CHAPTER – 1

INTRODUCTION :

Phishing is one of the techniques which are used by the intruders to get access to the user credentials or to gain access to the sensitive data. This type of accessing the is done by creating the replica of the websites which looks same as the original websites which we use on our daily basis but when a user click on the link he will see the website and think its original and try to provide his credentials .To overcome this problem we are using Multilayer stacked ensemble learning algorithm in which it will help us to identify the phishing websites based on the features present in the algorithm. By using these algorithm we can able to detect whether the URL is legitimate or phishing**.**

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.

The Internet has become an indispensable part of our life, However, It also has provided opportunities to anonymously perform malicious activities like Phishing. Phishers try to deceive their victims by social engineering or creating mockup websites to steal information such as account ID, username, password from individuals and organizations. Although many methods have been proposed to detect phishing websites, Phishers have evolved their methods to escape from these detection methods. One of the most successful methods for detecting these malicious activities is Machine Learning. This is because most Phishing attacks have some common characteristics which can be identified by machine learning methods.

CHAPTER – 2

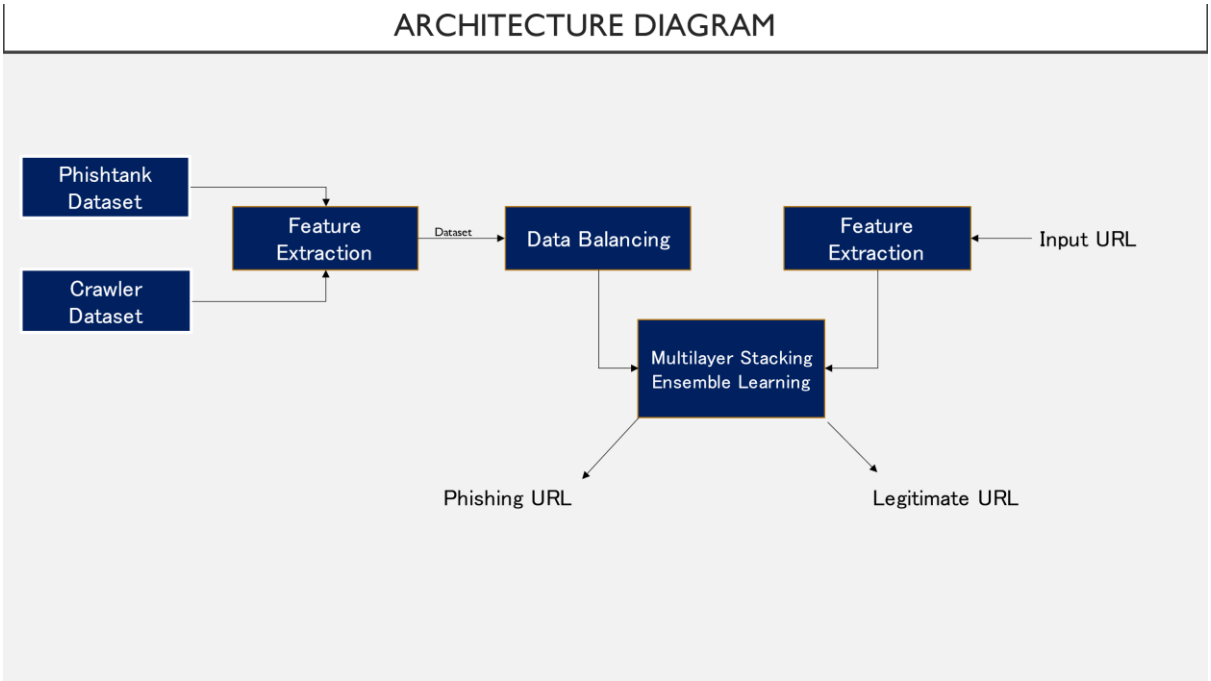
LITERATURE SURVEY

S.NO	Author & Title	Methods	Advantages	Disadvantages
1	**Manuel Sánchez-Paniagua, Eduardo Fidalgo Fernández, Enrique Alegre** - Phishing URL Detection: A Real-Case Scenario Through Login URLs	Machine Learning Techniques 1.Markov Chain Model, Random Forest , support vector machine, KNN, Naïve bayes Deep Learning Techniques 1.CNN	They used a big dataset to detect the phishing URL	They didn't proposed any model to predict the phishing they just tested their dataset with already proposed model
2	**Subhash Ariyadasa, Shantha Fernando, Subha Fernando** - Combining Long-Term Recurrent Convolutional and Graph Convolutional Networks to Detect Phishing Sites Using URL and HTML.	1. LSTM, 2. GCN, 3. LRCN, 4. CNN	The model will detect the whether the URL is phishing or not by using LRCN, and it will detect the whether the HTML code is phishing or not by using GCC	For prediction they used content based prediction basically content based approach takes log time to extract the features.
3	**Tuan Dung Pham, Thi Thanh Thuy Pham, Sy Tuong Hoang, Viet Cuong Ta** - PDGAN: Phishing Detection With Generative Adversarial Networks	1.LSTM , 2.CNN	Their model does not depends on website content and third party services to detect the phishing URL	It will detect URL whether phishing or not only generated by the LSTM.

