ABDA PROJECT REPORT

PHISHSING WEBSITE DETECTION USING DIFFERENT ML ALGORITHMS

Performed By [Group 7401]: -

Nidhay Vyas (18082261003)

Neel Rayal (18162121028)

Guided By:-

Prof. Shulabh Bhatt

Submitted to

Department of Computer Science & Engineering
Institute of Computer Technology

Year: 2021

CERTIFICATE

This is to certify that the Advanced Big Data Analysis Project work entitled "**Phishing Website Detection Using Different ML Algorithms**" by NEEL RAYAL (18162121028) NIDHAY VYAS (18082261003) of Ganpat University, towards the fulfillment of requirements of the degree of Bachelor of Technology – Computer Science and Engineering/Masters of computer applications, carried out by them in the CSE(BDA)/DD BCA-MCA(BDA)Department. The results/findings contained in this Project have not been submitted in part or full to any other University / Institute for award of any other Degree/Diploma.

Name & Signature of Internal Guide

Place: ICT – GUNI

Date:

ACKNOWLEDGEMENT

Advanced Big Data Analysis project is a golden opportunity for learning and self-development. I consider myself very lucky and honored to have so many wonderful people lead me through in completion of this project. First and foremost, I would like to thank **Dr. Hemal Shah,** Head of Department, Computer Science and Engineering, who gave us an opportunity to undertake this project. My grateful thanks to **Prof. Shulabh Bhatt** for their guidance in project work **Phishing Website Detection Using Different ML Algorithms**, who despite being extraordinarily busy with academics, took time out to hear, guide and keep us on the correct path. We do not know where would have been without his/her help. CSE/DD BCA-MCA department monitored our progress and arranged all facilities to make life easier. We choose this moment to acknowledge their contribution gratefully.

NEEL RAYAL (ENROLLMENT NO: 18162121028)

NIDHAY VYAS (ENROLLMENT NO: 18082261003)

ABSTRACT

Phishing is the most commonly used social engineering and cyberattack. Through such attacks, the phisher targets naive online users by tricking them into revealing confidential information, with the purpose of using it fraudulently. In order to avoid getting phished users should have awareness of phishing websites, have a blacklist of phishing websites which requires the knowledge of website being detected as phishing, detect them in their early appearance using machine learning models

Index

CHAPTER: 1 INTRODUCTION

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and web pages. The objective of this project is to collect data & extract the selective features from the URLs. Both phishing and benign URLs of websites are gathered to form a dataset and from them required URL and website content-based features are extracted. The performance level of each model is measures and compared.

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 open-source service called **PhishTank**. This service provides a set of phishing URLs in multiple formats like CSV, json, etc. that gets updated hourly.

For the legitimate URLs, we found a source that has a collection of benign, spam, phishing, malware & defacement URLs. The source of the dataset is the University of New Brunswick, the number of legitimate URLs in this collection is 35,300. The URL collection is downloaded & from that, 'Benign_list_big_final.csv' is the file of our interest.

Speaking about tools & technologies for this project, we have used Google Collab.

We have performed feature extraction in the Collab file.

CHAPTER: 2 PROJECT SCOPE

The models which have the highest accuracy can be used to deploy in a webapp or a chrome extension. Users can put up the website and the deployed model will evaluate if the website is phishing or legitimate.

