WEB PHISHING DETECTION USING MACHINE LEARNING

1.INTRODUCTION

Phishing has become a major source of concern for security professionals in recent years since it is relatively easy to develop a phoney website that appears to be identical to a legitimate website.

Although experts can detect bogus websites, not all users can, and as a result, they become victims of phishing attacks. The attacker's main goal is to steal bank account credentials. Because of a lack of user awareness, phishing assaults are becoming more successful. Because phishing attacks take advantage of user flaws, it is difficult to mitigate them, but it is critical to improve phishing detection tools. Phishing is a type of wide extortion in which a malicious website imitates a genuine one-time memory with the sole purpose of obtaining sensitive data, such as passwords, account details, or MasterCard numbers. Despite the fact that there are still some anti-phishing programming and strategies for detecting possible phishing attempts in messages and typical phishing content on websites, phishes devise fresh and crossbred procedures to get around public programming and frameworks. Phishing is a type of fraud that combines social engineering with access to sensitive and personal data, such as passwords and open-end credit unpretentious components by assuming the characteristics of a trustworthy person or business via electronic correspondence. Hacking uses spoof messages that appear legitimate and are instructed to originate from legitimate sources such as financial institutions, online business goals, and so on, to entice users to visit phoney destinations via links provided on phishing websites.

1.1 PROJECT REPORT

This section describes the proposed model of phishing attack detection. The proposed model focuses on identifying the phishing attack based on checking phishing websites features, Blacklist and WHOIS database. According to few selected features can be used to differentiate between legitimate and spoofed web pages. These selected features are many such as URLs, domain identity, security & encryption, source code, page style and contents, web address bar and social human factor. This study focuses only on URLs and domain name features. Features of URLs and domain names are checked using several criteria such as IP Address, long URL address, adding a prefix or suffix, redirecting using the symbol “//”, and URLs having the symbol “@”.These features are inspected using a set of rules in order to distinguish URLs of phishing webpages from the URLs of legitimate websites.

1.2 PURPOSE

At first the data sets is created using the information collected from the various sources. Once the data set is created, this data set is fed to K Means clustering algorithm and the model is trained using this data set. A web application is developed a front end GUI is created using HTML, CSS and simple JAVA script code and the model that is trained with the data sets that are created acts as a back end server.

When phishing URL is fed to the model, the model analyses the URL that is fed and gives the appropriate output. Once the machine learning model analyses the given URL, it sends a message to the front end portal whether it is a legitimate site or a phishing site.

2. LITERATURE SURVEY

Smithi Poddar;Harsh Salkar; Priya Agarwal; MilindParaye; Dayanand Ambawade; Narendra Bhagat	2022	PhishGuard- an automatic web phishing detection system	It provide an intelligent system for identifying phishing web sites that works an	

T. Nathezhtha; D. Sangeetha; V .Vaidehi	2019	WCPAD: Web Crawling basedPhishing AttackDetection	It takes the web traffics, web content and Uniform Resource Locator(URL) as input features, based on these features classification of phishing and non phishing websites are done. The experimental analysis of the proposed WC- PAD is done with datasets collected fromreal phishing cases	phishing detection approaches fails to providesolution to problem like zero-day phishing website attacks

B.RaviRaju,S.Sai likitha,N.Deepa, S.Sushma	2017	Phishing websites detection using machine learning	Machine learning is an effective method for combating phishing assaults. This study examines the features utilized in detection as well as machine learning based detection approaches.	

2.1 EXISTING PROBLEM

In recent years, with the increasing use of mobile devices, there is a growing trend to move almost all real-world operations to the cyber world. Although this makes easy our daily lives, it also brings many security breaches due to the anonymous structure of the Internet. Used antivirus programs and firewall systems can prevent most of the attacks. However, experienced attackers target on the weakness of the computer users by trying to phish them with bogus web pages. These pages imitate some popular banking, social media, e-commerce, etc. sites to steal some sensitive information such as, user-ids, passwords, bank account, credit card numbers, etc. Phishing detection is a challenging problem, and many different solutions are proposed in the market as a blacklist, rule-based detection, anomaly-based detection, etc. In the literature, it is seen that current works tend on the use of machine learning-based anomaly detection due to its dynamic structure, especially for catching the “zero-day” attacks. In this paper, we proposed a machine learning-based phishing detection system by using eight different algorithms to analyze the URLs, and three different datasets to compare the results with other works. The experimental results depict that the proposed models have an outstanding performance with a success rate.

2.2 REFERENCE

1] G. Canbek, \"A Review on Information, Information Security, and Security Processes,\" Politek. Derg., vol. 9, no. 3, 2006, pp. 165–174.

[2] IET Inf. Secur., vol. 8, no. 3, pp. 153–160, 2014. L. Mc Cluskey, F. Thabtah, and R. M. Mohammad, \"Intelligent rule based phishing websites classification,\" IET Inf. Secur., vol. 8, no. 3, pp. 153–160, 2014.

[3] \"Predicting phishing websites using a self-structuring neural network,\" Neural Computer. Appl.,vol. 25, no. 2, pp. 443 –458,2014. R. M. Mohammad, F. Thabtah, and L. McCluskey, \"Predicting phishing websites using a self-structuring neural network,\" Neural Computer Appl.

[4] \"A new fast associative classification method for identifying phishing websites, \" Appl. Soft Computer. J.,vol. 48, pp. 729–734, 2016. W. Hadi, F. Aburub, and S. Alhawari, \"A new fast associative classification algorithm for detecting phishing websites,\" Appl. Soft Computer.

[5] \"Multi-label rules for phishing classification,\" Appl. Computer Informatics, vol. 11, no. 1, pp. 29–46, 2015.

