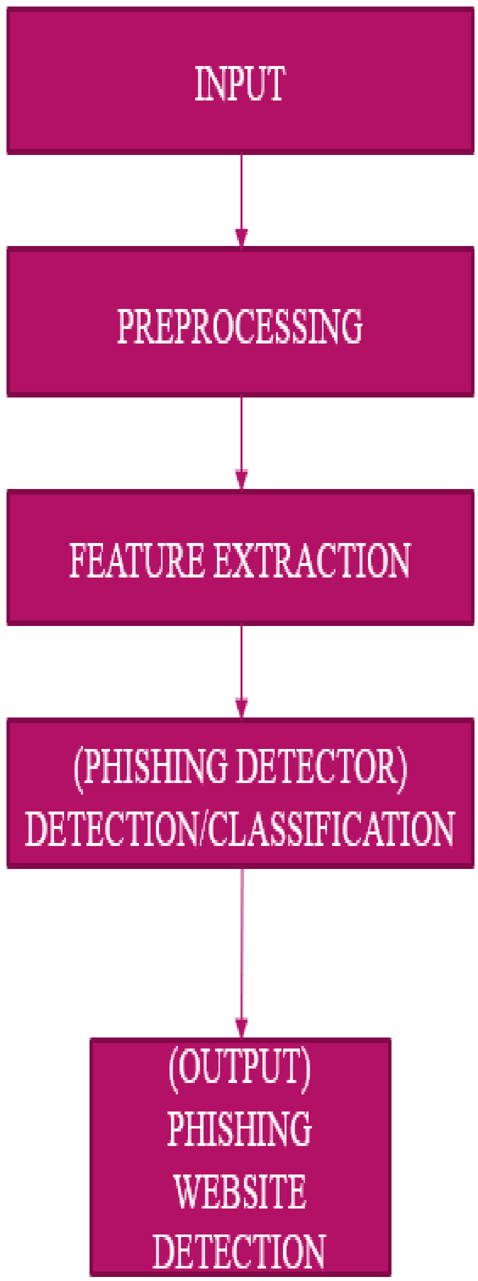
4. SYSTEM ANALYSIS

DATA FLOW DIAGRAM

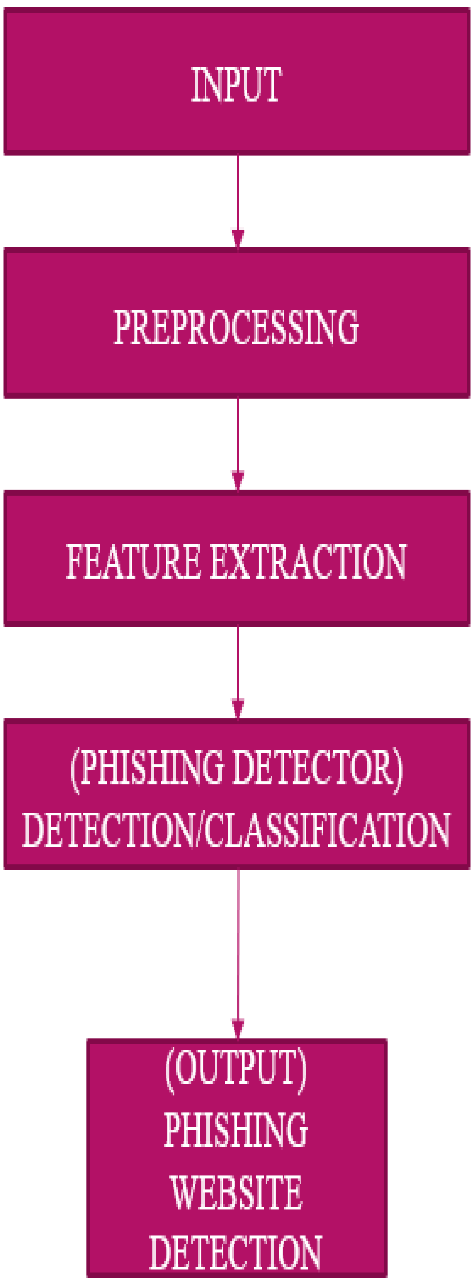
Level 0:



Level 1:


INPUT
PREPROCESSING
FEATURE EXTRACTION
(PHISHING DETECTOR)
DETECTION/CLASSIFICATION
(OUTPUT)
PHISHING
WEBSITE
DETECTION

LEVEL 2:



Input: Users enter URLs to check for phishing.

Data Preprocessing: Clean and remove irrelevant data from user input.

Feature Extraction: Extract relevant URL features like domain, IP, and SSL.

Phishing Detector: Check the URL using features to detect phishing, using algorithms like decision trees and deep learning. It determine if a website is phishing based on the model's output.

Output: Provide users with the final indication of whether the website is phishing or not.

5. APPROACH

Below mentioned are the steps involved in the completion of this project:

• Collect dataset containing phishing and legitimate websites from the open-source platforms.

• Write a code to extract the required features from the URL database.

• Analyze and preprocess the dataset by using EDA techniques.

• Divide the dataset into training and testing sets.

• Run selected machine learning and deep neural network algorithms like SVM, Random Forest, Autoencoder on the dataset. • Write a code for displaying the evaluation result considering accuracy metrics.

• Compare the obtained results for trained models and specify which is better.

6. CODING

6.1. Feature Extraction

1. Data Collection:

For this project, we need a bunch of URLs of type legitimate (0) and phishing (1).

The collection of phishing URLs is rather easy because of the opensource service called Phish Tank. This service provides a set of phishing URLs in multiple formats like csv, json etc. that gets updated hourly. To download the data: https://www.phishtank.com/developer_info.php

For the legitimate URLs, I found a source that has a collection of benign, spam, phishing, malware & defacement URLs. The source of the dataset is University of New Brunswick, https://www.unb.ca/cic/datasets/url-2016.html. The number of legitimate URLs in this collection are 35,300. The URL collection is downloaded & from that, 'Benign_list_big_final.csv' is the file of our interest. This file is then uploaded to the Colab for the feature extraction.

First Step is importing pandas.

1. Objective:

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. The objective of this notebook is to collect data & extract the selctive features form the URLs.

2. Collecting the Data:

For this project, we need a bunch of urls of type legitimate (0) and phishing (1).

The collection of phishing urls is rather easy because of the opensource service called PhishTank. This service provide a set of phishing URLs in multiple formats like csv, json etc. that gets updated hourly. To download the data: https://www.phishtank.com/developer_info.php

For the legitimate URLs, I found a source that has a collection of benign, spam, phishing, malware & defacement URLs. The source of the dataset is University of New Brunswick, https://www.unb.ca/cic/datasets/url-2016.html. The number of legitimate URLs in this collection are 35,300. The URL collection is downloaded & from that, *'Benign_list_big_final.csv'* is the file of our interest. This file is then uploaded to the Colab for the feature extraction.

2.1. Phishing URLs:

The phishing URLs are collected from the PhishTank from the link provided. The csv file of phishing URLs is obtained by using wget command. After downlaoding the dataset, it is loaded into a DataFrame.