4	**Xiuwen Liu, Jianming Fu** - SPWalk: Similar Property Oriented Feature Learning for Phishing Detection	Hidden Markov model	It will detect whether URL is phishing or not by similarity based approach.	The result depends on third party services (Google Page Rank)
5	**Maria Sameen** - PhishHaven—An Efficient Real-Time AI Phishing URLs Detection System	Decision Tree, KNN, Random Forest, Logistic Regression, Ada Boosting	They used multi – threading approach for detection	They used a third party AI for feature extraction.

CHAPTER – 3

SYSTEM DESIGN :



Architecture Diagram Description

Phishtank Dataset:

Phishtank Dataset contains list of Phishing URLs.

Crawler Dataset:

Crawler Dataset contains the list of legitimate URLs.

Feature Extraction:

To Extract the feature we categorized the features into three categories

1. Address Bar Based Features
2. Domain Based Features
3. HTML / JavaScript Based Features

Address Bar Based Features:

1. Domain name Based Feature
2. IP address in URL.
3. '@' symbol in domain.
4. Length of URL
5. Depth of URL
6. Redirection in '//' in URL
7. 'http' or 'https' in URL
8. Prefix and suffix '-' in URL.

Domain Based Features:

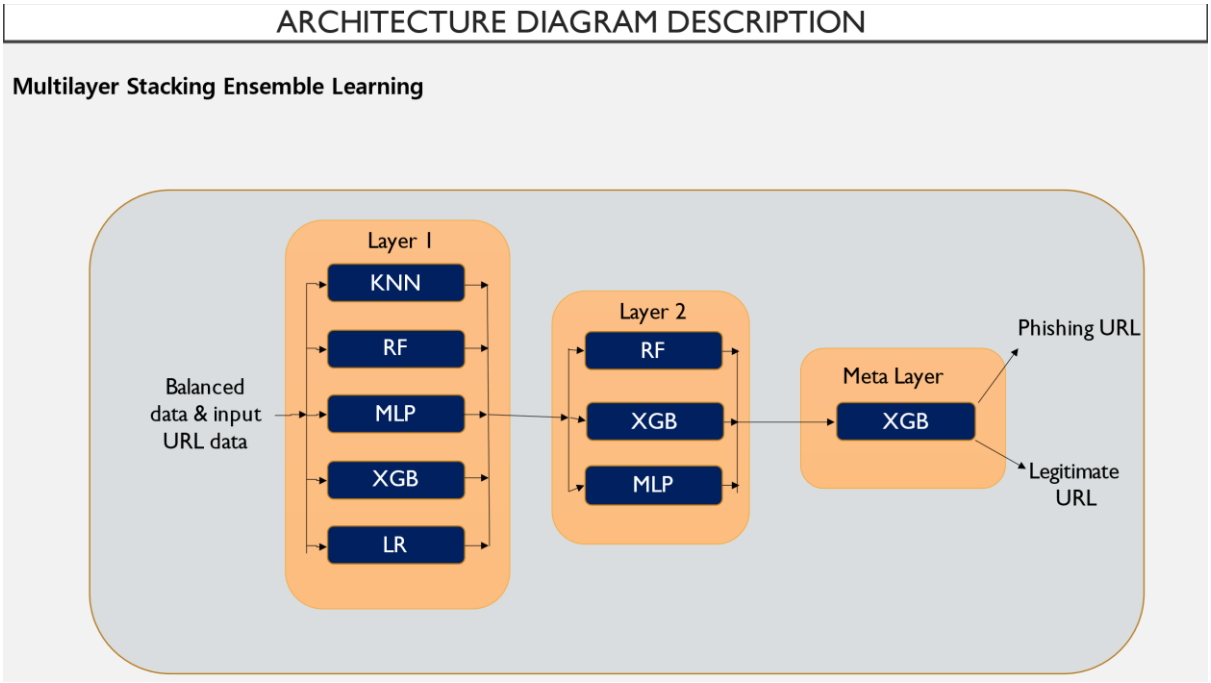
1. Domain Record
2. Web Traffic
3. Age of Domain
4. End period of Domain

HTML / JavaScript Based Features:

1. Iframe Redirection
2. Status Bar Customization
3. Disabling Right Click.
4. Website Forwarding.

Data Balancing:

Imbalanced dataset may detect wrongly. To avoid wrong prediction we need to balance our dataset. A balanced dataset refers to a dataset whose distribution of labels is approximately equal.



MODULE DESCRIPTION

ARCHITECTURE DIAGRAM DESCRIPTION

Stacking Ensemble Learning:

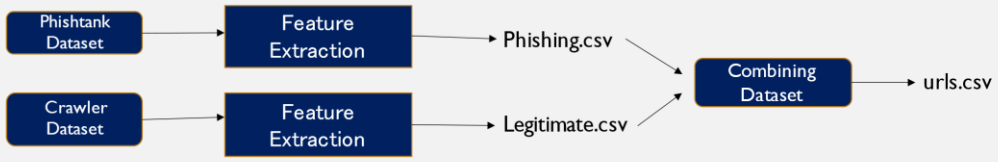
Stacking consists n number of classification algorithms and one meta learner. The outputs of classification algorithms will give to meta learner as input. The meta model will classify the input.

Multilayer Stacking Ensemble Learning:

A possible extension of Stacking is multi layer stacking. In first Layer we fit N number of classification algorithm and the output of first layer will give input to the next layer. Next layer contains m number of classification algorithm as meta learner. Finally the last layer has one meta learner.