2.3 PROBLEM STATEMENT DEFINITION

Phishing attacks succeed when human users fail to detect phishing sites. Generally speaking, past work in anti-phishing falls into four categories: studies to understand why people fall for phishing attacks, methods for training people not to fall for phishing attacks, user interfaces for helping people make better decisions about rusting email and websites, and automated tools to detect phishing [4]. Our work describes an automated approach to detect phishing. Most of the end user normally takes decision only based on what he/she look and feel. When a user is accessing internet he/she only see the screen of a browser. He/she then work on the command of a web-page.

The user doesn‘t concern about the back end process and most phishing attempts get this type of unintentional opportunity given by the user and make them fool.

3.1 EMPATHY MAP

3.2 IDEATION & BRAINSTORMING

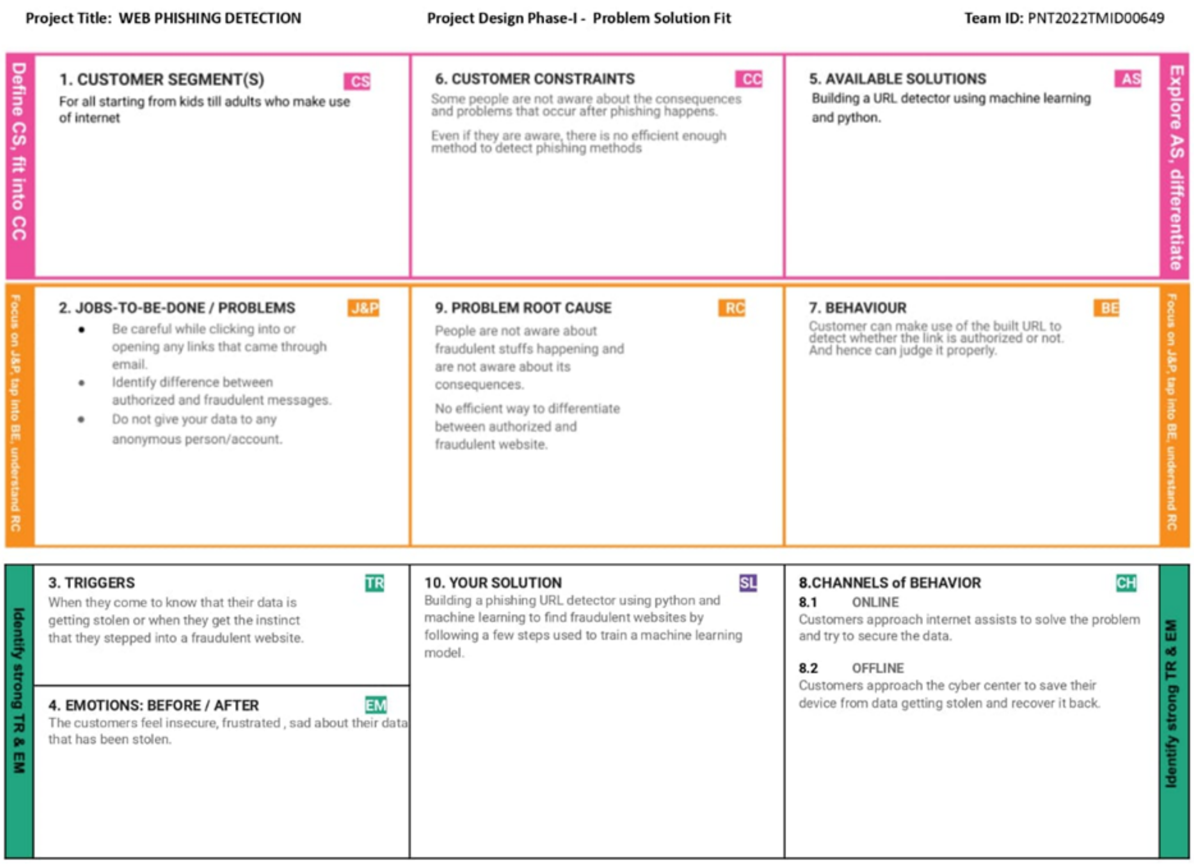
3.3 PROPOSED SOLUTION

S.NO	PROBLEM STATEMENT	DESCRIPTION
1	Problem Statement (Problem to bes	Phishingis 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 posesas 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 victimgets

		“hacked”. Hence, Problemto be solvedis: Detection of malicious websites ransomware; Identify,block, and targeted threats.
2	Idea / Solution description	Phishing is popular since it is a l high rewardattack. To build phis detector using Python andmach learning
3	Novelty / Uniqueness	To come up with effective feature engineeringtechniques to evaluate the given URL's authenticity. Identify phishing URLs, build and train a simple decisiontree model to evaluate any givenURL, and indicate whether it is actually valid or no

4	Social Impact/ Customer Satisfaction	To spread the cyber awareness on multiple attacks mainly on this phishing attack. An individual can unlearn and relearnthis model in various types of aspects in Cybersecurity and Data theft. Add great value in terms of security to organization. The major objective is to identify phishing websites and safeguard user information from phishing to protect users' privacy.
5	Business Model(Revenue Model)	In Business Organization, can use this tool to get rid from cyber attack and can implement how to improve the security when this attack occurs next time. Will be great business to the org that prefershigh security.
6	Scalability of the Solution	Machine Learning models and effective feature engineering techniques helps identify phishing websitesand come up with key features that are common in most phishing websites. The model is tested and trained in multiple types of datasets to get high accuracy than other algorithms.