CHAPTER: 3 SOFTWARE AND HARDWARE REQUIREMENTS

CHAPTER 3 SOFTWARE AND HARDWARE REQUIREMENTS

Minimum Hardware Requirements

Processor	2.0 GHz
RAM	4GB
HDD	40GB

Table 3.1 Minimum Hardware Requirements

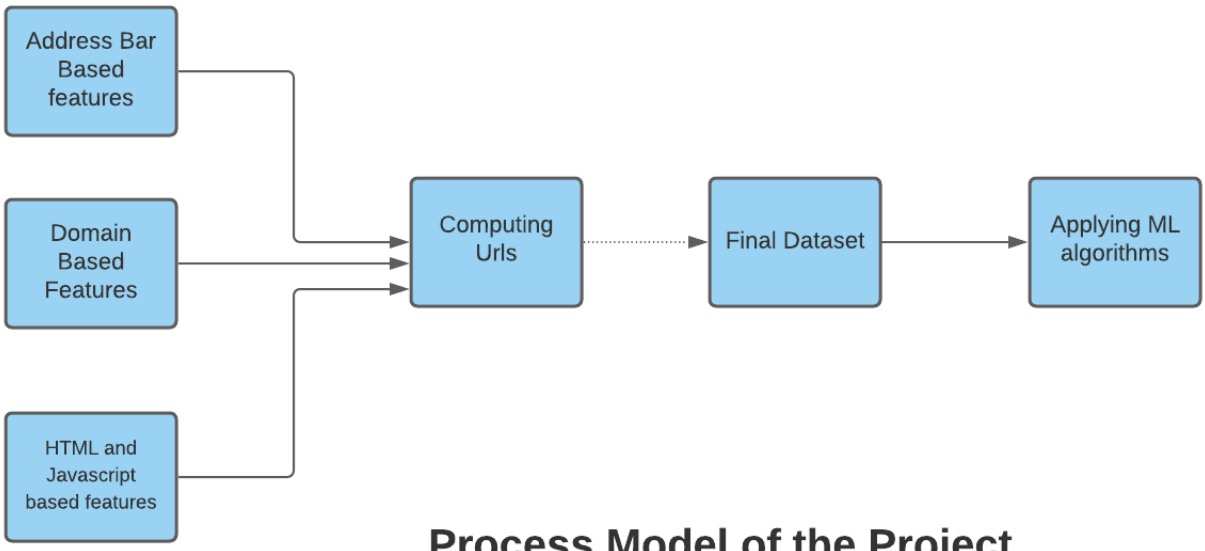
Minimum Software Requirements

Operating System	Any operating system which can support an internet browser.
Programming language	-
Other tools & tech	Internet browser

Table 3.2 Minimum Software Requirements

CHAPTER: 4 PROCESS MODEL

Feature Extraction



Process Model of the Project

CHAPTER: 5 PROJECT PLAN

CHAPTER 5 PROJECT PLAN

5.1 List of Major Activities

Task: 1 Extract all the Address Based Features
Task: 2 Extract all domain-based features
Task: 3 Extract all the HTML and Java-Script based features
Task: 4 After extracting all the features combined all the features into one and made a data frame out of it for both legitimate URLs and phishing URLs.
Task: 5 Understanding the dataset and finding insights
Task: 6 Applying different ML algorithms on the final dataset
Task: 7 Selection of Best Model