In [0]:
```
#importing required packages for this module
import pandas as pd
```

Now downloading the phishing websites.

In [0]:
```
#Downloading the phishing URLs file
!wget http://data.phishtank.com/data/online-valid.csv
```

```
--2020-05-10 07:33:37--  http://data.phishtank.com/data/online-valid.csv
Resolving data.phishtank.com (data.phishtank.com)... 104.16.101.75, 104.17.177.85, 2606:4700::6810:654b, ...
Connecting to data.phishtank.com (data.phishtank.com)|104.16.101.75|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://data.phishtank.com/data/online-valid.csv [following]
--2020-05-10 07:33:37--  https://data.phishtank.com/data/online-valid.csv
Connecting to data.phishtank.com (data.phishtank.com)|104.16.101.75|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://d1750zhbc38ec0.cloudfront.net/datadumps/verified_online.csv?Expires=1589096027&Signature=NeznemrBS2h3ozoDsM8x9fZ73pTe1OhCjCyYEEtKcqjyJlO62TdCD9eAh4tC0fvlytZAq4ihqhtRGtgk
waWfw6QJE8HhE-UfnzUlOxU6w-lnHJppNbsbWsIqCjYeBoNbGvLTpa4CklK5Lo7PV6vd3bSl8wAq0PNjyct7f6qyO2nazZilc0NIdzHp2t-XwAozQj39S7czLORAzloGH98cqa1XBc3honvarNeV3S6d8QJCO8dHf3zk201KUSJFRIky6sFZP3--z5a
DSL06fZj-yAyIDE-Xn0SNaiqLFVuMQUx0tTo5eIdk98zC2D7R5XOvAkGdpo1fGHT45f77MzUv4Q__&Key-Pair-Id=APKAILB45UG3RB4CSOJA [following]
--2020-05-10 07:33:37--  https://d1750zhbc38ec0.cloudfront.net/datadumps/verified_online.csv?Expires=1589096027&Signature=NeznemrBS2h3ozoDsM8x9fZ73pTe1OhCjCyYEEtKcqjyJlO62TdCD9eAh4tC0fvly
tZAq4ihqhtRGtgkwaWfw6QJE8HhE-UfnzUlOxU6w-lnHJppNbsbWsIqCjYeBoNbGvLTpa4CklK5Lo7PV6vd3bSl8wAq0PNjyct7f6qyO2nazZilc0NIdzHp2t-XwAozQj39S7czLORAzloGH98cqa1XBc3honvarNeV3S6d8QJCO8dHf3zk201KUSJF
RIky6sFZP3--z5aDSL06fZj-yAyIDE-Xn0SNaiqLFVuMQUx0tTo5eIdk98zC2D7R5XOvAkGdpo1fGHT45f77MzUv4Q__&Key-Pair-Id=APKAILB45UG3RB4CSOJA
Resolving d1750zhbc38ec0.cloudfront.net (d1750zhbc38ec0.cloudfront.net)... 143.204.101.142, 143.204.101.147, 143.204.101.48, ...
Connecting to d1750zhbc38ec0.cloudfront.net (d1750zhbc38ec0.cloudfront.net)|143.204.101.142|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3232768 (3.1M) [text/csv]
Saving to: 'online-valid.csv'

online-valid.csv    100%[===================>]   3.08M  5.13MB/s    in 0.6s

2020-05-10 07:33:38 (5.13 MB/s) - 'online-valid.csv' saved [3232768/3232768]
```

The above command downlaods the file of phishing URLs, *online-valid.csv* and stores in the */content/* folder.

2. Samples

Taking 5000 sample phishing website

In [0]:
```
#loading the phishing URLs data to dataframe
data0 = pd.read_csv("online-valid.csv")
data0.head()
```

Out[0]:

	phish_id	url	phish_detail_url	submission_time	verified	verification_time	online	target
0	6557033	http://u1047531.cp.regruhosting.ru/acces-inges...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T22:01:43+00:00	yes	2020-05-09T22:03:07+00:00	yes	Other
1	6557032	http://hoysalacreations.com/wp-content/plugins...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T22:01:37+00:00	yes	2020-05-09T22:03:07+00:00	yes	Other
2	6557011	http://www.accsystemprblemhelp.site/checkpoint...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:54:31+00:00	yes	2020-05-09T21:55:38+00:00	yes	Facebook
3	6557010	http://www.accsystemprblemhelp.site/login_atte...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:53:48+00:00	yes	2020-05-09T21:54:34+00:00	yes	Facebook
4	6557009	https://firebasestorage.googleapis.com/v0/b/so...	http://www.phishtank.com/phish_detail.php?phis...	2020-05-09T21:49:27+00:00	yes	2020-05-09T21:51:24+00:00	yes	Microsoft

In [0]:
```
data0.shape
```

Out[0]:
```
(14858, 8)
```

So, the data has thousands of phishing URLs. But the problem here is, this data gets updated hourly. Without getting into the risk of data imbalance, I am considering a margin value of 10,000 phishing URLs & 5000 legitimate URLs.

Thereby, picking up 5000 samples from the above dataframe randomly.

In [0]:
```
#Collecting 5,000 Phishing URLs randomly
phishurl = data0.sample(n = 5000, random_state = 12).copy()
phishurl = phishurl.reset_index(drop=True)
phishurl.head()
```

Out[0]:

	phish_id	url	phish_detail_url	submission_time	verified	verification_time	online	target
0	6485787	https://eevee.tv/Bootstrap/assets/css/acces	http://www.phishtank.com/phish_detail.php?phis...	2020-04-04T03:01:00+00:00	yes	2020-04-04T03:03:56+00:00	yes	Other
1	6422543	https://appleid.apple.com-sa.pm/appleid/?	http://www.phishtank.com/phish_detail.php?phis...	2020-02-27T17:01:01+00:00	yes	2020-03-17T01:50:51+00:00	yes	Other
2	6543602	https://grandcup.xyz/	http://www.phishtank.com/phish_detail.php?phis...	2020-05-02T23:07:29+00:00	yes	2020-05-02T23:09:03+00:00	yes	Steam
3	6528783	https://villa-azzurro.com/onedrive/	http://www.phishtank.com/phish_detail.php?phis...	2020-04-25T20:54:02+00:00	yes	2020-04-25T21:46:55+00:00	yes	Other
4	6498136	http://mygpstrip.net/ii/u.php	http://www.phishtank.com/phish_detail.php?phis...	2020-04-10T15:01:56+00:00	yes	2020-04-10T16:01:37+00:00	yes	Other

Uploading the Legitimate URL excel file which is got from https://www.unb.ca/cic/datasets/url-2016.html

In [0]:
```
phishurl.shape
```

Out[0]:
```
(5000, 8)
```

As of now we collected 5000 phishing URLs. Now, we need to collect the legitimate URLs.

2.2. Legitimate URLs:

From the uploaded *Benign_list_big_final.csv* file, the URLs are loaded into a dataframe.