3.4 PROBLEM SOLUTION FIT



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Quality data and preprocessdata	A systematic review of current trends in web phishing detection techniques is carried out and ataxonomy of automated web phishing detection ispresented. The objective of this study is to acknowledge the status of current research in automated web phishing detection and evaluate theirperformance.
FR-2	Accurately predict	Although scientifically there is reliable method ofpredicting the range of length that justify a websiteasphishing or non-phishing but then it is criteria used
FR-3	Confirmation	Display the result with the description of WebPhishing Detection.

4.2 NON FUNCTIONAL REQUIREMENTS

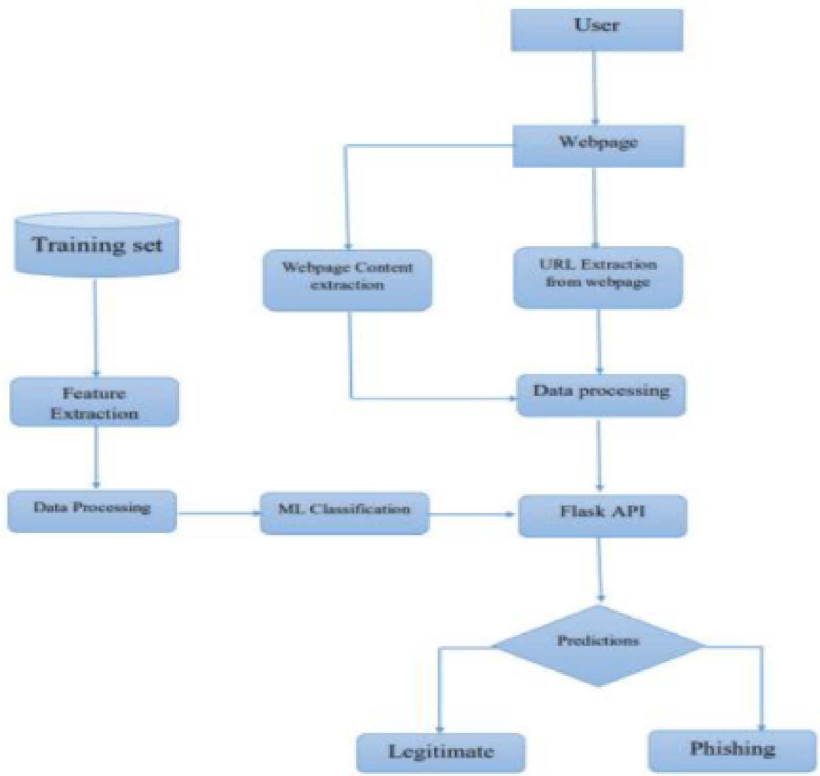
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Phishing is aform of crimein which identity theft is accomplished by use of deceptive electronicmail and a fake site on the World Wide Web.

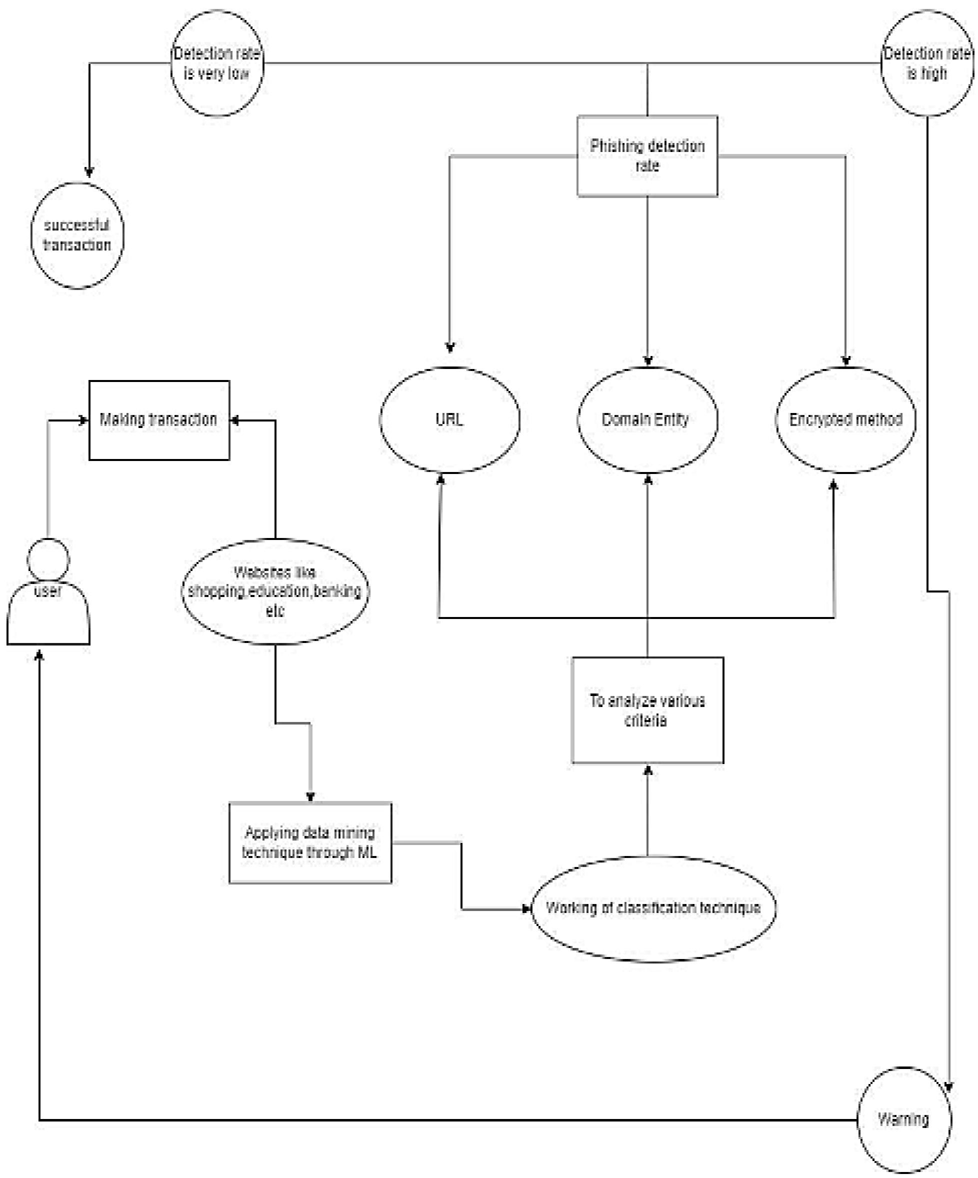
NFR-2	**Security**	Web phishing is one of many security threats toweb services on the Internet. Web phishing aims to steal private information, such as usernames, passwords, and credit card details,by way of impersonating a legitimate entity. It will leadto information disclosure and property damage.
NFR-3	**Reliability**	The initial dataset for phishing websites was obtained from a community website called PhishTank. An accuracy detection rate of about 99% was achieved.
NFR-4	**Performance**	Web phishing aims to steal private information, such as usernames, passwords, and credit carddetails, by way of impersonating a legitimate entity.
NFR-5	**Availability**	The system uses to find the framework thattracks websites for phishing sites.
NFR-6	**Scalability**	Framework is fit for taking careof increment allout throughput under an expanded burden when assets (commonly equipment) are included. Framework can work ordinarily under circumstances, for example, low data transfercapacity and substantial number of clients.