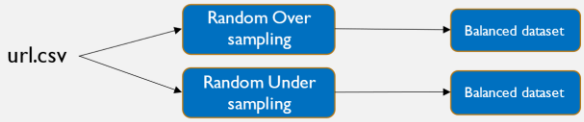
MODULE DESCRIPTION

Module 1: Feature Extraction for phishing and legitimate URL Dataset



MODULE DESCRIPTION

Module 2: Data Balancing



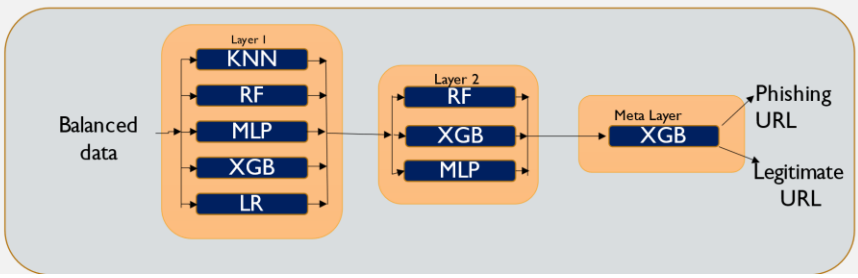
Random Over sampling:

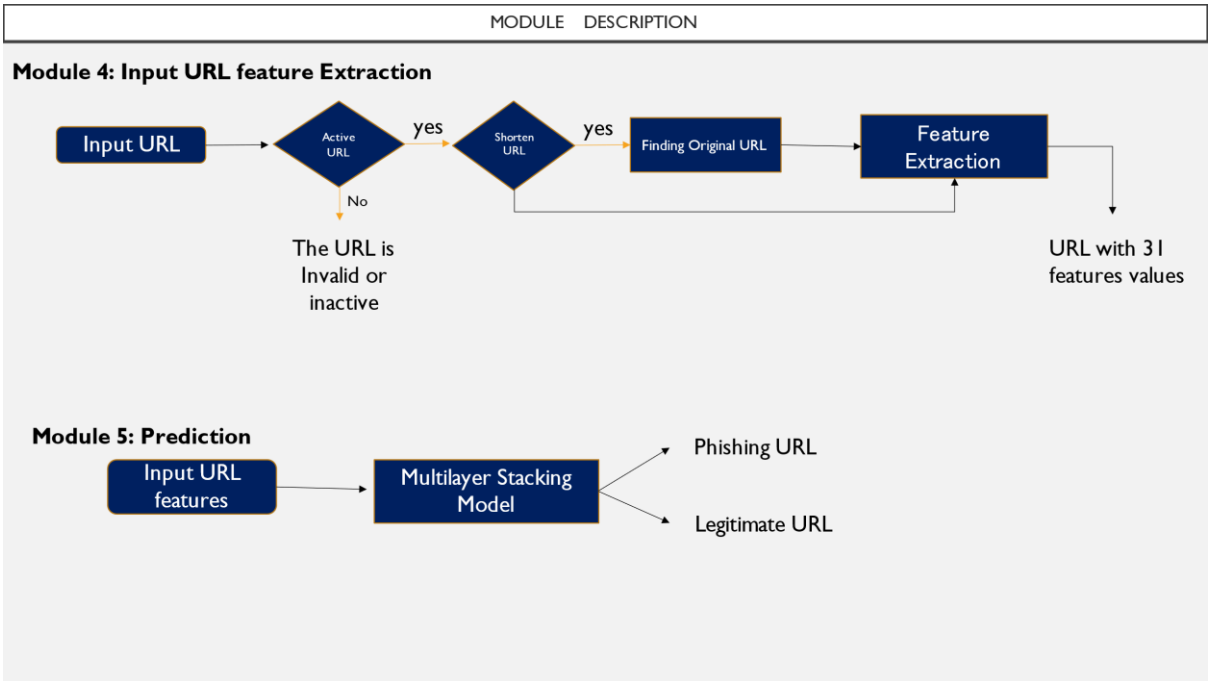
Random over sampling involves randomly duplicating example from the minority class and adding them to the training dataset.

Random Under sampling:

Random under sampling involves randomly deletes example from the majority class and adding them to the training dataset.

Module 3: Multilayer Stacking Ensemble Learning Model Training





CHAPTER – 4

IMPLEMENTATION :

INDEX.HTML :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="This website is develop for identify the safety of url.">
    <meta name="keywords" content="phishing url,phishing,cyber security,machine learning,classifier,python">
    <meta name="author" content="VAIBHAV BICHAVE">

    <!-- BootStrap -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
```

```
    integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">

  <link href="static/styles.css" rel="stylesheet">
  <title>URL detection</title>

</head>

<body>

<div class=" container">
  <div class="row">
    <div class="form col-md" id="form1">
      <h2>PHISHING URL DETECTION</h2>

      <br>
      <form action="/" method ="post">
        <input type="text" class="form__input" name ='url' id="url" placeholder="Enter URL"
required="" />
        <label for="url" class="form__label">URL</label>
        <button class="button" role="button" >Check here</button>
      </form>

  </div>

  <div class="col-md" id="form2">

    <br>
    <h6 class = "right "><a href= {{ url }} target="_blank">{{ url }}</a></h6>

    <br>
    <h3 id="prediction"></h3>
```

```
      <button class="button2" id="button2" role="button" onclick="window.open('{{url}}')"
target="_blank" >Still want to Continue</button>

      <button class="button1" id="button1" role="button"  onclick="window.open('{{url}}')"
target="_blank">Continue</button>

