

Detection Of Phishing Attack by Hyper Tuning Method

Submitted in partial fulfilment of the requirements for the award of Master of Engineering degree in Computer Science and Engineering

by

C.GOKUL (REG.NO.40310001)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL OF
COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

MAY - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of C. GOKUL (Reg.no.40310001) who carried out the project entitled "**Detection of Phishing Attack by Hyper Tuning Method**" under my supervision from November 2021 to May 2022.

Internal Guide

Dr. S. JAYANTHI (M.E., PhD.,)

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **C. GOKUL** hereby declare that the Project Report entitled “**Detection of Phishing Attack by Hyper Tuning Method**” done by me under the guidance of **Dr. S. Jayanthi (M.E., PhD.,)** at **Sathyabhama Institute of Science and Technology**, is submitted in partial fulfilment of the requirements for the award of **Master of Engineering in Computer Science and Engineering**.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr. T. Sasikala M.E., Ph.D. Dean, School of Computing, Dr.S. Vigneshwari M.E., Ph.D. and Dr.L. Lakshmanan M.E., Ph.D., Heads of the Department of Computer Science and Engineering for providing me the necessary support and guidelines at the right time during the progressive reviews.

I wish to express my sincere and deep sense of gratitude to my Project Guide (Dr. S. Jayanthi M.E., PhD.) for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work. I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways for the completion of the project

ABSTRACT

Tremendous resources are spent by organizations guarding against and recovering from cyber security attacks by online hackers who gain access to sensitive and valuable user data. Many cyber infiltrations are accomplished through phishing attacks where users are tricked into interacting with web pages that appear to be legitimate. Phishing is a common attack on credulous people by making them to disclose their unique information using counterfeit websites. The objective of phishing website URLs is to purloin the personal information like user name, passwords and online banking transactions. Phishers use the websites which are visually and semantically similar to those real websites. As technology continues to grow, phishing techniques started to progress rapidly and this needs to be prevented by using anti-phishing mechanisms to detect phishing.

Phishing is popular among attackers, since it is easier to trick someone into clicking a malicious link which seems legitimate than trying to break through a computers defence system. The malicious links within the body of the message are designed to make it appear that they go to the spoofed organization using that organizations logos and other legitimate contents.

Machine learning is a powerful tool used to strive against phishing attacks. This paper surveys the features used for detection and detection techniques using machine learning.

TABLE OF CONTENT

S.NO	TITLE	PAGE NUMBER
	ABSTRACT	
I.	INTRODUCTION	1
II.	LITERATURE SURVEY	3
III.	AIM	6
	a. OBJECTIVES	6
	b. SCOPE OF THE PRESENT INVESTIGATION	6
	c. EXISTING SYSTEM	6
	d. PROPOSED SYSTEM	7
IV.	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	8
	a. SOFTWARE AND HARDWARE REQUIREMENTS	8
	b. OVERVIEW OF GOOGLE COLABS	8
	c. PYTHON OVERVIEW	9
	d. PYTHON BUILTIN MODULES	10
	e. METHODOLOGY	12
	f. ALGORITHMS USED	15
V.	RESULTS AND DISCUSSION, PERFOMANCE ANALYSIS	20
VI.	SUMMARY AND CONCLUSIONS	24
VII.	APPENDIX OF CODE	25
VIII.	REFERENCE	41

ABBREVIATIONS

SVM	Support vector machine
DTC	Decision tree
RF	Random Forest
NB	Naive Bayes
KNN	K Nearest Neighbour

LIST OF FIGURES

Fig 1 : Dataflow Diagram	16
Fig 2 : General Architecture Diagram	17
Fig 3 : Architecture Diagram with Hyper Tuning	18
Fig 4 : Splitting Data set Diagram	19

CHAPTER 1

I. INTRODUCTION

While cybersecurity attacks continue to escalate in both scale and sophistication, social engineering approaches are still some of the simplest and most effective ways to gain access to sensitive or confidential information. The United States Computer Emergency Readiness Team (US-CERT) defines phishing as a form of social engineering that uses e-mails or malicious websites to solicit personal information from an individual or company by posing as a trustworthy organization or entity. While organizations should educate employees about how to recognize phishing e-mails or links to help protect against the above types of attacks, software such as HTTrack is readily available for users to duplicate entire websites for their own purposes. As a result, even trained users can still be tricked into revealing private or sensitive information by interacting with a malicious website that they believe to be legitimate.

The above problem implies that computer-based solutions for guarding against phishing attacks are needed along with user education. Such a solution would enable a computer to have the ability to identify malicious websites in order to prevent users from interacting with them. One general approach to recognizing illegitimate phishing websites relies on their Uniform Resource Locators (URLs). A URL is a global address of a document in the World Wide Web, and it serves as the primary means to locate a document on the Internet. Even in cases where the content of websites are duplicated, the URLs could still be used to distinguish real sites from imposters.