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 SOLUTION AND TECHNICAL ARCHITECTURE



5.3 USER STORIES

User Stories

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

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Web user)	User input	USN-1	As a user, I can input the particular URL in the required field and wait for validation.	I can go access the website without any problem	High	Sprint-1
Customer Care Executive	Feature extraction	USN-1	After I compare in case if none found on comparison, then we can extract feature using heuristic and visual similarity approach.	As a User I can have comparison between websites for security.	High	Sprint-1
Administrator	Prediction	USN-1	Here the Model will predict the URL websites using Machine Learning algorithms such as Logistic Regression, KNN	In this I can have correct prediction on the particular algorithms	High	Sprint-1
	Classifier	USN-2	Here I will send all the model output to classifier in order to produce final result.	In this I will find the correct classifier for producing the result	Medium	Sprint-2

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Project Planning Phase

Project Planning Template (Product Backlog,
Sprint Planning, Stories, Story points)

Date	22 October 2022
Team ID	PNT2022TMID00649
Project Name	Web Phishing Detection
Maximum Marks	8 Marks

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team members
Sprint-1	User Input	USN-1	User input an URL in the required field to check their details.	1	Medium	Preethi Aparna Gayathri Hinduja
Sprint-1	Website comparison	USN-2	Model compare the websites using blacklist and whitelist approach.	1	High	Preethi Aparna Gayathri Hinduja
Sprint-2	Feature Extraction	USN-3	After comparison, if none found on comparison then it extract feature using heuristic andvisual similarity.	2	High	Preethi Aparna Gayathri Hinduja
Sprint-2	Prediction	USN-4	Model predicts the URL using machine learning algorithms such as	2	Medium	Preethi Aparna Gayathri Hinduja

			logistic regression, KNN.			
Sprint-3	Classifier	USN-5	Model sends all the output to the classifier and produces the final result.	1	Medium	Preethi Aparna Gayathri Hinduja
Sprint-4	Announcement	USN-6	Model then displays whether the website is legal site or phishing site.	1	High	Preethi Aparna Gayathri Hinduja
Sprint-5	Events	USN-7	The model needs the capability of retrieving and displaying accurate result for the website.	1	High	Preethi Aparna Gayathri Hinduja

6.2 SPRINT DELIVERY SCHEDULE

Project Planning Phase

Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

Date	22 October 2022
Team ID	PNT2022TMID00649
Project Name	Web Phishing Detection
Maximum Marks	8 Marks

Project Tracker, Velocity &Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	08 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	10 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	14 Nov 2022

Velocity:

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

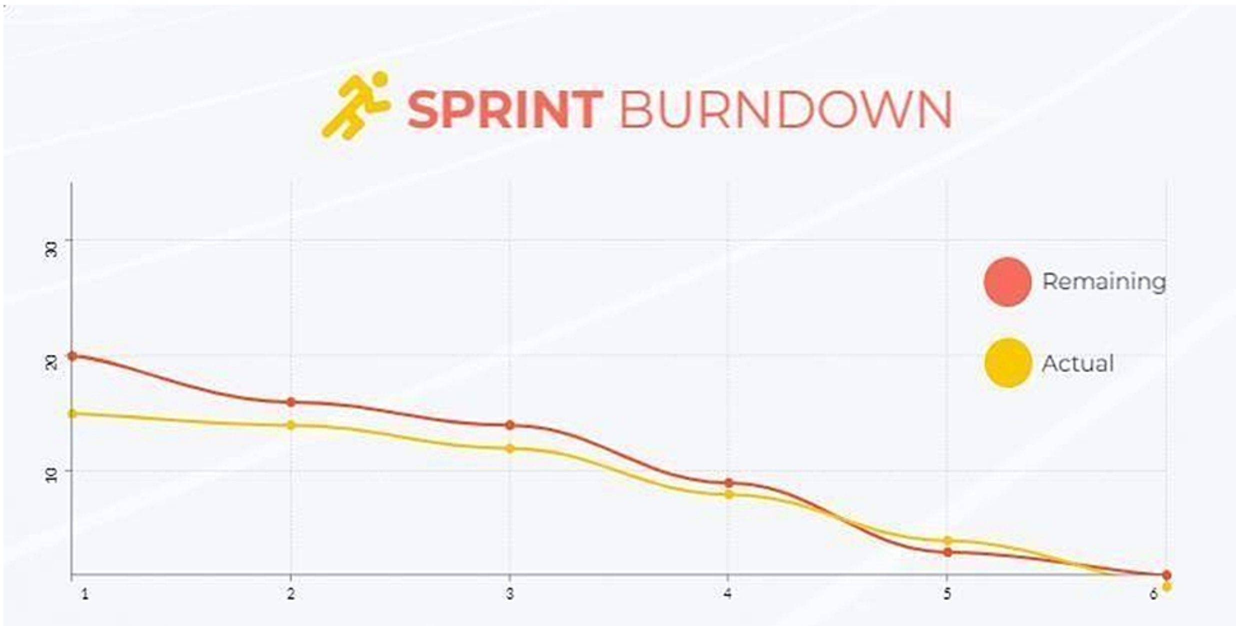
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