   </div>

</div>

<br>

<p> 2022 GCT STUDENT</p>

</div>


   <!-- JavaScript -->

   <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"

      integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"

      crossorigin="anonymous"></script>

   <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"

      integrity="sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"

      crossorigin="anonymous"></script>

   <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"

      integrity="sha384-
OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"

      crossorigin="anonymous"></script>



   <script>


         let x = '{{xx}}';

         let num = x*100;

         if (0<=x && x<0.50){

            num = 100-num;

         }

         let txtx = num.toString();

         if(x<=1 && x>=0.50){
```

```
            var label = "Website is "+txtx +"% safe to use...";
            document.getElementById("prediction").innerHTML = label;
            document.getElementById("button1").style.display="block";
        }
        else if (0<=x && x<0.50){
            var label = "Website is "+txtx +"% unsafe to use..."
            document.getElementById("prediction").innerHTML = label ;
            document.getElementById("button2").style.display="block";
        }

    </script>

</body>

</html>
```

APP.PY :

```
#importing required libraries
from flask import Flask, request, render_template
import numpy as np
import pandas as pd
from sklearn import metrics
import warnings
import pickle
warnings.filterwarnings('ignore')
from feature import FeatureExtraction
file = open("pickle/model.pkl","rb")
gbc = pickle.load(file)
file.close()
```

```
app = Flask(_name_)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":

        url = request.form["url"]
        obj = FeatureExtraction(url)
        x = np.array(obj.getFeaturesList()).reshape(1,30)

        y_pred =gbc.predict(x)[0]
        #1 is safe
        #-1 is unsafe
        y_pro_phishing = gbc.predict_proba(x)[0,0]
        y_pro_non_phishing = gbc.predict_proba(x)[0,1]
        # if(y_pred ==1 ):
        pred = "It is {0:.2f} % safe to go ".format(y_pro_phishing*100)
        return render_template('index.html',xx =round(y_pro_non_phishing,2),url=url )
    return render_template("index.html", xx =-1)



if _name_ == "_main_":
    app.run(debug=True)
```

FEATURE.PY :

```
 import ipaddress
import re
import urllib.request
from bs4 import BeautifulSoup
import socket
import requests
```

```
from googlesearch import search
import whois
from datetime import date, datetime
import time
from dateutil.parser import parse as date_parse
from urllib.parse import urlparse

class FeatureExtraction:
    features = []
    def _init_(self,url):
        self.features = []
        self.url = url
        self.domain = ""
        self.whois_response = ""
        self.urlparse = ""
        self.response = ""
        self.soup = ""

        try:
            self.response = requests.get(url)
            self.soup = BeautifulSoup(response.text, 'html.parser')
        except:
            pass

        try:
            self.urlparse = urlparse(url)
            self.domain = self.urlparse.netloc
        except:
            pass

        try:
            self.whois_response = whois.whois(self.domain)
```

```
except:
    pass



self.features.append(self.UsingIp())
self.features.append(self.longUrl())
self.features.append(self.shortUrl())
self.features.append(self.symbol())
self.features.append(self.redirecting())
self.features.append(self.prefixSuffix())
self.features.append(self.SubDomains())
self.features.append(self.Hppts())
self.features.append(self.DomainRegLen())
self.features.append(self.Favicon())


self.features.append(self.NonStdPort())
self.features.append(self.HTTPSDomainURL())
self.features.append(self.RequestURL())
self.features.append(self.AnchorURL())
self.features.append(self.LinksInScriptTags())
self.features.append(self.ServerFormHandler())
self.features.append(self.InfoEmail())
self.features.append(self.AbnormalURL())
self.features.append(self.WebsiteForwarding())
self.features.append(self.StatusBarCust())

self.features.append(self.DisableRightClick())
self.features.append(self.UsingPopupWindow())
self.features.append(self.IframeRedirection())
```

```
    self.features.append(self.AgeofDomain())
    self.features.append(self.DNSRecording())
    self.features.append(self.WebsiteTraffic())
    self.features.append(self.PageRank())
    self.features.append(self.GoogleIndex())
    self.features.append(self.LinksPointingToPage())
    self.features.append(self.StatsReport())


 # 1.UsingIp
def UsingIp(self):
    try:
        ipaddress.ip_address(self.url)
        return -1
    except:
        return 1

# 2.longUrl
def longUrl(self):
    if len(self.url) < 54:
        return 1
    if len(self.url) >= 54 and len(self.url) <= 75:
        return 0
    return -1

# 3.shortUrl
def shortUrl(self):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
                'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
                'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'
                'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
                'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
```

```
            'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'

'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|tr\.im|link\.zip\.net', self.url)
        if match:
            return -1
        return 1

    # 4.Symbol@
    def symbol(self):
        if re.findall("@",self.url):
            return -1
        return 1

    # 5.Redirecting//
    def redirecting(self):
        if self.url.rfind('//')>6:
            return -1
        return 1

    # 6.prefixSuffix
    def prefixSuffix(self):
        try:
            match = re.findall('\-', self.domain)
            if match:
                return -1
            return 1
        except:
            return -1

    # 7.SubDomains
    def SubDomains(self):
        dot_count = len(re.findall("\.", self.url))
```

```
    if dot_count == 1:
        return 1
    elif dot_count == 2:
        return 0
    return -1

# 8.HTTPS
def Hppts(self):
    try:
        https = self.urlparse.scheme
        if 'https' in https:
            return 1
        return -1
    except:
        return 1