In [0]:
```
#Loading legitimate files
data1 = pd.read_csv("Benign_list_big_final.csv")
data1.columns = ['URLs']
data1.head()
```

Out[0]:

	URLs
0	http://1337x.to/torrent/1110018/Blackhat-2015-...
1	http://1337x.to/torrent/1122940/Blackhat-2015-...
2	http://1337x.to/torrent/1124395/Fast-and-Furio...
3	http://1337x.to/torrent/1145504/Avengers-Age-o...
4	http://1337x.to/torrent/1160078/Avengers-age-o...

As stated above, 5000 legitimate URLs are randomaly picked from the above dataframe.

Taking 5000 Samples of Legitimate website

In [0]:
```
#Collecting 5,000 Legitimate URLs randomly
legiurl = data1.sample(n = 5000, random_state = 12).copy()
legiurl = legiurl.reset_index(drop=True)
legiurl.head()
```

Out[0]:

	URLs
0	http://graphicriver.net/search?date=this-month...
1	http://ecnavi.jp/redirect/?url=http://www.cros...
2	https://hubpages.com/signin?explain=follow+Hub...
3	http://extratorrent.cc/torrent/4190536/AOMEI+B...
4	http://icicibank.com/Personal-Banking/offers/o...

In [0]:
```
legiurl.shape
```

Out[0]:
```
(5000, 1)
```

3. Feature Extraction:

In this step, features are extracted from the URLs dataset.

The extracted features are categorized into

1. Address Bar based Features
2. Domain based Features
3. HTML & Javascript based Features

3.1. Address Bar Based Features:

Many features can be extracted that can be considered as address bar base features. Out of them, below mentioned were considered for this project.

- Domain of URL
- IP Address in URL
- "@" Symbol in URL
- Length of URL
- Depth of URL
- Redirection "//" in URL
- "http/https" in Domain name
- Using URL Shortening Services “TinyURL”
- Prefix or Suffix "-" in Domain

Each of these features are explained and the coded below:

```
In [0]: # importing required packages for this section
from urllib.parse import urlparse,urlencode
import ipaddress
import re
```

3.1.1. Domain of the URL

Here, we are just extracting the domain present in the URL. This feature doesn't have much significance in the training. May even be dropped while training the model.

```
In [0]: # 1.Domain of the URL (Domain)
def getDomain(url):
  domain = urlparse(url).netloc
  if re.match(r"^www.",domain):
             domain = domain.replace("www.","")
  return domain
```

3.1.2. IP Address in the URL

Checks for the presence of IP address in the URL. URLs may have IP address instead of domain name. If an IP address is used as an alternative of the domain name in the URL, we can be sure that someone is trying to steal personal information with this URL.

If the domain part of URL has IP address, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

```
In [0]: # 2.Checks for IP address in URL (Have_IP)
def havingIP(url):
  try:
    ipaddress.ip_address(url)
    ip = 1
  except:
    ip = 0
  return ip
```

3.1.3. "@" Symbol in URL

Checks for the presence of '@' symbol in the URL. Using "@" symbol in the URL leads the browser to ignore everything preceding the "@" symbol and the real address often follows the "@" symbol.

If the URL has '@' symbol, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

```
In [0]: # 3.Checks the presence of @ in URL (Have_At)
def haveAtSign(url):
  if "@" in url:
    at = 1
  else:
    at = 0
  return at
```

3.1.4. Length of URL

Computes the length of the URL. Phishers can use long URL to hide the doubtful part in the address bar. In this project, if the length of the URL is greater than or equal 54 characters then the URL classified as phishing otherwise legitimate.

If the length of URL >= 54 , the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

```
In [0]: # 4.Finding the length of URL and categorizing (URL_Length)
def getLength(url):
  if len(url) < 54:
    length = 0
  else:
    length = 1
  return length
```

3.1.5. Depth of URL

Computes the depth of the URL. This feature calculates the number of sub pages in the given url based on the '/'.

The value of feature is a numerical based on the URL.

In [0]:
```
# 5.Gives number of '/' in URL (URL_Depth)
def getDepth(url):
  s = urlparse(url).path.split('/')
  depth = 0
  for j in range(len(s)):
    if len(s[j]) != 0:
      depth = depth+1
  return depth
```

3.1.6. Redirection "//" in URL

Checks the presence of "//" in the URL. The existence of "//" within the URL path means that the user will be redirected to another website. The location of the "//" in URL is computed. We find that if the URL starts with "HTTP", that means the "//" should appear in the sixth position. However, if the URL employs "HTTPS" then the "//" should appear in seventh position.

If the "//" is anywhere in the URL apart from after the protocal, thee value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 6.Checking for redirection '//' in the url (Redirection)
def redirection(url):
  pos = url.rfind('//')
  if pos > 6:
    if pos > 7:
      return 1
    else:
      return 0
  else:
    return 0
```

3.1.7. "http/https" in Domain name

Checks for the presence of "http/https" in the domain part of the URL. The phishers may add the "HTTPS" token to the domain part of a URL in order to trick users.

If the URL has "http/https" in the domain part, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 7.Existence of "HTTPS" Token in the Domain Part of the URL (https_Domain)
def httpDomain(url):
  domain = urlparse(url).netloc
  if 'https' in domain:
    return 1
  else:
    return 0
```

3.1.8. Using URL Shortening Services "TinyURL"

URL shortening is a method on the "World Wide Web" in which a URL may be made considerably smaller in length and still lead to the required webpage. This is accomplished by means of an "HTTP Redirect" on a domain name that is short, which links to the webpage that has a long URL.

If the URL is using Shortening Services, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
#listing shortening services
shortening_services = r"bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|" \
                      r"yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|" \
                      r"short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|" \
                      r"doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|db\.tt|" \
                      r"qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|q\.gs|is\.gd|" \
                      r"po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|x\.co|" \
                      r"prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|" \
                      r"tr\.im|link\.zip\.net"
```

In [0]:
```
# 8. Checking for Shortening Services in URL (Tiny_URL)
def tinyURL(url):
    match=re.search(shortening_services,url)
    if match:
        return 1
    else:
        return 0
```

3.1.9. Prefix or Suffix "-" in Domain

Checking the presence of '-' in the domain part of URL. The dash symbol is rarely used in legitimate URLs. Phishers tend to add prefixes or suffixes separated by (-) to the domain name so that users feel that they are dealing with a legitimate webpage.

If the URL has '-' symbol in the domain part of the URL, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

```
In [0]:
# 9.Checking for Prefix or Suffix Separated by (-) in the Domain (Prefix/Suffix)
def prefixSuffix(url):
    if '-' in urlparse(url).netloc:
        return 1            # phishing
    else:
        return 0            # legitimate
```

3.2. Domain Based Features:

Many features can be extracted that come under this category. Out of them, below mentioned were considered for this project.

- DNS Record
- Website Traffic
- Age of Domain
- End Period of Domain

Installing & importing required data

3.2. Domain Based Features:

Many features can be extracted that come under this category. Out of them, below mentioned were considered for this project.

- DNS Record
- Website Traffic
- Age of Domain
- End Period of Domain

Each of these features are explained and the coded below:

```
In [0]:
!pip install python-whois
```

```
Collecting python-whois
  Downloading https://files.pythonhosted.org/packages/f0/ab/11c2d01db2554bbaabb2c32b06b6a73f7277372533484c320c78a304dfd7/python-whois-0.7.2.tar.gz (90kB)
     |████████████████████████████████| 92kB 5.5MB/s
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from python-whois) (0.16.0)
Building wheels for collected packages: python-whois
  Building wheel for python-whois (setup.py) ... done
  Created wheel for python-whois: filename=python_whois-0.7.2-cp36-none-any.whl size=85245 sha256=900afbc18f144913762a57978778098dda65b687b3b5a1f14f7998e9631564e8
  Stored in directory: /root/.cache/pip/wheels/69/e6/62/1e6a746ca8e690f472611511b6948c325b232aaf693245ce46
Successfully built python-whois
Installing collected packages: python-whois
Successfully installed python-whois-0.7.2
```

```
In [0]:
# importing required packages for this section
import re
from bs4 import BeautifulSoup
import whois
import urllib
import urllib.request
from datetime import datetime
```

Each of these features are explained and the coded below:

3.2.1. DNS Record

For phishing websites, either the claimed identity is not recognized by the WHOIS database or no records founded for the hostname. If the DNS record is empty or not found then, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 11.DNS Record availability (DNS_Record)
# obtained in the featureExtraction function itself
```

3.2.2. Web Traffic

This feature measures the popularity of the website by determining the number of visitors and the number of pages they visit. However, since phishing websites live for a short period of time, they may not be recognized by the Alexa database (Alexa the Web Information Company., 1996). By reviewing our dataset, we find that in worst scenarios, legitimate websites ranked among the top 100,000. Furthermore, if the domain has no traffic or is not recognized by the Alexa database, it is classified as "Phishing".

If the rank of the domain < 100000, the vlaue of this feature is 1 (phishing) else 0 (legitimate).

In [0]:
```
# 12.Web traffic (Web_Traffic)
def web_traffic(url):
  try:
    #Filling the whitespaces in the URL if any
    url = urllib.parse.quote(url)
    rank = BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" + url).read(), "xml").find(
        "REACH")['RANK']
    rank = int(rank)
  except TypeError:
        return 1
  if rank <100000:
    return 1
  else:
    return 0
```

3.2.3. Age of Domain

This feature can be extracted from WHOIS database. Most phishing websites live for a short period of time. The minimum age of the legitimate domain is considered to be 12 months for this project. Age here is nothing but different between creation and expiration time.

If age of domain > 12 months, the vlaue of this feature is 1 (phishing) else 0 (legitimate).

In [0]:
```
# 13.Survival time of domain: The difference between termination time and creation time (Domain_Age)
def domainAge(domain_name):
  creation_date = domain_name.creation_date
  expiration_date = domain_name.expiration_date
  if (isinstance(creation_date,str) or isinstance(expiration_date,str)):
    try:
      creation_date = datetime.strptime(creation_date,'%Y-%m-%d')
      expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
    except:
      return 1
  if ((expiration_date is None) or (creation_date is None)):
      return 1
  elif ((type(expiration_date) is list) or (type(creation_date) is list)):
      return 1
  else:
    ageofdomain = abs((expiration_date - creation_date).days)
    if ((ageofdomain/30) < 6):
      age = 1
    else:
      age = 0
  return age
```

3.2.4. End Period of Domain

This feature can be extracted from WHOIS database. For this feature, the remaining domain time is calculated by finding the different between expiration time & current time. The end period considered for the legitimate domain is 6 months or less for this project.

If end period of domain > 6 months, the vlaue of this feature is 1 (phishing) else 0 (legitimate).

In [0]:
```
# 14.End time of domain: The difference between termination time and current time (Domain_End)
def domainEnd(domain_name):
  expiration_date = domain_name.expiration_date
  if isinstance(expiration_date,str):
    try:
      expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
    except:
      return 1
  if (expiration_date is None):
      return 1
  elif (type(expiration_date) is list):
      return 1
  else:
    today = datetime.now()
    end = abs((expiration_date - today).days)
    if ((end/30) < 6):
      end = 0
    else:
      end = 1
  return end
```

3.3. HTML and JavaScript based Features

Many features can be extracted that come under this category. Out of them, below mentioned were considered for this project.

- IFrame Redirection
- Status Bar Customization
- Disabling Right Click
- Website Forwarding

Each of these features are explained and the coded below:

In [0]:
```
# importing required packages for this section
import requests
```

3.3.1. IFrame Redirection

IFrame is an HTML tag used to display an additional webpage into one that is currently shown. Phishers can make use of the “iframe” tag and make it invisible i.e. without frame borders. In this regard, phishers make use of the “frameBorder” attribute which causes the browser to render a visual delineation.

If the iframe is empty or repsonse is not found then, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 15. IFrame Redirection (iFrame)
def iframe(response):
  if response == "":
      return 1
  else:
      if re.findall(r"[<iframe>|<frameBorder>]", response.text):
          return 0
      else:
          return 1
```

3.3.2. Status Bar Customization

Phishers may use JavaScript to show a fake URL in the status bar to users. To extract this feature, we must dig-out the webpage source code, particularly the "onMouseOver" event, and check if it makes any changes on the status bar

If the response is empty or onmouseover is found then, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 16.Checks the effect of mouse over on status bar (Mouse_Over)
def mouseOver(response):
  if response == "" :
    return 1
  else:
    if re.findall("<script>.+onmouseover.+</script>", response.text):
      return 1
    else:
      return 0
```

3.3.3. Disabling Right Click

Phishers use JavaScript to disable the right-click function, so that users cannot view and save the webpage source code. This feature is treated exactly as "Using onMouseOver to hide the Link". Nonetheless, for this feature, we will search for event "event.button==2" in the webpage source code and check if the right click is disabled.

If the response is empty or onmouseover is not found then, the value assigned to this feature is 1 (phishing) or else 0 (legitimate).

In [0]:
```
# 17.Checks the status of the right click attribute (Right_Click)
def rightClick(response):
  if response == "":
    return 1
  else:
    if re.findall(r"event.button ?== ?2", response.text):
      return 0
    else:
      return 1
```

3.3.4. Website Forwarding

The fine line that distinguishes phishing websites from legitimate ones is how many times a website has been redirected. In our dataset, we find that legitimate websites have been redirected one time max. On the other hand, phishing websites containing this feature have been redirected at least 4 times.

In [0]:
```
# 18.Checks the number of forwardings (Web_Forwards)
def forwarding(response):
  if response == "":
    return 1
  else:
    if len(response.history) <= 2:
      return 0
    else:
      return 1
```

Computing URL Features

Create a list and a function that calls the other functions and stores all the features of the URL in the list. We will extract the features of each URL and append to this list.