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

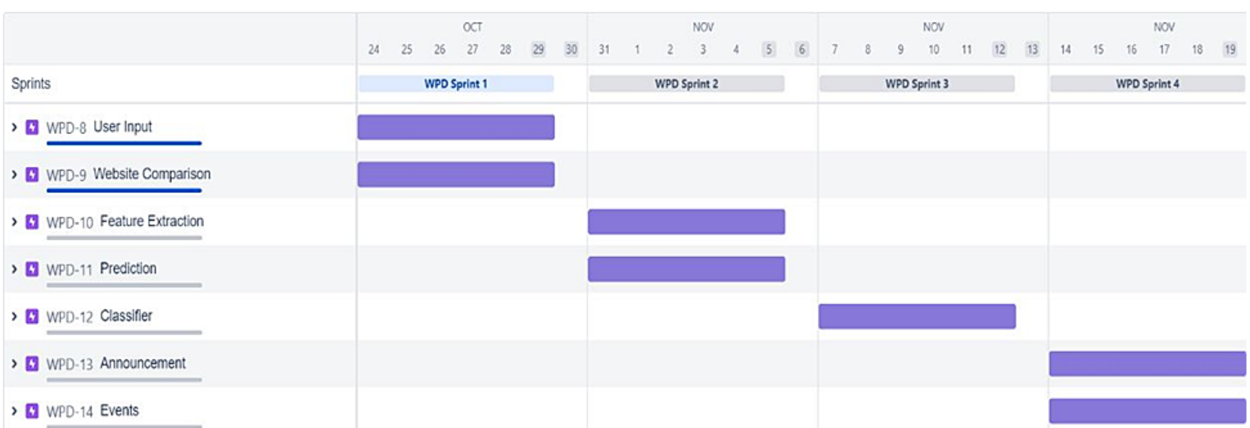
AV = (Sprint Duration / Velocity) = 20 /6 = 3.33

Burndown Chart:

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



6.3 REPORTS FROM JIRA



7. CODING AND SOLUTIONING

7.1 FEATURE 1

```
#importing required libraries

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

```
#Loading data into dataframe

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

Familiarizing with Data & EDA:

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

```
#Shape of dataframe

data.shape
```

```
#Listing the features of the dataset

data.columns
#Information about the dataset
```

```
data.info()
# nunique value in columns

data.nunique()

#droping index column

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

#description of dataset

data.describe().T
```

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.

```
#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 Data:

The data is split into train & test sets, 80-20 split.

```
# 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
```

7.2 FEATURE 2

Model Building & Training:

```
# 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))
```

Logistic Regression

```
# Linear regression model
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log = LogisticRegression()

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

#predicting the target value from the model for the samples

y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)

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

acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print("Logistic Regression : Accuracy on training Data:
{:.3f}".format(acc_train_log))
print("Logistic Regression : Accuracy on test Data:
```

```
{:.3f}".format(acc_test_log))
print()

f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print("Logistic Regression : f1_score on training Data:
{:.3f}".format(f1_score_train_log))
print("Logistic Regression : f1_score on test Data:
{:.3f}".format(f1_score_test_log))
print()

recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print("Logistic Regression : Recall on training Data:
{:.3f}".format(recall_score_train_log))
print("Logistic Regression : Recall on test Data:
{:.3f}".format(recall_score_test_log))
print()

precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print("Logistic Regression : precision on training Data:
{:.3f}".format(precision_score_train_log))
print("Logistic Regression : precision on test Data:
{:.3f}".format(precision_score_test_log))

#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_log))
#storing the results. The below mentioned order of parameter passing is
important.

storeResults('Logistic Regression',acc_test_log,f1_score_test_log,
recall_score_train_log,precision_score_train_log)
```

K-Nearest Neighbors : Classifier

```
# K-Nearest Neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)