# 9.DomainRegLen
def DomainRegLen(self):
    try:
        expiration_date = self.whois_response.expiration_date
        creation_date = self.whois_response.creation_date
        try:
            if(len(expiration_date)):
                expiration_date = expiration_date[0]
        except:
            pass
        try:
            if(len(creation_date)):
                creation_date = creation_date[0]
        except:
            pass
```

```
            age = (expiration_date.year-creation_date.year)*12+ (expiration_date.month-creation_date.month)
            if age >=12:
                return 1
            return -1
        except:
            return -1

    # 10. Favicon
    def Favicon(self):
        try:
            for head in self.soup.find_all('head'):
                for head.link in self.soup.find_all('link', href=True):
                    dots = [x.start(0) for x in re.finditer('\.', head.link['href'])]
                    if self.url in head.link['href'] or len(dots) == 1 or domain in head.link['href']:
                        return 1
            return -1
        except:
            return -1

    # 11. NonStdPort
    def NonStdPort(self):
        try:
            port = self.domain.split(":")
            if len(port)>1:
                return -1
            return 1
        except:
            return -1

    # 12. HTTPSDomainURL
    def HTTPSDomainURL(self):
        try:
```

```
        if 'https' in self.domain:
            return -1
        return 1
    except:
        return -1

# 13. RequestURL
def RequestURL(self):
    try:
        for img in self.soup.find_all('img', src=True):
            dots = [x.start(0) for x in re.finditer('\.', img['src'])]
            if self.url in img['src'] or self.domain in img['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for audio in self.soup.find_all('audio', src=True):
            dots = [x.start(0) for x in re.finditer('\.', audio['src'])]
            if self.url in audio['src'] or self.domain in audio['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for embed in self.soup.find_all('embed', src=True):
            dots = [x.start(0) for x in re.finditer('\.', embed['src'])]
            if self.url in embed['src'] or self.domain in embed['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for iframe in self.soup.find_all('iframe', src=True):
            dots = [x.start(0) for x in re.finditer('\.', iframe['src'])]
            if self.url in iframe['src'] or self.domain in iframe['src'] or len(dots) == 1:
                success = success + 1
            i = i+1
```

```
            try:
                percentage = success/float(i) * 100
                if percentage < 22.0:
                    return 1
                elif((percentage >= 22.0) and (percentage < 61.0)):
                    return 0
                else:
                    return -1
            except:
                return 0
        except:
            return -1

    # 14. AnchorURL
    def AnchorURL(self):
        try:
            i,unsafe = 0,0
            for a in self.soup.find_all('a', href=True):
                if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or not (url in a['href'] or self.domain in a['href']):
                    unsafe = unsafe + 1
                i = i + 1


            try:
                percentage = unsafe / float(i) * 100
                if percentage < 31.0:
                    return 1
                elif ((percentage >= 31.0) and (percentage < 67.0)):
                    return 0
                else:
                    return -1
            except:
```

```
            return -1

    except:
        return -1

# 15. LinksInScriptTags
def LinksInScriptTags(self):
    try:
        i,success = 0,0

        for link in self.soup.find_all('link', href=True):
            dots = [x.start(0) for x in re.finditer('\.', link['href'])]
            if self.url in link['href'] or self.domain in link['href'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for script in self.soup.find_all('script', src=True):
            dots = [x.start(0) for x in re.finditer('\.', script['src'])]
            if self.url in script['src'] or self.domain in script['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        try:
            percentage = success / float(i) * 100
            if percentage < 17.0:
                return 1
            elif((percentage >= 17.0) and (percentage < 81.0)):
                return 0
            else:
                return -1
        except:
            return 0
```

```
    except:
        return -1

# 16. ServerFormHandler
def ServerFormHandler(self):
    try:
        if len(self.soup.find_all('form', action=True))==0:
            return 1
        else :
            for form in self.soup.find_all('form', action=True):
                if form['action'] == "" or form['action'] == "about:blank":
                    return -1
                elif self.url not in form['action'] and self.domain not in form['action']:
                    return 0
                else:
                    return 1
    except:
        return -1

# 17. InfoEmail
def InfoEmail(self):
    try:
        if re.findall(r"[mail\(\)|mailto:?]", self.soap):
            return -1
        else:
            return 1
    except:
        return -1

# 18. AbnormalURL
def AbnormalURL(self):
    try:
```

```
        if self.response.text == self.whois_response:
            return 1
        else:
            return -1
    except:
        return -1

# 19. WebsiteForwarding
def WebsiteForwarding(self):
    try:
        if len(self.response.history) <= 1:
            return 1
        elif len(self.response.history) <= 4:
            return 0
        else:
            return -1
    except:
         return -1

# 20. StatusBarCust
def StatusBarCust(self):
    try:
        if re.findall("<script>.+onmouseover.+</script>", self.response.text):
            return 1
        else:
            return -1
    except:
         return -1

# 21. DisableRightClick
def DisableRightClick(self):
    try:
```

```
        if re.findall(r"event.button ?== ?2", self.response.text):
            return 1
        else:
            return -1
    except:
        return -1

# 22. UsingPopupWindow
def UsingPopupWindow(self):
    try:
        if re.findall(r"alert\(", self.response.text):
            return 1
        else:
            return -1
    except:
        return -1

# 23. IframeRedirection
def IframeRedirection(self):
    try:
        if re.findall(r"[<iframe>|<frameBorder>]", self.response.text):
            return 1
        else:
            return -1
    except:
        return -1