5.2 Estimated Time Duration in Days

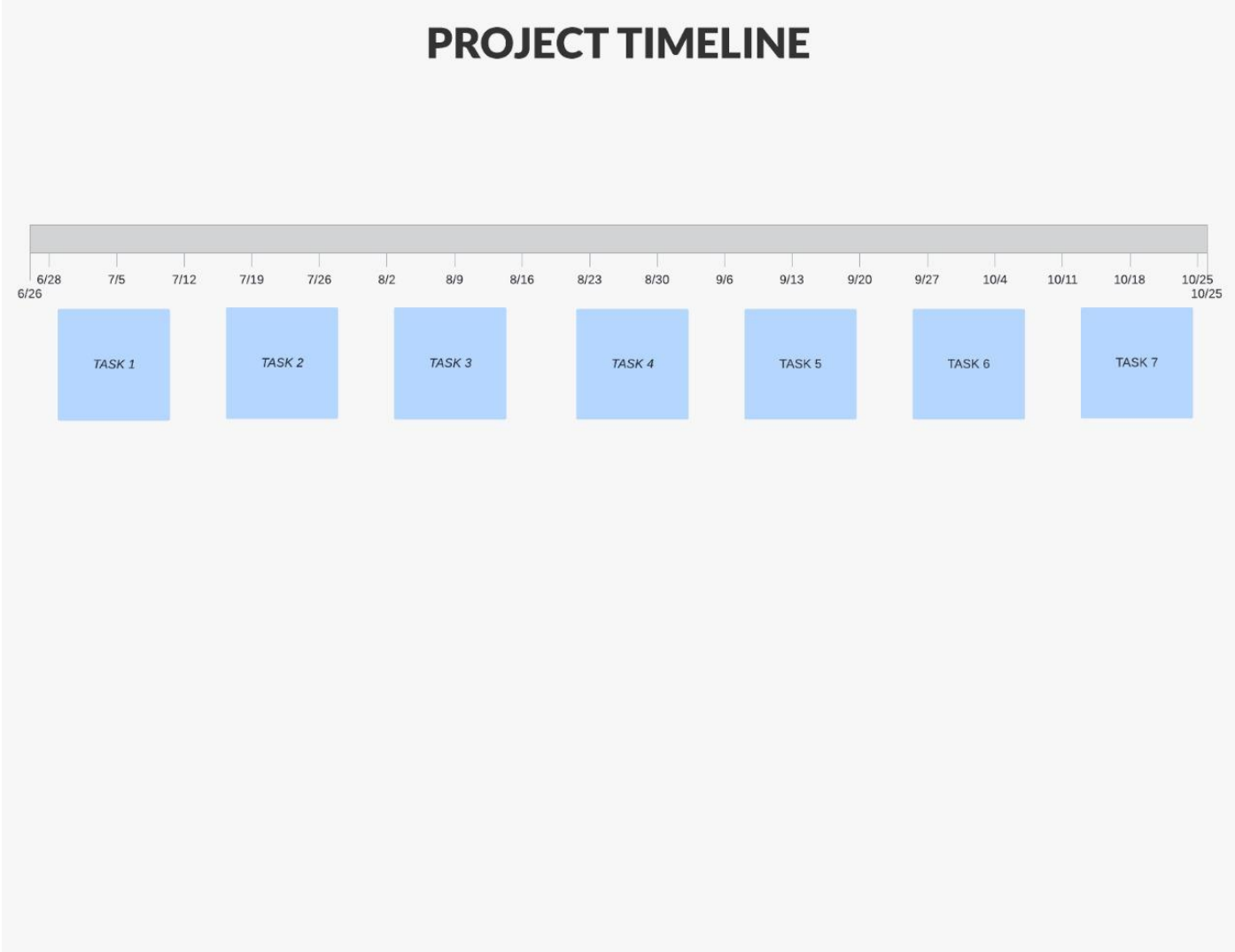


Figure 5.1 Task Completion Estimated Time Duration in Days

CHAPTER: 6 IMPLEMENTATION DETAILS

6.1 Extraction of Address Based Features

Various features are obtained and identified as address bar-based features from which some of them are considered for this project as follows:

1. **Domain of URL.**
   The domain will be fetched from the url.
   Eg. From this link:
   'https://www.youtube.com/results?search_query=extracting+url+information'
   youtube.com will be fetched

```
def getDomain(url):
 domain = urlparse(url).netloc
 if re.match(r"^www.",domain):
       domain = domain.replace("www.","")
 return domain
```

2. **IP Address of URL.**

```
def haveIP(url):
 try:
  ipaddress.ip_address(url)
  ip = 1
 except:
  ip = 0
 return ip
```

This examines the existence of IP address in the URL. If IP address is used then its phishing website otherwise its legitimate.
Eg.
x = '127.0.0.1' will 1 phishing and 'www.google.com' will return legitimate.

3. **"@"Symbol in URL.**

The @ symbol is used to misguide the browser. Presence of @ symbol indicates it might be a phishing website.

```
def haveAt(url):
 if '@' in url:
  return 1
 else:
  return 0
```

4. Length of URL.

```
def urlLength(url):
 if len(url) < 54:
  return 0
 else:
  return 1
```

Phishing websites are lengthy in nature. So if any url length is > 53 then it might be phishing.

5. Depth of URL.

```
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
```

Each '/' represents a sub-page in the url. So here we are calculating the sub-pages within a url.

6. Redirection "//" in URL.

```
def redirection(url):
 pos = url.rfind('//')
 if pos > 6:
  if pos > 7:
   return 1
  else:
   return 0
 else:
  return 0
```

The '//' will redirect the user to some other website. So '//' in the url indicates the url might be phishing.

7. "HTTP/HTTPS" in Domain Name.

```
[20] #http/https in domain name
     def httpDomain(url):
       domain = urlparse(url).netloc
       if 'https' in domain:
         return 1
       else:
         return 0
```

If https is found in the URL then the URL might be phishing.