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

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

```
y_test_knn = knn.predict(X_test)

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

acc_train_knn = metrics.accuracy_score(y_train,y_train_knn)
acc_test_knn = metrics.accuracy_score(y_test,y_test_knn)
print("K-Nearest Neighbors : Accuracy on training Data:
{:.3f}".format(acc_train_knn))
print("K-Nearest Neighbors : Accuracy on test Data:
{:.3f}".format(acc_test_knn))
print()

f1_score_train_knn = metrics.f1_score(y_train,y_train_knn)
f1_score_test_knn = metrics.f1_score(y_test,y_test_knn)
print("K-Nearest Neighbors : f1_score on training Data:
{:.3f}".format(f1_score_train_knn))
print("K-Nearest Neighbors : f1_score on test Data:
{:.3f}".format(f1_score_test_knn))
print()

recall_score_train_knn = metrics.recall_score(y_train,y_train_knn)
recall_score_test_knn = metrics.recall_score(y_test,y_test_knn)
print("K-Nearest Neighborsn : Recall on training Data:
{:.3f}".format(recall_score_train_knn))
print("Logistic Regression : Recall on test Data:
{:.3f}".format(recall_score_test_knn))
print()

precision_score_train_knn = metrics.precision_score(y_train,y_train_knn)
precision_score_test_knn = metrics.precision_score(y_test,y_test_knn)
print("K-Nearest Neighbors : precision on training Data:
{:.3f}".format(precision_score_train_knn))
print("K-Nearest Neighbors : precision on test Data:
{:.3f}".format(precision_score_test_knn))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_knn))

training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
    knn = KNeighborsClassifier(n_neighbors=n)

    knn.fit(X_train, y_train)
```

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


#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend();
#storing the results. The below mentioned order of parameter passing is
important.

storeResults('K-Nearest Neighbors',acc_test_knn,f1_score_test_knn,
             recall_score_train_knn,precision_score_train_knn

# Support Vector Classifier model
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'gamma': [0.1],'kernel': ['rbf','linear']}

svc = GridSearchCV(SVC(), param_grid)

# fitting the model for grid search
svc.fit(X_train, y_train)

#predicting the target value from the model for the samples
y_train_svc = svc.predict(X_train)
y_test_svc = svc.predict(X_test)

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()
```

```
f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))


print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()
```

```
f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))


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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()
```

```
recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
```

```
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
```

```
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

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

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data:
{:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
```

```
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))

#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_svc))

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

storeResults('Support Vector Machine',acc_test_svc,f1_score_test_svc,
             recall_score_train_svc,precision_score_train_svc)

# Naive Bayes Classifier Model
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline

# instantiate the model
nb=  GaussianNB()

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

#predicting the target value from the model for the samples
y_train_nb = nb.predict(X_train)
y_test_nb = nb.predict(X_test)

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

```
acc_train_nb = metrics.accuracy_score(y_train,y_train_nb)
acc_test_nb = metrics.accuracy_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_nb))
print("Naive Bayes Classifier : Accuracy on test Data:
{:.3f}".format(acc_test_nb))
print()

f1_score_train_nb = metrics.f1_score(y_train,y_train_nb)
f1_score_test_nb = metrics.f1_score(y_test,y_test_nb)
print("Naive Bayes Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_nb))
print("Naive Bayes Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_nb))
print()

recall_score_train_nb = metrics.recall_score(y_train,y_train_nb)
recall_score_test_nb = metrics.recall_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_nb))
print("Naive Bayes Classifier : Recall on test Data:
{:.3f}".format(recall_score_test_nb))
print()

precision_score_train_nb = metrics.precision_score(y_train,y_train_nb)
precision_score_test_nb = metrics.precision_score(y_test,y_test_nb)
print("Naive Bayes Classifier : precision on training Data:
{:.3f}".format(precision_score_train_nb))
print("Naive Bayes Classifier : precision on test Data:
{:.3f}".format(precision_score_test_nb))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_svc))
#storing the results. The below mentioned order of parameter passing is
important.

storeResults('Naive Bayes Classifier',acc_test_nb,f1_score_test_nb,
             recall_score_train_nb,precision_score_train_nb)
```

Decision Trees : Classifier

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

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

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

#predicting the target value from the model for the samples

y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)

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

acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.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))
print()

f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print("Decision Tree : f1_score on training Data:
{:.3f}".format(f1_score_train_tree))
print("Decision Tree : f1_score on test Data:
{:.3f}".format(f1_score_test_tree))
print()

recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print("Decision Tree : Recall on training Data:
{:.3f}".format(recall_score_train_tree))
print("Decision Tree : Recall on test Data:
{:.3f}".format(recall_score_test_tree))
print()

precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print("Decision Tree : precision on training Data:
{:.3f}".format(precision_score_train_tree))
```

```
print("Decision Tree : precision on test Data:
{:.3f}".format(precision_score_test_tree))

#computing the classification report of the model


print(metrics.classification_report(y_test, y_test_tree))

training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 30
depth = range(1,30)
for n in depth:
    tree_test = DecisionTreeClassifier(max_depth=n)

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


#plotting the training & testing accuracy for max_depth from 1 to 30
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('Decision Tree',acc_test_tree,f1_score_test_tree,
             recall_score_train_tree,precision_score_train_tree)

# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

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

#predicting the target value from the model for the samples
```

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

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

acc_train_forest = metrics.accuracy_score(y_train,y_train_forest)
acc_test_forest = metrics.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))
print()

f1_score_train_forest = metrics.f1_score(y_train,y_train_forest)
f1_score_test_forest = metrics.f1_score(y_test,y_test_forest)
print("Random Forest : f1_score on training Data:
{:.3f}".format(f1_score_train_forest))
print("Random Forest : f1_score on test Data:
{:.3f}".format(f1_score_test_forest))
print()

recall_score_train_forest = metrics.recall_score(y_train,y_train_forest)
recall_score_test_forest = metrics.recall_score(y_test,y_test_forest)
print("Random Forest : Recall on training Data:
{:.3f}".format(recall_score_train_forest))
print("Random Forest : Recall on test Data:
{:.3f}".format(recall_score_test_forest))
print()

precision_score_train_forest = metrics.precision_score(y_train,y_train_forest)
precision_score_test_forest = metrics.precision_score(y_test,y_test_tree)
print("Random Forest : precision on training Data:
{:.3f}".format(precision_score_train_forest))
print("Random Forest : precision on test Data:
{:.3f}".format(precision_score_test_forest))

#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_forest))

training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
```

```python
    forest_test =  RandomForestClassifier(n_estimators=n)

    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 20
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("n_estimators")
plt.legend();

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

storeResults('Random Forest',acc_test_forest,f1_score_test_forest,
             recall_score_train_forest,precision_score_train_forest)

# 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))

#computing the classification report of the model

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)

#  catboost Classifier Model
from catboost import CatBoostClassifier

# instantiate the model
cat = CatBoostClassifier(learning_rate  = 0.1)

# fit the model
```

```
cat.fit(X_train,y_train)

#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();

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

storeResults('CatBoost Classifier',acc_test_cat,f1_score_test_cat,
             recall_score_train_cat,precision_score_train_cat)
```