In [0]:
```
#Function to extract features
def featureExtraction(url,label):

  features = []
  #Address bar based features (10)
  features.append(getDomain(url))
  features.append(havingIP(url))
  features.append(haveAtSign(url))
  features.append(getLength(url))
  features.append(getDepth(url))
  features.append(redirection(url))
  features.append(httpDomain(url))
  features.append(tinyURL(url))
  features.append(prefixSuffix(url))

  #Domain based features (4)
  dns = 0
  try:
    domain_name = whois.whois(urlparse(url).netloc)
  except:
    dns = 1

  features.append(dns)
  features.append(web_traffic(url))
  features.append(1 if dns == 1 else domainAge(domain_name))
  features.append(1 if dns == 1 else domainEnd(domain_name))

  # HTML & Javascript based features (4)
  try:
    response = requests.get(url)
  except:
    response = ""
  features.append(iframe(response))
  features.append(mouseOver(response))
  features.append(rightClick(response))
  features.append(forwarding(response))
  features.append(label)

  return features
```

Legitimate URLs:

Now, feature extraction is done on legitimate URLs.

In [0]:
```
legiurl.shape
```

Out[0]:
```
(5000, 1)
```

In [0]:
```
#Extracting the feautres & storing them in a list
legi_features = []
label = 0

for i in range(0, 5000):
  url = legiurl['URLs'][i]
  legi_features.append(featureExtraction(url,label))
```

In [0]:
```
#converting the list to dataframe
feature_names = ['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']

legitimate = pd.DataFrame(legi_features, columns= feature_names)
legitimate.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	Lab
0	graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	

In [0]:
```
# Storing the extracted legitimate URLs fatures to csv file
legitimate.to_csv('legitimate.csv', index= False)
```

Phishing URLs:

Now, feature extraction is performed on phishing URLs.

In [0]:
```
phishurl.shape
```

Out[0]:
```
(5000, 8)
```

In [0]:
```
#Extracting the feautres & storing them in a list
phish_features = []
label = 1
for i in range(0, 5000):
  url = phishurl['url'][i]
  phish_features.append(featureExtraction(url,label))
```

In [0]:
```
#converting the list to dataframe
feature_names = ['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']

phishing = pd.DataFrame(phish_features, columns= feature_names)
phishing.head()
```

Out[0]:

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	Tiny_URL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards
0	eevee.tv	0	0	0	4	0	0	0	0	0	1	0	0	0	0	1	0
1	appleid.apple.com-sa.pm	0	0	0	1	0	0	0	1	0	1	1	1	0	0	1	0
2	grandcup.xyz	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1
3	villa-azzurro.com	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0
4	mygpstrip.net	0	0	0	2	0	0	0	0	0	1	0	1	0	0	1	0

In [0]:
```
# Storing the extracted legitimate URLs fatures to csv file
phishing.to_csv('phishing.csv', index= False)
```

Final Dataset

In the above section we formed two data frames of legitimate & phishing URL features. Now, we will combine them to a single data frame and export the data to csv file for the Machine Learning training done in other notebook.

In [0]:
```
#Concatenating the dataframes into one
urldata = pd.concat([legitimate, phishing]).reset_index(drop=True)
urldata.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	Lab
0	graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	

In [0]:
```
urldata.tail()
```

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_Forward
9995	wvk12-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	
9996	adplife.com	0	0	1	4	0	0	0	0	0	1	0	1	0	0	1	
9997	kurortnoye.com.ua	0	1	1	3	0	0	1	0	0	0	1	1	1	0	1	
9998	norcaltc-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	
9999	sieck-kuehlsysteme.de	0	1	1	4	0	0	1	1	0	1	1	1	0	0	1	

In [0]:
```
urldata.shape
```

Out[0]:
```
(10000, 18)
```

In [0]:
```
# Storing the data in CSV file
urldata.to_csv('urldata.csv', index=False)
```

With this the objective of this notebook is achieved. We finally extracted 18 features for 10,000 URL which has 5000 phishing & 5000 legitimate URLs.

6.2 Detecting Phishing Website

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 resulted csv file is uploaded to this notebook and stored in the data frame.

```
In [1]: #importing basic packages
        import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
```

```
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	Lab
0	graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	

```
In [0]: #Checking the shape of the dataset
        data0.shape

Out[0]: (10000, 18)
```

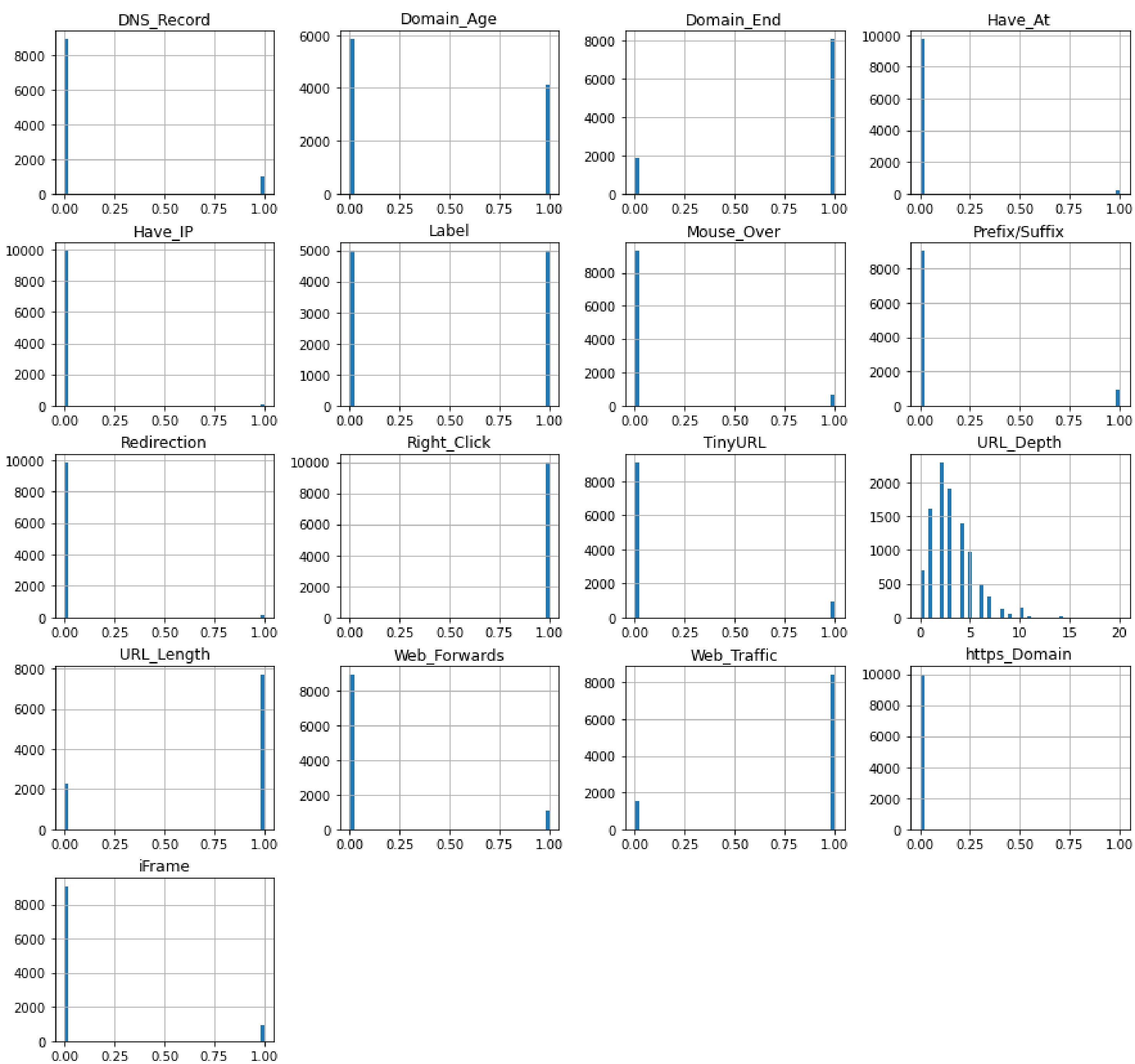
```
In [0]: #Listing the features of the dataset
        data0.columns