8. **Using URL Shortening Services "TinyURL".**

```
#Checking if the url uses tinyUrl services

short_url_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"
def usesTinyUrl(url):
 temp = re.search(short_url_services, url)
 if(temp):
  return 1
 else:
  return 0
```

Phishing websites uses tiny URL services. So the presence of a tiny URL indicates that the URL might be phishing.

9. **Prefix or Suffix "-" in Domain**

```
[22] #Prefix or sufix in domain
     def prefixSuffix(url):
         if '-' in urlparse(url).netloc:
             return 1
         else:
             return 0
```

Prefix or suffixes are added, separated by a hyphen (-) to the original domain name fooling the user that it's a legitimate URL.

6.2 Extraction of Domain Based Features

Many features are extracted from the URL falls under Domain Based Features which are as follows:

1. **DNS record**
   - It is widely used for querying databases which store the registered users or assignees of internet resource, such as domain name, IP address block, or an autonomous system.
   - If the DNS record is empty or not found, the value assigned to this feature is 1(Phishing), 0(Legitimate).

2. **Website Traffic**

```
def web_traffic(url):
 domain = getDomain(url)
 try:
  rank = [i for i, v in enumerate(alexa) if v[1] == domain][0] + 1
 except:
  return 0
 if rank <100000:
  return 1
 else:
  return 0
```

   - This aspect assesses the popularity of the website by specifying the number of novices and the number of pages they visit.
   - However, since phishing websites live for a short period, they may not be recognized by the Alexa database.
   - Furthermore, if the domain has no traffic or is not recognized by the Alexa database, it is classified as “Phishing”.

3. **Age of Domain**
   - This feature can also be extracted from the WHOIS database.
   - Most Phishing websites live for a short period of time.
   - We have considered 12 month as minimum age of domain therefore, if age of domain >12 months value assigned to this feature is 1(Phishing) else 0(Legitimate).

4. **End Period of Domain**
   - This feature can also be extracted from WHOIS database.
   - End time of domain : Difference b/w termination time and current time
   - If end period of domain > 6 months the value of this feature is 1(Phishing) or 0(Legitimate).

6.3 HTML & JAVASCRIPT BASED FEATURES

It is a page source code-based feature. It includes URLs embedded in the webpage and HTML and JAVASCRIPT. It falls under the category of Content Based-Approach method for detection of phishing Under ad-hoc methods.

1. Iframe Redirection

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

- Webpage which is replica of the current webpage.
- Phishers use frameborder attribute to render visual portrayal.
- If the Iframe is empty or the response is not found then, the value assigned to this feature is 1(Phishing) or else 0(Legitimate).

2. Status Bar Customization

```
def mouseOver(response):
  if response == "" :
    return 1
  else:
    if re.findall("<script>.+onmouseover.+</script>", response.text):
      return 1
    else:
      return 0
```

- This feature is extracted by exploring the webpage source code & finding the onMouseOver event.
- After finding it monitor it on status bar say if it makes any changes or not.
- If the onMouseOver is found then the value assigned will be 1(Phishing) or else 0(legitimate)

3. Website Forwarding

```
def forwarding(response):
  if response == "":
    return 1
  else:
    if len(response.history) <= 2:
      return 0
    else:
      return 1
```

- Through our dataset we came to knew that legitimate websites have been redirected one time max.
- Phishing website compared to legitimate has been redirected 4 times.
- If the redirect is greater then or equal to 2 value assigned to the feature is 1(Phishing) or else 0(Legitimate).

6.4 Computing URL Features

We have created different function for all the features. For a single website we have to calculate all these features. So, we have created another function which calls all the other functions. There might be some issues which might occur while iterating through the websites. Some websites might get stuck when the server doesn't respond or it may also get stuck whenever the server returns status code other than 400. This was the major problem which we faced while extracting our features. So, we have introduced timeout timer and checking the status codes of the website.

```
try:
  response = requests.get(url, timeout=5 )
  print('HTTP response code: ', response.status_code)
  if response.status_code == 200:
    features.append(iframe(response))
    features.append(mouseOver(response))
    features.append(rightClick(response))
    features.append(forwarding(response))
    features.append(label)
  else:
    print('Not reachable - ', url)
    features.extend(temp)
except:
  print('Timeout - ', url)
  features.extend(temp)
```

After this we ran the **featureExtraction()** for legitimate and phishing websites.