# 24. AgeofDomain
def AgeofDomain(self):
    try:
        creation_date = self.whois_response.creation_date
        try:
```

```
            if(len(creation_date)):
                creation_date = creation_date[0]
        except:
            pass

        today  = date.today()
        age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
        if age >=6:
            return 1
        return -1
    except:
        return -1

# 25. DNSRecording
def DNSRecording(self):
    try:
        creation_date = self.whois_response.creation_date
        try:
            if(len(creation_date)):
                creation_date = creation_date[0]
        except:
            pass

        today  = date.today()
        age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
        if age >=6:
            return 1
        return -1
    except:
        return -1

# 26. WebsiteTraffic
```

```
    def WebsiteTraffic(self):
        try:
            rank = BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url="
+ url).read(), "xml").find("REACH")['RANK']
            if (int(rank) < 100000):
                return 1
            return 0
        except :
            return -1

    # 27. PageRank
    def PageRank(self):
        try:
            prank_checker_response = requests.post("https://www.checkpagerank.net/index.php",
{"name": self.domain})

            global_rank = int(re.findall(r"Global Rank: ([0-9]+)", rank_checker_response.text)[0])
            if global_rank > 0 and global_rank < 100000:
                return 1
            return -1
        except:
            return -1


    # 28. GoogleIndex
    def GoogleIndex(self):
        try:
            site = search(self.url, 5)
            if site:
                return 1
            else:
                return -1
        except:
```

```
            return 1


    # 29. LinksPointingToPage
    def LinksPointingToPage(self):
        try:
            number_of_links = len(re.findall(r"<a href=", self.response.text))
            if number_of_links == 0:
                return 1
            elif number_of_links <= 2:
                return 0
            else:
                return -1
        except:
            return -1


    # 30. StatsReport
    def StatsReport(self):
        try:
            url_match = re.search(

'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|96\.lt|ow\.ly',
url)
            ip_address = socket.gethostbyname(self.domain)
            ip_match =
re.search('146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\.116|78\.46\.211\.158|1
81\.174\.165\.13|46\.242\.145\.103|121\.50\.168\.40|83\.125\.22\.219|46\.242\.145\.98|'


'107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.108|107\.151\.
148\.109|119\.28\.52\.61|54\.83\.43\.69|52\.69\.166\.231|216\.58\.192\.225|'


'118\.184\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219|141\.8\.224\.
221|10\.10\.10\.10|43\.229\.108\.32|103\.232\.215\.140|69\.172\.201\.153|'


'216\.218\.185\.162|54\.225\.104\.146|103\.243\.24\.98|199\.59\.243\.120|31\.170\.160\.61|213\.19\.12
8\.77|62\.113\.226\.131|208\.100\.26\.234|195\.16\.127\.102|195\.16\.127\.157|'
```

```
'34\.196\.13\.28|103\.224\.212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\.200\.56\.183|
23\.253\.164\.103|52\.48\.191\.26|52\.214\.197\.72|87\.98\.255\.18|209\.99\.17\.27|'


'216\.38\.62\.18|104\.130\.124\.96|47\.89\.58\.141|78\.46\.211\.158|54\.86\.225\.156|54\.82\.156\.19|37
\.157\.192\.102|204\.11\.56\.48|110\.34\.231\.42', ip_address)
            if url_match:
                return -1
            elif ip_match:
                return -1
            return 1
        except:
            return 1


    def getFeaturesList(self):
        return self.features
```

Phishing URL detection. ipynb:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')

#Loading data into dataframe

data = pd.read_csv("phishing.csv")
data.head()

data.shape
data.columns

data.info()
data.nunique()

#droping index column

data = data.drop(['Index'],axis = 1)

#description of dataset
```