Out[0]: Index(['Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
               'Redirection', 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Record',
               'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over',
               'Right_Click', 'Web_Forwards', 'Label'],
              dtype='object')
```

```
In [0]: #Information about the dataset
        data0.info()

        <class 'pandas.core.frame.DataFrame'>
        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
```

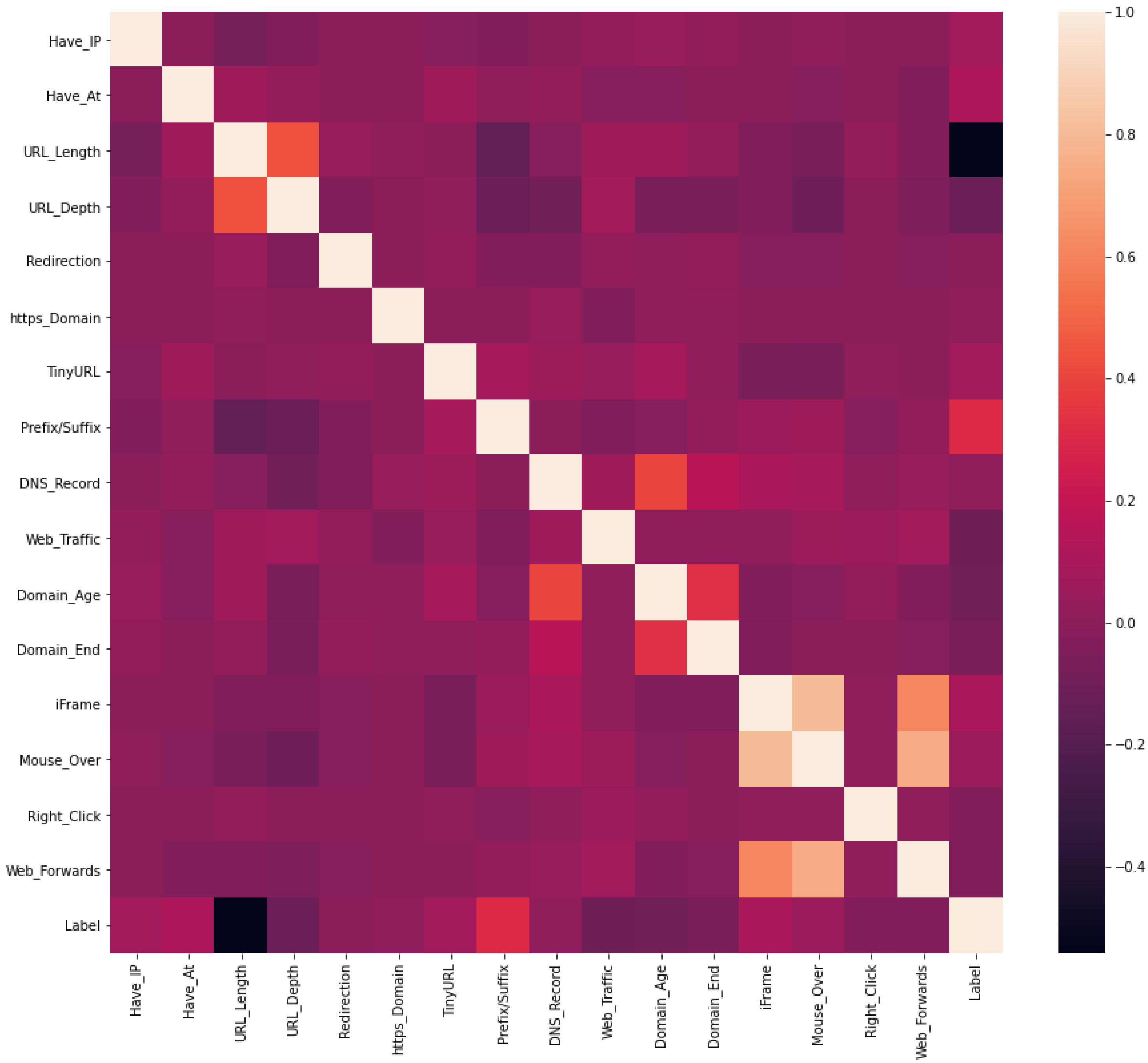
Visualizing the data

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()
```



In [0]:
```python
#Correlation heatmap

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



Data Preprocessing & EDA

Here, we clean the data by applying data preprocessing 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	We
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	1(
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	
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	
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	
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	
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	
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	
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	

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 doesn't have any missing values.

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

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))
```

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))
```

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)
```

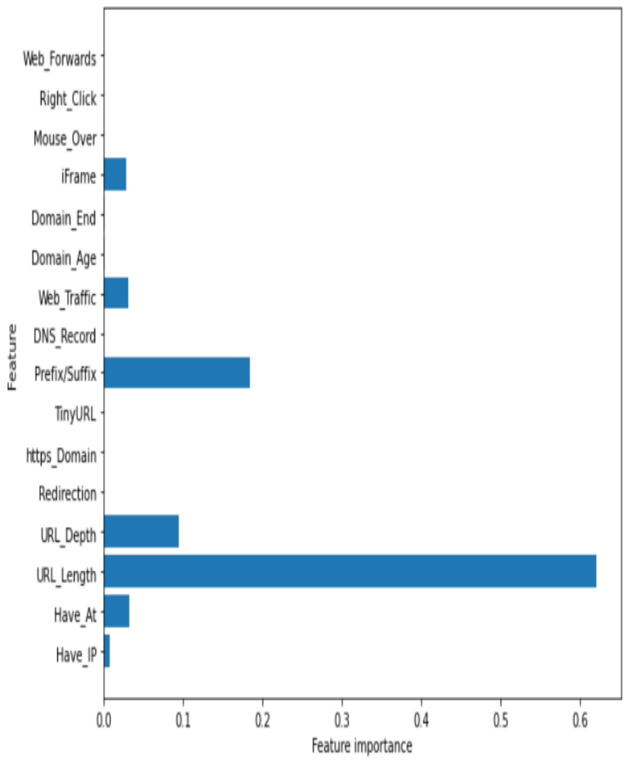
Performance Evaluation:

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
```

Checking the feature importance in the model & storing the results.