Legitimate URLs.

Phishing URLs.

```
Streaming output truncated to the last 5000 lines.
Timeout -  https://fineiron.pk/n26-support/n26_fr-m3tri-withcurlnoantibots/n26_fr-m3tri/Error.php?hash=NjYxMDQ0NjkwMjc0MQ==&token=TW96aWxsYS81LjAgKFgxMTsgTGlu
i is:  3488 url:  http://distractify.com/post/related/id/5552931a4a0c4b5b10d5da74/skip/30/limit/10/back/0
jbshtl.secure52serv.com
Error trying to connect to socket: closing socket
HTTP response code:  404
Not reachable -  https://jbshtl.secure52serv.com/receipt/secureNetflix/085e4dd9bf9b42f434fbc5482c404ce6/
i is:  3489 url:  http://olx.ro/i2/electronice-si-electrocasnice/electrocasnice/electronice-si-electrocasnice/electrocasnice
surveygizmolibrary.s3.amazonaws.com
HTTP response code:  403
Not reachable -  https://surveygizmolibrary.s3.amazonaws.com/library/717354/ch.html
i is:  3490 url:  http://sourceforge.net/directory/audio-video/add_facet_filter?facet=developmentstatus&constraint=4+-+Beta
www.samakav.net
HTTP response code:  404
Not reachable -  http://www.samakav.net/wv/
i is:  3491 url:  http://mashable.com/category/international-space-station/2014/10/14/chris-hadfield-space-photos-book
linktr.ee
HTTP response code:  200
i is:  3492 url:  http://allegro.pl/ciaza-i-macierzynstwo-78013?limit=180&order=m&string=%7Bstring%7D&bmatch=seng-v10-p-sm-pers-isqm-chl-moda-0414
amazon-co-jp.youtian8356.vip
Error trying to connect to socket: closing socket
HTTP response code:  200
```

6.5 Understanding the dataset & finding insights

We are familiarizing ourselves with data and some data frame methods which are used into the data & its features.

```
[ ]  data.columns
```

```
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')
```

```
[ ]  data.info
```