XGBoost Classifier

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)

#  XGBoost Classifier Model
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier()

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

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

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

acc_train_xgb = metrics.accuracy_score(y_train,y_train_xgb)
acc_test_xgb = metrics.accuracy_score(y_test,y_test_xgb)
print("XGBoost Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_xgb))
print("XGBoost Classifier : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
print()
```

```
f1_score_train_xgb = metrics.f1_score(y_train,y_train_xgb)
f1_score_test_xgb = metrics.f1_score(y_test,y_test_xgb,average="micro")
print("XGBoost Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_xgb))
print("XGBoost Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_xgb))
print()

recall_score_train_xgb = metrics.recall_score(y_train,y_train_xgb)
recall_score_test_xgb = metrics.recall_score(y_test,y_test_xgb,average="micro")
print("XGBoost Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_xgb))
print("XGBoost Classifier : Recall on test Data:
{:.3f}".format(recall_score_train_xgb))
print()

precision_score_train_xgb = metrics.precision_score(y_train,y_train_xgb)
precision_score_test_xgb =
metrics.precision_score(y_test,y_test_xgb,average="micro")
print("XGBoost Classifier : precision on training Data:
{:.3f}".format(precision_score_train_xgb))
print("XGBoost Classifier : precision on test Data:
{:.3f}".format(precision_score_train_xgb))
#storing the results. The below mentioned order of parameter passing is
important.

storeResults('XGBoost Classifier',acc_test_xgb,f1_score_test_xgb,
             recall_score_train_xgb,precision_score_train_xgb)
```

Multi-layer Perceptron classifier

```
# Multi-layer Perceptron Classifier Model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier()
#mlp = GridSearchCV(mlpc, parameter_space)

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

#predicting the target value from the model for the samples
```

```
y_train_mlp = mlp.predict(X_train)
y_test_mlp = mlp.predict(X_test)

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

acc_train_mlp  = metrics.accuracy_score(y_train,y_train_mlp)
acc_test_mlp = metrics.accuracy_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Accuracy on training Data:
{:.3f}".format(acc_train_mlp))
print("Multi-layer Perceptron : Accuracy on test Data:
{:.3f}".format(acc_test_mlp))
print()

f1_score_train_mlp = metrics.f1_score(y_train,y_train_mlp)
f1_score_test_mlp = metrics.f1_score(y_test,y_test_mlp,average="micro")
print("Multi-layer Perceptron : f1_score on training Data:
{:.3f}".format(f1_score_train_mlp))
print("Multi-layer Perceptron : f1_score on test Data:
{:.3f}".format(f1_score_train_mlp))
print()

recall_score_train_mlp = metrics.recall_score(y_train,y_train_mlp)
recall_score_test_mlp = metrics.recall_score(y_test,y_test_mlp,average="micro")
print("Multi-layer Perceptron : Recall on training Data:
{:.3f}".format(recall_score_train_mlp))
print("Multi-layer Perceptron : Recall on test Data:
{:.3f}".format(recall_score_test_mlp))
print()

precision_score_train_mlp = metrics.precision_score(y_train,y_train_mlp)
precision_score_test_mlp =
metrics.precision_score(y_test,y_test_mlp,average="micro")
print("Multi-layer Perceptron : precision on training Data:
{:.3f}".format(precision_score_train_mlp))
print("Multi-layer Perceptron : precision on test Data:
{:.3f}".format(precision_score_test_mlp))

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

storeResults('Multi-layer Perceptron',acc_test_mlp,f1_score_test_mlp,
             recall_score_train_mlp,precision_score_train_mlp)
```

Comparision of Models

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

# dispalying total result
result

#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

#  XGBoost Classifier Model
from xgboost import XGBClassifier

# instantiate the model
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('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()
```

8. TESTING

8.1 TEST CASES

TEST CASE	FEATURE TYPE	COMPONENT	TEST SCENARIO	PRE-REQUISITE	STEPS TO EXECUTE	EXPECTED RESULTS	ACTUAL RESULT	STATUS	COMMENTS	TC FOR AUTOMATION (y/n)	BUG ID	EXECUTED BY
LoginPage_TC_OO1	Functional	Home Page	Verify user is able to see the Landing Page when user can type the URL in the bo		1.Enter local host in any of the browser 2.Type the URL 3.Verify whether it is processing or not.	Should Display the Webpage	Working as expected	Pass		N		Preethi K
LoginPage_TC_OO2	UI	Home Page	Verify the UI elements is Responsive		1. Enter any safe URL you know already and click check 2. Check whether the button is responsive or not 3. Reload and Test Simultaneously	Should Wait for Response and then gets Acknowledge	Working as expected	Pass		N		Aparna L
LoginPage_TC_OO3 F	Functional	Home page	Verify whether the link is		1. Enter URL and click go	User should	Working as	Pass		N		Gayathri M

			legitimate or not		2. Type or copy paste the URL 3. Check the website is legitimate or not 4. Observe the results	observe whether the website is legitimate or not.	expected					
LoginPage_TC_OO4	Functional	Home Page	Verify user is able to access the legitimate website or not		1. Enter any unsafe URL you know and click check 2. Check the website is legitimate or not 3. Check if the result is unsafe or not.	Application should show that Safe Webpage orUnsafe.	Working as expected	Pass		N		Hindhuja N

8.2 USER ACCEPTANCE

The purpose of this document is to briefly explain the test coverage and open issues of the [Web Phishing Detection] project at the time of the release to User Acceptance Testing

(UAT).