```
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)
```

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)
```

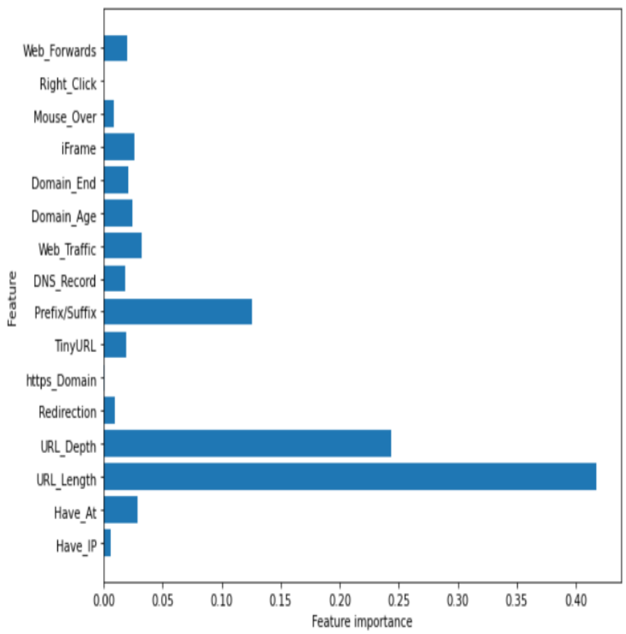
Performance Evaluation:

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
```

Checking the feature importance in the model & storing the results.

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)
```

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)
```

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)
```

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.

Importing required packages,

```
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
```

```
In [0]: #building autoencoder model

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
_________________________________________________________________
```

Compiling the model

In [0]:
```
#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)
```

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)
```

Comparison 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
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.

Saving the model file & testing the same.

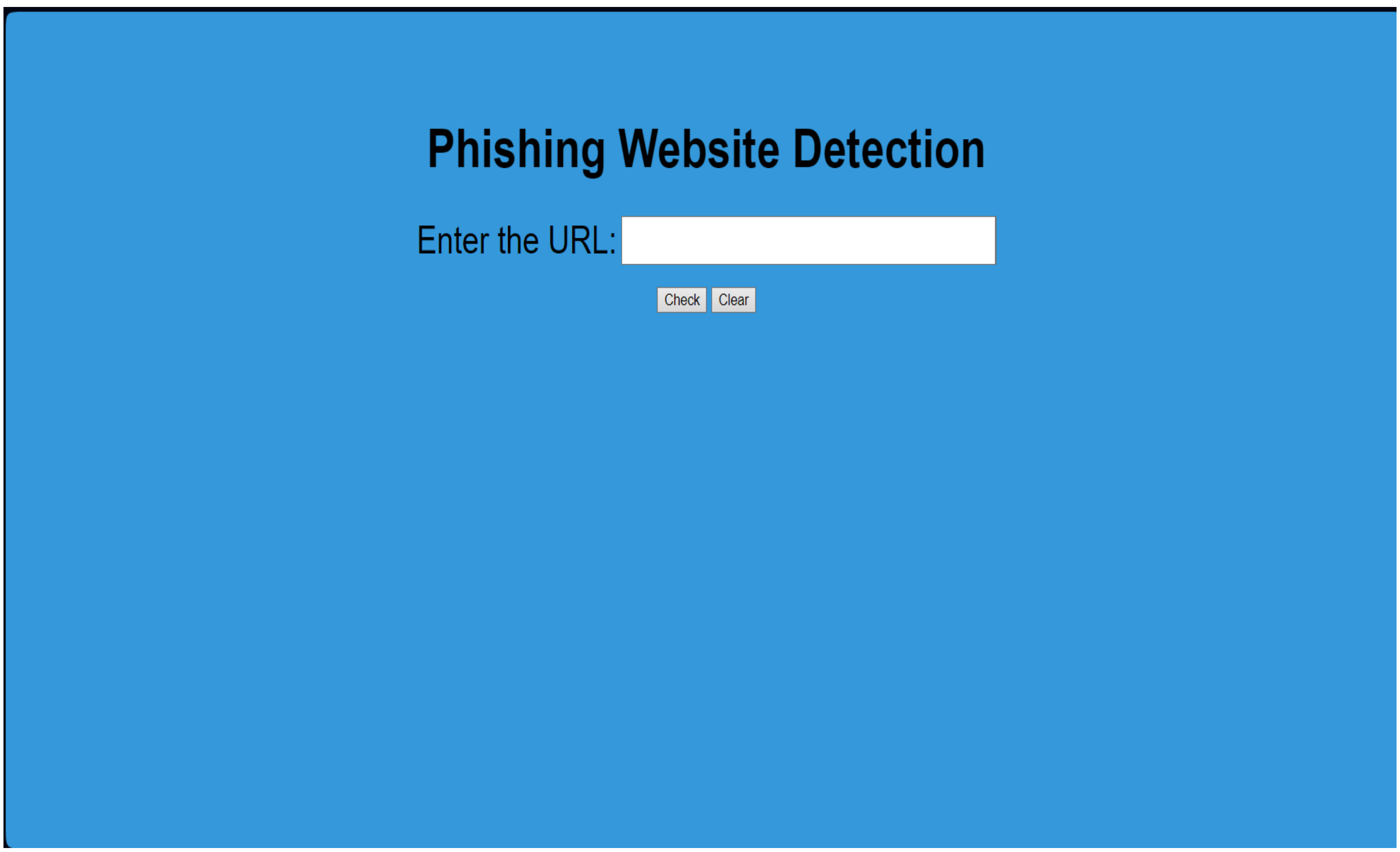
```
In [0]: # save XGBoost model to file
        import pickle
        pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

Testing the saved model:

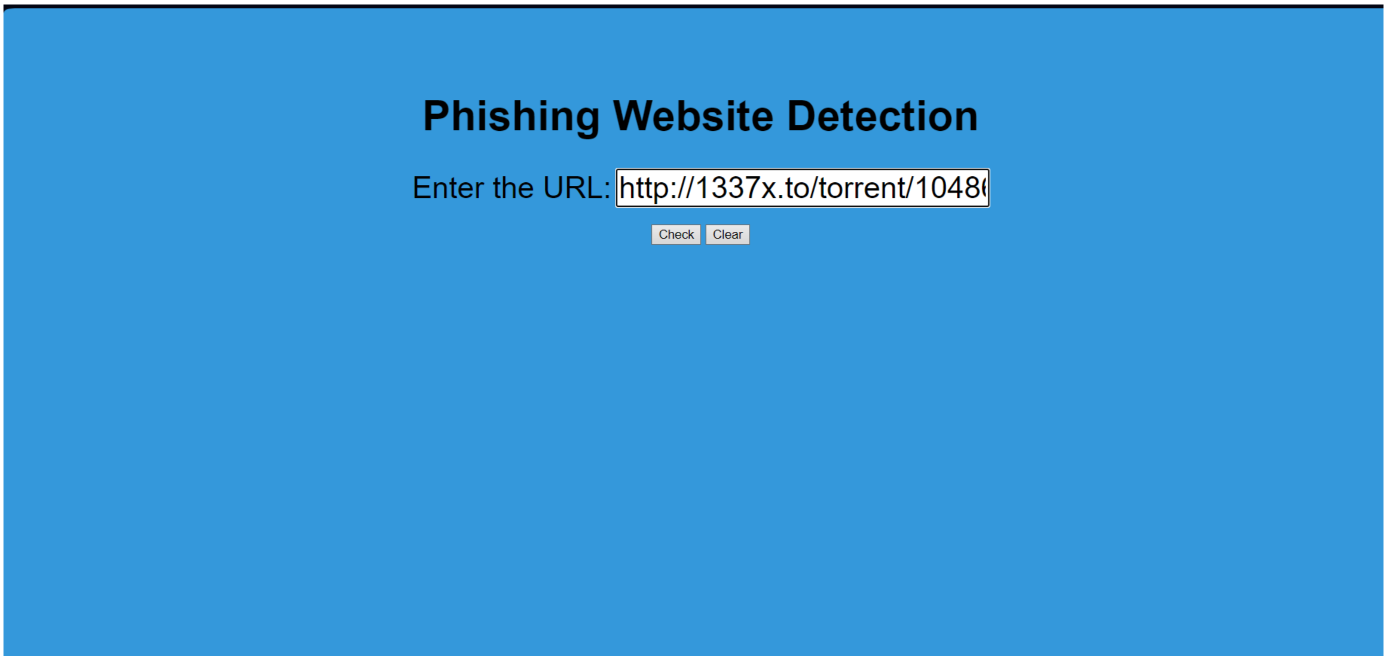
```
In [0]: # load model from file
        loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
        loaded_model

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)
```

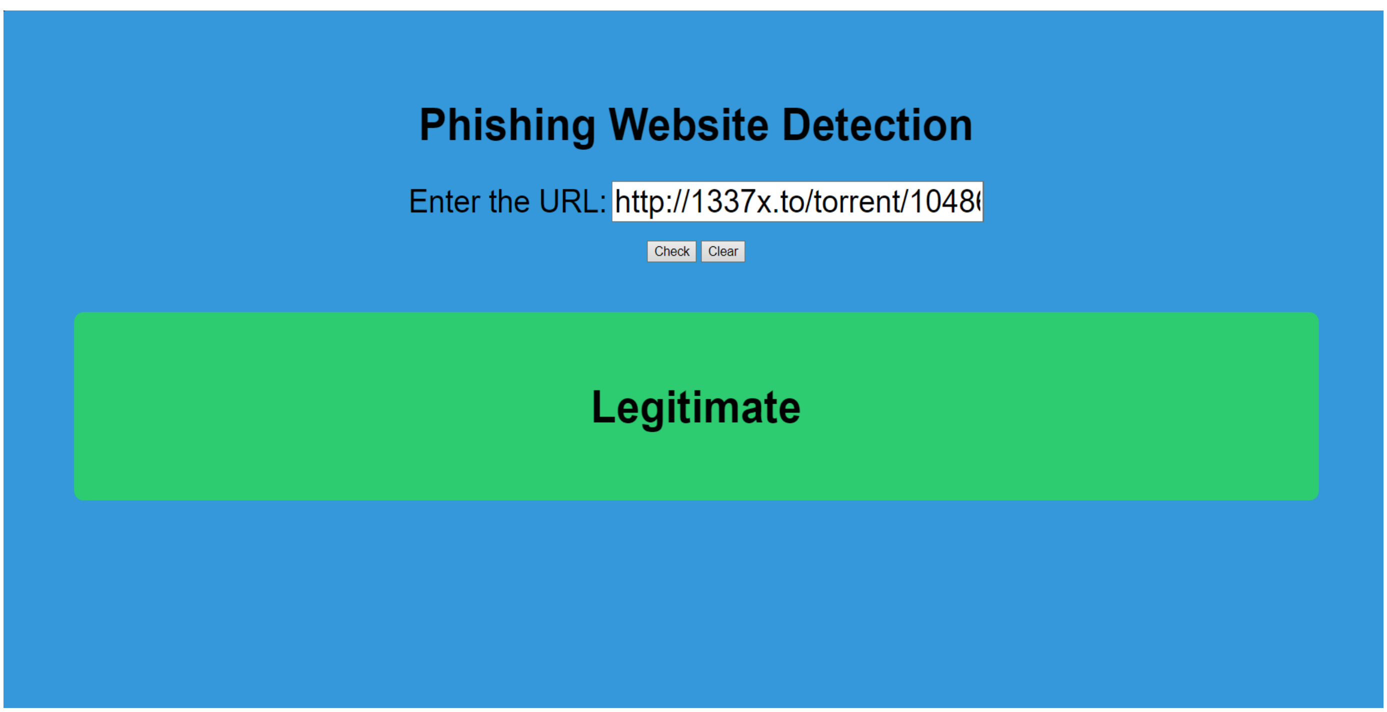
7. Front Page

7.1 Home Page



7.2 Legitimate URL





7.3 Phishing URL

8. Future Plans

Engaging in this project is not only intellectually stimulating but also highly rewarding, offering a wealth of knowledge and insights into the intricate realm of phishing websites and their differentiation from legitimate ones. As the project progresses, a deeper understanding of the nuanced features distinguishing these two categories unfolds, contributing to a broader comprehension of cybersecurity challenges. To extend the impact of this research, future steps may involve taking the project a step further by considering the development of browser extensions or creating a Graphical User Interface (GUI). Such extensions or interfaces could leverage the insights gained to classify inputted URLs, determining their legitimacy or phishing status using the trained and saved machine learning model. This extension or GUI would serve as a practical tool, providing users with real-time information and contributing to the broader initiative of fortifying online security measures. These next steps not only enhance the practical utility of the project but also open avenues for broader applications in the ongoing fight against evolving cyber threats.

9. BIBLIOGRAPHY

[1] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel, "PhishStorm: Detecting Phishing with Streaming Analytics," IEEE Transactions on Network and Service Management, vol. 11, issue: 4, pp. 458-471, December 2014

[2] Mohammed Nazim Feroz, Susan Mengel, "Phishing URL Detection Using URL Ranking," IEEE International Congress on Big Data, July 2015

[3] Mahdieh Zabihimayvan, Derek Doran, "Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection," International Conference on Fuzzy Systems (FUZZ-IEEE), New Orleans, LA,

USA, June 2019

[4] Moitrayee Chatterjee, Akbar-Siami Namin, "Detecting Phishing Websites through Deep Reinforcement Learning," IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), July 2019

[5] Chun-Ying Huang, Shang-Pin Ma, Wei-Lin Yeh, Chia-Yi Lin, ChienTsung Liu, "Mitigate web phishing using site signatures," TENCON 2010-2010 IEEE Region 10 Conference, January 2011

Website Reference:

- 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/siddharthanshree/Sid
- https://machinelearningmastery.com/save-gradient-boosting-models-xgboost-python/
- https://archive.ics.uci.edu/ml/datasets/Phishing+Websites