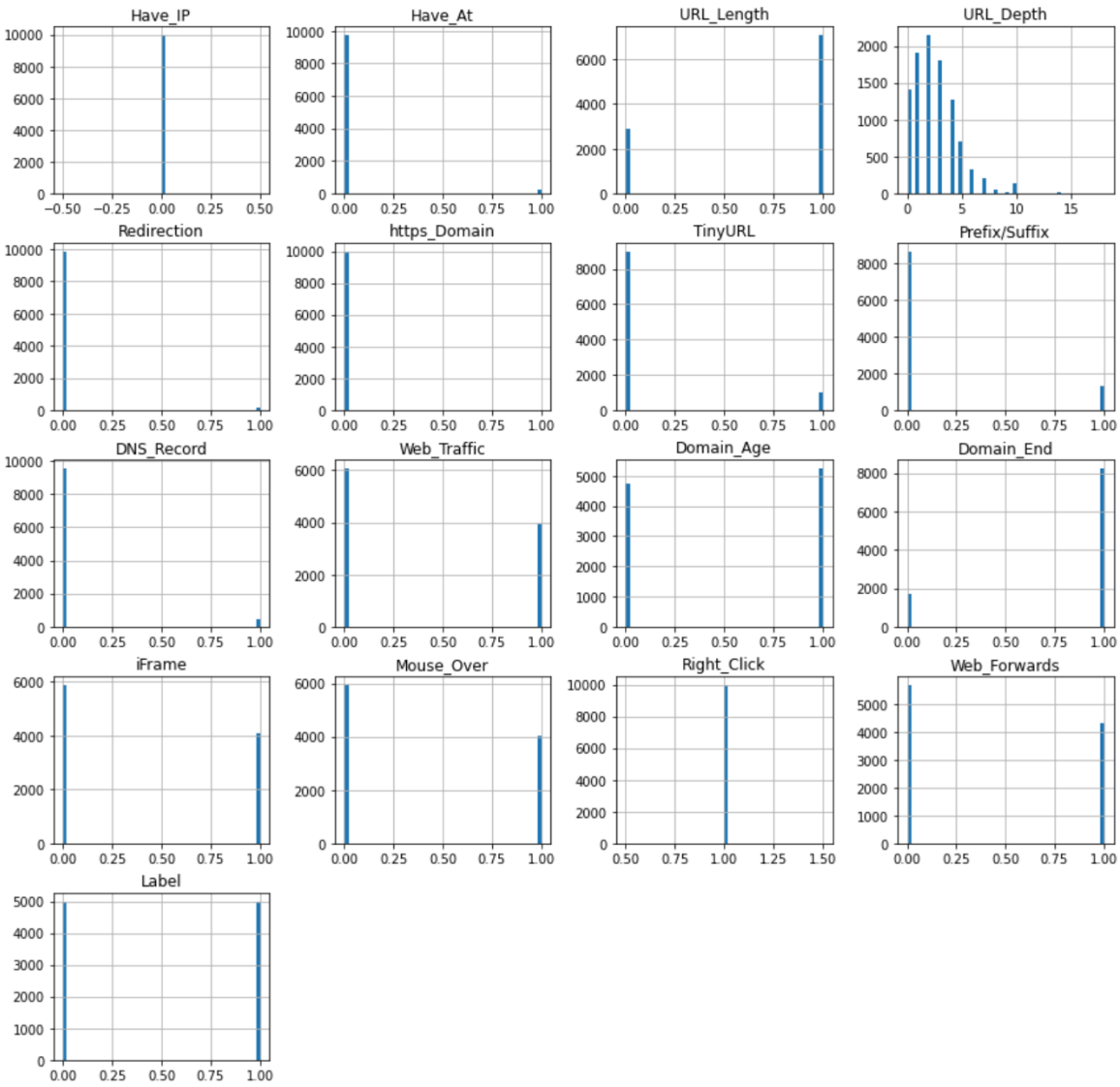
```
<bound method DataFrame.info of                                          Domain  Have_IP  ...  Web_Forwards  Label
0                                graphicriver.net        0  ...             0      0
1                                       ecnavi.jp        0  ...             0      0
2                                   hubpages.com        0  ...             0      0
3                                 extratorrent.cc        0  ...             1      0
4                                   icicibank.com        0  ...             0      0
...                                           ...      ...  ...           ...    ...
9995                  dg54asdg15g1.agilecrm.com        0  ...             0      1
9996                                    osmaslo.ru        0  ...             0      1
9997             firebasestorage.googleapis.com        0  ...             1      1
9998  free-balance-tokenwallet.000webhostapp.com        0  ...             1      1
9999                                    linktr.ee        0  ...             0      1

[10000 rows x 18 columns]>
```

In the above code we have reviewed the shape of the dataset and summarized the features of the dataset

We have plotted the data distribution for each and every feature to find how the features are related to each other.

```
data.hist(bins = 50,figsize = (15,15))
plt.show()
```



Example: let us consider the Web Traffic plot, as you can see from above 6000 URL' s from the dataset are legitimate while 4000 URL's are Phishing.

Now we are trying to apply data preprocessing techniques and altering the data for the procedure of ML models, here most of the data is made of 0's & 1's except 'Domain' & 'URL Depth' Columns.

```
data.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
count	10000.0	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0	10000.000000	10000.000000
mean	0.0	0.022800	0.711500	2.582100	0.011700	0.000800	0.099600	0.132200	0.04270	0.390900	0.525900	0.826600	0.410100	0.404800	1.0	0.431100	0.500000
std	0.0	0.149273	0.453087	2.059243	0.107537	0.028274	0.299481	0.338725	0.20219	0.487976	0.499354	0.378612	0.491876	0.490878	0.0	0.495255	0.500025
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	1.0	0.000000	0.000000
25%	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000	0.000000	0.000000	1.0	0.000000	0.000000
50%	0.0	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	1.0	0.000000	0.500000
75%	0.0	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0	1.000000	1.000000
max	0.0	1.000000	1.000000	18.000000	1.000000	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0	1.000000	1.000000

We are dropping the domain column because it does not possess any significance with respect to machine learning training model.