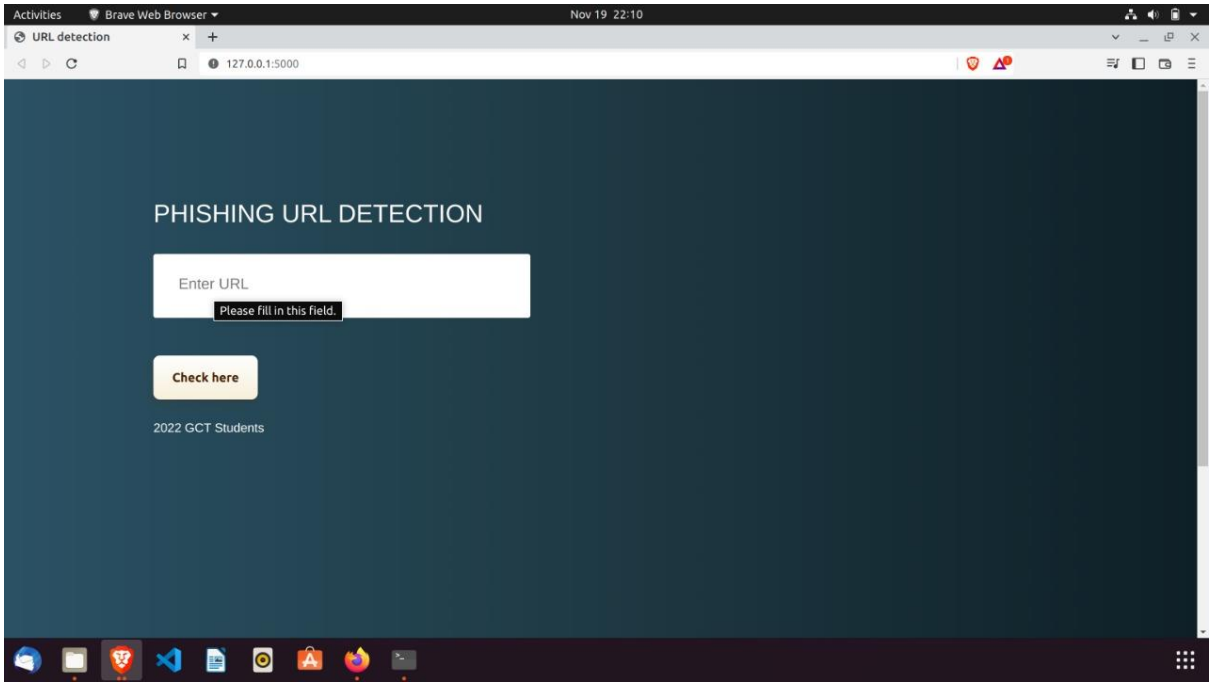
```
data.describe().T



#Correlation heatmap

plt.figure(figsize=(15,15))
sns.heatmap(data.corr(), annot=True)
plt.show()

#pairplot for particular features

df = data[['PrefixSuffix-', 'SubDomains', 'HTTPS','AnchorURL','WebsiteTraffic','class']]
sns.pairplot(data = df,hue="class",corner=True);

# Phishing Count in pie chart

data['class'].value_counts().plot(kind='pie',autopct='%1.2f%%')
plt.title("Phishing Count")
plt.show()

# 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 = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape

# Creating holders to store the model performance results
ML_Model = []
accuracy = []
f1_score = []
recall = []
precision = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
  ML_Model.append(model)
  accuracy.append(round(a, 3))
  f1_score.append(round(b, 3))
  recall.append(round(c, 3))
  precision.append(round(d, 3))

# Gradient Boosting Classifier Model
from sklearn.ensemble import GradientBoostingClassifier

# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model
gbc.fit(X_train,y_train)

#predicting the target value from the model for the samples
y_train_gbc = gbc.predict(X_train)
```

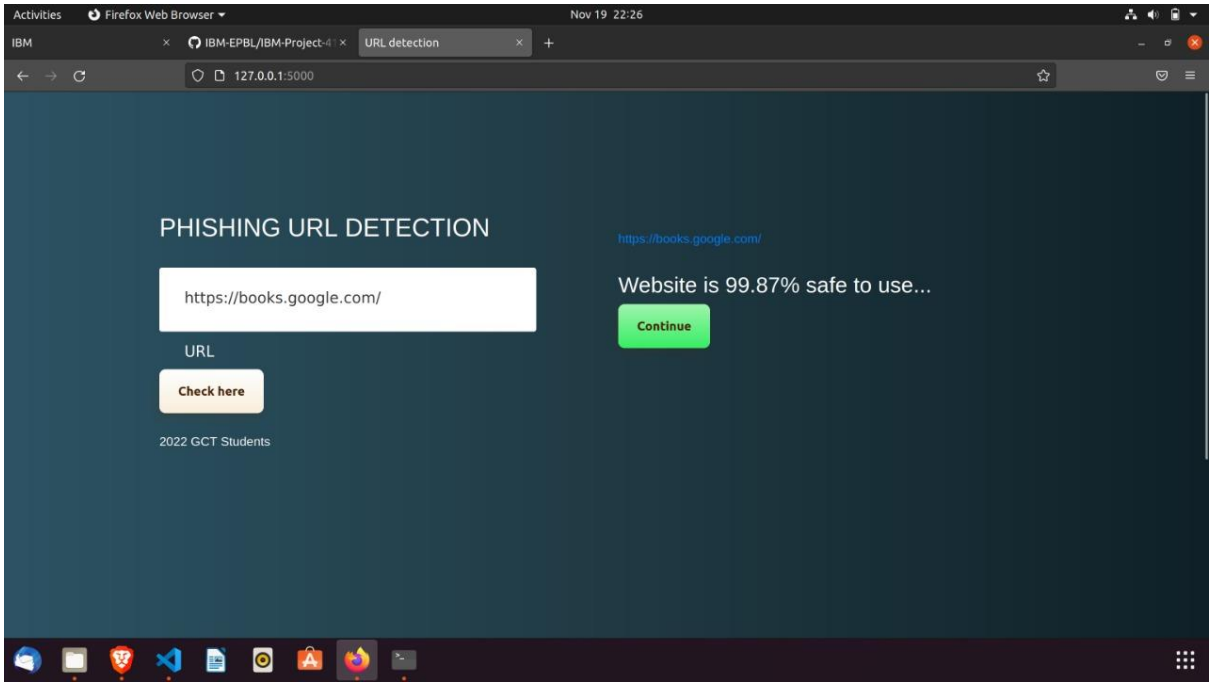
```
y_test_gbc = gbc.predict(X_test)



#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Accuracy on training Data: {:.3f}".format(acc_train_gbc))
print("Gradient Boosting Classifier : Accuracy on test Data: {:.3f}".format(acc_test_gbc))
print()

f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_gbc))
print("Gradient Boosting Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_gbc))
print()

recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc =  metrics.recall_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Recall on training Data: {:.3f}".format(recall_score_train_gbc))
print("Gradient Boosting Classifier : Recall on test Data: {:.3f}".format(recall_score_test_gbc))
print()

precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)
precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : precision on training Data:
{:.3f}".format(precision_score_train_gbc))
print("Gradient Boosting Classifier : precision on test Data: {:.3f}".format(precision_score_test_gbc))

print(metrics.classification_report(y_test, y_test_gbc))

training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  GradientBoostingClassifier(learning_rate = n*0.1)

    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))


#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("learning_rate")
plt.legend();
```