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

RESOLUTION	SEVERITY 1	SEVERITY 2	SEVERITY 3	SEVERITY 4	SUB TOTAL
By design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	10	2	4	20	36
Not reproduced	0	0	1	0	1
Skipped	0	0	0	0	0
Won’t fix	0	0	2	1	3
Totals	23	9	12	25	70

This report shows the number of test cases that heve passed, failed and untested.

SECTION	TOTAL CASES	NOT TESTED	FAIL	PASS

Print Engine	10	0	0	10
Client Application	50	0	0	50
Security	5	0	0	4
Outsource shipping	3	0	0	3

9. RESULTS

Phishing detection is now an area of great interest among the researchers due to its significance in protecting privacy and providing security. There are many methods that perform phishing detection by classification of websites using trained machine learning models. URL based analysis increases the speed of detection. Furthermore, by applying feature selection algorithms and dimensionality reduction techniques, we can reduce the number of features and remove irrelevant data. There are many machine learning algorithms that perform classification with good performance measures. In this paper, we have done a study of the process of phishing detection and the phishing detection schemes in the recent research literature. This will serve as a guide for new researchers to understand the process and to develop more accurate phishing detection systems.

10. ADVANTAGES & DISADVANTAGES

S.NO	TECHNIQUES	ADVANTAGES	DISADVANTAGES

	USED		
1	Methods based on Bag-of-words model	Build secure connection between users mail transfer agent (MTA) and mail user agents (MUA)	-Time consuming -Huge number of features -Consuming memory
2	Compared multi classifiers algorithm	-Provide clear idea about the effective level of each classifier on phishing email detection	Non standard classifier
3	Hybrid system	-High level of accuracy by take the advantages of many classifiers	-Time consuming because this technique has many layers to make the final result.
4	Classifiers Model-based Features	-High level of accuracy -create new type of features like Markov features	-huge number of features -many algorithm for classification which mean time consuming -higher cost -need large mail server and high memory requirement
5	Clustering of phishing email	-fast in classification process	-less accuracy because it depend on unsupervised learning, need feed continuously.
6	Evolving connectionist system(ECOS) for phishing email detection	Fast, less consuming memory, high accuracy, evolving with time, online working	Need feed continuously

7	Blacklists	-Requiring low resources on host machine -Effective when minimal FP rates are required	-Mitigation of zero-hour phishing attacks -Can result in excessive queries with heavily loaded server.
8	Heuristics and visual similarity	-Mitigate zero hour attacks.	-Higher FP rate than blacklists. -High computational cost.
9	Machine learning	-Mitigate zero hour attacks. -construct own classification models	-Time consuming. -costly -huge number of rules
10	List based approach	-This approach is 100% accurate on decision for blacklisting of website -This approach also produce less false positive rate. -It also requires less computational cost and easy to use	-It produce much memory overhead -If the websites are not in the list of blacklist then the accuracy is nil.
11	Ant colony Optimization	-This approach is accurate by determining the best rules or features. -can be used in dynamic	-It enhances false negative rate as compare to other ones.
12	Fuzzy logic	-It requires less memory -Its inference speed is also very high	-It is not 100% effective -It is complex to design

11. CONCLUSION

The proposed study emphasized the phishing technique in the context of classification, where phishing website is considered to involve automatic categorization of websites into a predetermined set of class values based on several features and the class variable. The ML based phishing techniques depend on website functionalities to gather information that can help classify websites for detecting phishing sites. The problem of phishing cannot be eradicated, nonetheless can be reduced by combating it in two ways, improving targeted anti-phishing procedures and techniques and informing the public on how fraudulent phishing websites can be detected and identified. To combat the ever evolving and complexity of phishing attacks and tactics, ML anti-phishing techniques are essential. Authors employed LSTM technique to identify malicious and legitimate websites. A crawler was developed that crawled 7900 URLs from AlexaRank portal and also employed Phishtank dataset to measure the efficiency of the proposed URL detector. The outcome of this study reveals that the proposed method presents superior results rather than the existing deep learning methods. A total of 7900 malicious URLS were detected using the proposed URL detector. It has achieved better accuracy and F1—score with limited amount of time. The future direction of this study is to develop an unsupervised deep learning method to generate insight from a URL. In addition, the study can be extended in order to generate an outcome for a larger network and protect the privacy of an individual.

11. FUTURE SCOPE

We'll look into the links between phishing sites and hosting and DNS registration providers in more detail. We'll also look at other features like Content Security Policies, certificate authorities, and TLS fingerprinting that can be used. In addition, we will compare SVMs and neural networks to other machine learning techniques such as random forest classifiers for speed and accuracy. Finally, we'll check for aspects in the underlying HTML structure, such as tag counts, tag positioning, use of and counts of specific JavaScript functions, inline and included CSS, and so on.

13. APPENDIX

GIT HUB AND PROJECT DEMO LINK:

https://github.com/IBM-EPBL/IBM-Project-3806-1658644448/tree/main/Video%20Recording