```
[ ]  data = data.drop(['Domain'], axis = 1).copy()
```

We are checking the data for null/missing values, features of both were concatenated and no shuffling was done resulting 5000 legitimate and 5000 phishing URL's dataset, for training testing & splitting there is a requirement of even distribution, so the shuffle is performed which results into evading of overfitting during the model training.

```
[ ] data = data.sample(frac=1).reset_index(drop=True)
    data.head()
```

	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	3	0	0	0	0	0	0	0	1	1	1	1	1	0
1	0	0	1	4	0	0	0	0	0	0	1	1	0	0	1	0	1
2	0	0	0	0	0	0	1	0	1	0	1	1	0	0	1	0	1
3	0	0	1	4	0	0	0	0	0	1	0	1	0	0	1	0	0
4	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	1

The data does not have any missing values & it is preprocessed, ready for training.

Now we Splitting the data, we have separated & assigned features with target columns X&y. The data is split into train & test of 80-20(split).

```
[ ]  y = data['Label']
     X = data.drop('Label',axis=1)
     X.shape, y.shape
```

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

```
[ ]  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
```

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

6.6 Applying different ML Algorithms on the final dataset

The data set comes under classification problem therefore it is a supervised ml task, as the input URL is classified as phishing(1) or legitimate(0).

1. Decision Tree

- 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.

```
[ ] from sklearn.tree import DecisionTreeClassifier
    tree = DecisionTreeClassifier(max_depth = 5)
    tree.fit(X_train, y_train)
```

```
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')
```

```
[ ] y_test_tree = tree.predict(X_test)
    y_train_tree = tree.predict(X_train)
```

```
[ ] 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: 0.904
Decision Tree: Accuracy on test Data: 0.914
```

```
[ ] 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()
```

- 

2. Random Forest

- Random forests for regression and classification are currently among the most widely used machine learning methods. 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.

```
[ ] from sklearn.ensemble import RandomForestClassifier
    forest = RandomForestClassifier(max_depth=5)
    forest.fit(X_train, y_train)
```

```
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)
```

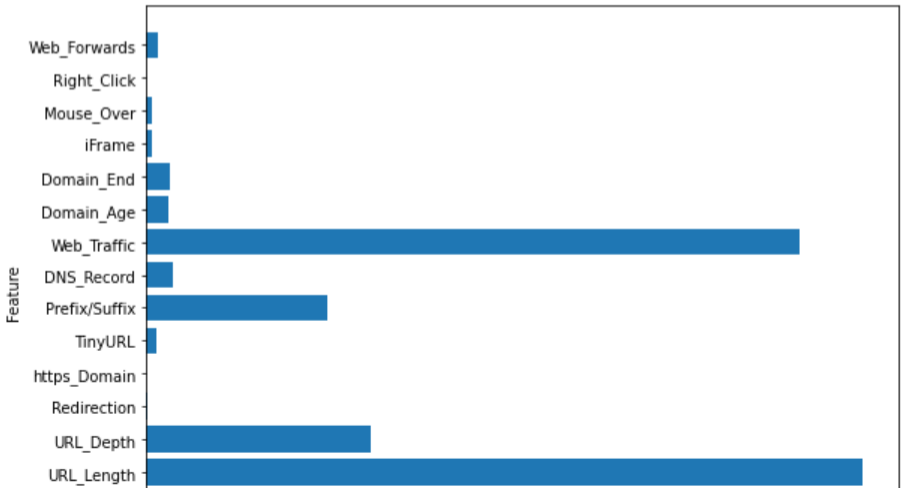
```
[ ] y_test_forest = forest.predict(X_test)
    y_train_forest = forest.predict(X_train)
```

```
[ ] 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: 0.920
Random forest: Accuracy on test Data: 0.926
```

```
[ ] 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()
```

- 

3. 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.