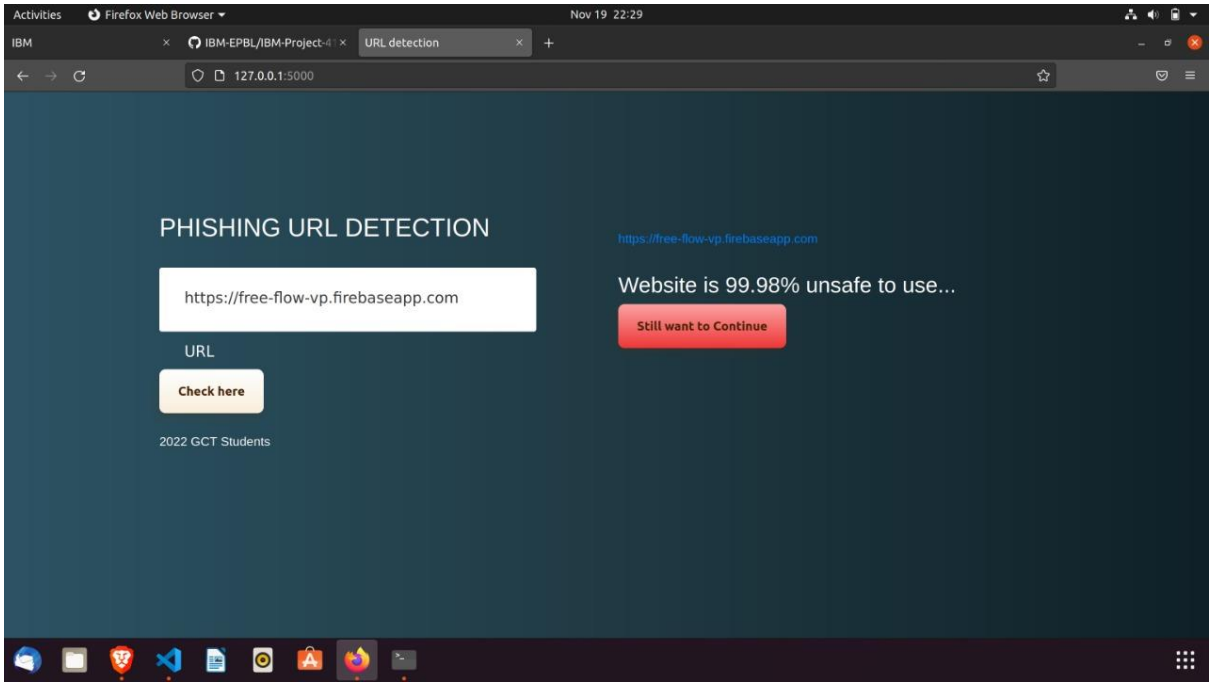
```
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10,1)
for n in depth:
    forest_test =  GradientBoostingClassifier(max_depth=n,learning_rate = 0.7)

    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))


#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();

#storing the results. The below mentioned order of parameter passing is important.

storeResults('Gradient Boosting Classifier',acc_test_gbc,f1_score_test_gbc,
          recall_score_train_gbc,precision_score_train_gbc)

#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                    'Accuracy' : accuracy,
                    'f1_score' : f1_score,
                    'Recall'   : recall,
                    'Precision': precision,
                   })

#Sorting the datafram on accuracy
sorted_result=result.sort_values(by=['Accuracy',
'f1_score'],ascending=False).reset_index(drop=True)
# dispalying total result
sorted_result


from xgboost import XGBClassifier

gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model
gbc.fit(X_train,y_train)


import pickle

# dump information to that file
pickle.dump(gbc, open('pickle/model.pkl', 'wb'))

#checking the feature improtance in the model
```

```
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), gbc.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.title("Feature importances using permutation on full model")
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

IMPLEMENTATION RESULT :




Activities
Firefox Web Browser
Nov 19 22:26
IBM
IBM-EPBL/IBM-Project-4
URL detection
127.0.0.1:5000
PHISHING URL DETECTION
https://books.google.com/
URL
Check here
2022 GCT Students
https://books.google.com/
Website is 99.87% safe to use...
Continue


Activities
Firefox Web Browser
Nov 19 22:29
IBM
IBM-EPBL/IBM-Project-4
URL detection
127.0.0.1:5000
PHISHING URL DETECTION
https://free-flow-vp.firebaseapp.com
URL
Check here
2022 GCT Students
https://free-flow-vp.firebaseapp.com
Website is 99.98% unsafe to use...
Still want to Continue

CHAPTER – 5

RESULT AND CONCLUSION

The final take away form this project is to explore various machine learning models, perform Exploratory Data Analysis on phishing dataset and understanding their features.Creating this notebook helped me to learn a lot about the features affecting the models to detect whether URL is safe or not, also I came to know how to tuned model and how they affect the model performance.

The final conclusion on the Phishing dataset is that the some feature like "HTTTPS", "AnchorURL", "WebsiteTraffic" have more importance to classify URL is phishing URL or not.Gradient Boosting Classifier currectly classify URL upto 97.4% respective classes and hence reduces the chance of malicious attachments.