```
[ ]  from xgboost import XGBClassifier
     xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
     xgb.fit(X_train, y_train)
```

```
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)
```

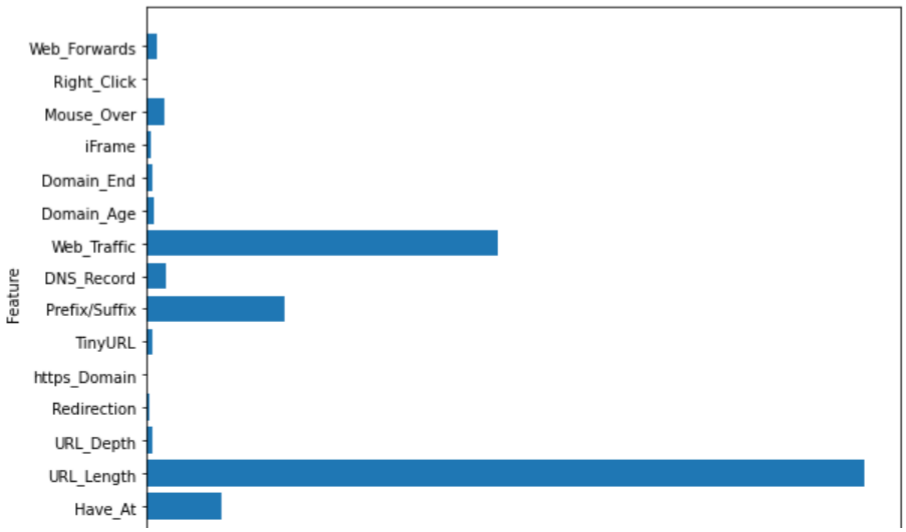
```
[ ]  y_test_xgb = xgb.predict(X_test)
     y_train_xgb = xgb.predict(X_train)
```

```
[ ]  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: 0.954
XGBoost : Accuracy on test Data: 0.951
```

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



4. 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.

SUPPORT VECTOR MACHINES(SVM)

```
[ ]  svm = SVC(kernel='linear', C=1.0, random_state=12)
     svm.fit(X_train, y_train)
```

```
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)
```

```
[ ]  y_test_svm = svm.predict(X_test)
     y_train_svm = svm.predict(X_train)
```

```
[ ]  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: 0.924
SVM : Accuracy on test Data: 0.927
```

```
[ ]  storeResults('SVM', acc_train_svm, acc_test_svm)
```

-

6.7 Selection of Best Model

- We have created a dataframe and sorted the dataframe on accuracy.

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

```
[ ]  results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=True)
```

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.904	0.914
1	Random Forest	0.920	0.926
3	SVM	0.924	0.927
2	XGBoost	0.954	0.951

```
[ ]  import pickle
     pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))
```

```
[ ]  loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
     loaded_model
```

```
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)
```

- The XGBoost Classifier works well with this dataset so we are saving & then testing the saved model.

CHAPTER: 7 CONCLUSION AND FUTURE WORK

CHAPTER 7 CONCLUSION AND FUTURE WORK

Conclusion

- Phishing is the most commonly used social engineering and cyber attack
- Through such attacks, the phisher targets naïve online users by tricking them into revealing confidential information, with the purpose of using it fraudulently.
- In order to avoid getting phished,
    - users should have awareness of phishing websites.
    - have a blacklist of phishing websites which requires the knowledge of website being detected as phishing.
    - detect them in their early appearance, using ML algorithms
- We have proposed a model that automatically extracts important features for phishing website detection without requiring any human intervention
- This project will help the naïve users to avoid getting tricked by phishing websites.

Future work

- After implementing all these algorithms, select the one which has best accuracy.
- Refactor the whole project into one neat file.
- Choose a platform to deploy the model among GCP, Heroku, Azure, AWS lambda etc.
- If the time permits, a small browser extension for the project.
- The user just have to input the url and will get the results

CHAPTER: 8 REFERENCES

CHAPTER 8 REFERENCES

1. https://stackoverflow.com/questions/52588552/google-co-laboratory-notebook-pdf-download/54191922#54191922
2. http://eprints.hud.ac.uk/id/eprint/24330/6/MohammadPhishing14July2015.pdf
3. https://stackoverflow.com/questions/49721695/python-requests-get-gets-stuck
4. https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5
5. https://machinelearningmastery.com/save-gradient-boosting-models-xgboost-python/
6. https://mc.ai/a-beginners-guide-to-build-stacked-autoencoder-and-tying-weights-with-it/
7. https://blog.keras.io/building-autoencoders-in-keras.html