One solution approach is to use a blacklist of malicious URLs developed by anti-virus groups. The problem with this approach is that the blacklist cannot be exhaustive because new malicious URLs keep cropping up continuously. Thus, approaches are needed that can automatically classify a new, previously unseen URL as either a phishing site or a legitimate

one. Such solutions are typically machine-learning based approaches where a system can categorize new phishing sites through a model developed using training sets of known attacks.

One of the main problems with developing machine learning based approaches for this problem is that very few training data sets containing phishing URLs are available in the public domain. As a result, studies are needed that evaluate the effectiveness of machine-learning approaches based on the data sets that do exist. This work aims to contribute to this need. Specifically, the goal of this research is to compare the performance of the commonly used machine learning algorithms on the same phishing data set. In this work, we use a data set, where features from the data URLs have already been extracted, and the class labels are available. We have tested common machine learning algorithms for the purpose of classifying URLs such as SVM, Naïve Bayes' classifier, decision tree, and neural network.

CHAPTER 2

II. LITERATURE SURVEY

[1] This paper aims at surveying many of the recently proposed phishing mitigation techniques. A high-level overview of various categories of phishing mitigation techniques is also presented, such as: detection, offensive defense, correction, and prevention, which we believe is critical to present where the phishing detection techniques fit in the overall mitigation process.

[2] It has been observed that the decision tree classifier predicts the phishing website more accurately than other learning algorithms, but it is not effective for future detection. This paper has proposed a new approach called multi-tier classification model for phishing email filtering.

[3] It has a method for extracting the features of phishing email related to weighting of message content and message header and selects the features according to priority ranking. An empirical performance and analysis of the proposed algorithm have been presented. Due to rapidly evolving nature of both legitimate and phishing emails, existing corpus rapidly becomes outdated. This paper describes a Comprehensive survey on state of art of security analytics, which is its description, technology, trends and tools.

[4] Security analytics aims to detect previously undiscovered threats by use of analytic techniques. Common techniques of security analytics include clustering and graph-based event correlation are used, but not mentioned about the false positive rate for fraudulent transactions. Relationships between files are represented as a graph to detect malware presence. In this paper the scheme first collects from the clients the file lists which describe their mutual relationships, and determines if there are potentially malicious relationships.

[5] The file associations are then used to generate an undirected weighted file relationship graph, and based on the graph a belief propagation classifier is trained, but this is unable to determine the information apart from the file contents extracted. This paper uses the Lexical Features as the classifying parameters in the Detection of Malicious URLs, by leveraging the Visible Attributes it is possible to classify the Malicious Short URLs.

[6] In this paper authors proposes a classification model in order to classify the phishing attacks. This model comprises of feature extraction from sites and classification of website. In feature extraction, 30 features have been taken from UCIIrvine machine learning repository data set and phishing feature extraction rules has been clearly defined. In order to classification of these features, Support Vector Machine (SVM), Naïve Bayes (NB) and Extreme Learning Machine (ELM) were used. In Extreme Learning Machine (ELM), six activation functions were used and achieved 95.34% accuracy than SVM and NB. The results were obtained with the help of MATLAB.

[7] This paper proposes a efficient way to detect phishing URL websites by using c4.5 decision tree approach. This technique extracts features from the sites and calculates heuristic values. These values were given to the c4.5 decision tree algorithm to determine whether the site is phishing or not. Dataset is collected from Phish Tank and Google. This process includes two phases namely pre-processing phase and detection phase. In which features are extracted based on rules in pre-processing phase and the features and their respected values were inputted to the c4.5 algorithm and obtained 89.40% accuracy.

[8] This paper focuses on detecting phishing website URLs with domain name features. Web spoofing attack categories content-based, heuristic-based and blacklist-based approaches are explained and the proposed model Phish Checker is developed with the help of Microsoft Visual Studio Express 2013 and C# language. Dataset used from Phish tank and Yahoo directory set and obtained an accuracy of 96%. This paper checks only the validity of URLs.

[9] In this paper author introduces a phishing URL detection system depends on URL lexical analysis named PhishScore. This approach is based on intra-URL relatedness. This relatedness reflects the relationship into part of the URL Right around 12 site highlights removed from a solitary URL are utilized to include machine learning algorithms to identify phishing URLs. This experiment results accuracy of 94.91%.

CHAPTER – 3

III. AIM

3.1 OBJECTIVE

- A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages.
- The objective of this project is to train machine learning models and to predict phishing websites. selection methods. In that lower-level value of feature vector are neglected from data frame.
- Once they are selected, apply these values to the classifier and fine tuning their results using the hyper functions. To get the good accuracy of classifier model.

3.2 SCOPE OF THE PRESENT INVESTIGATION

The phishing website has evolved as a major cybersecurity threat in recent times. The phishing websites host spam, malware, ransomware, drive-by exploits, etc. A phishing website many a time look-alike a very popular website and lure an unsuspecting user to fall victim to the trap. The victim of the scams incurs a monetary loss, loss of private information and loss of reputation. Hence, it is imperative to find a solution that could mitigate such security threats in a timely manner.

To Detect the Phishing attack, we are training the machine learning models to predict phishing websites, selection methods are used to prioritise the higher value and the lower-level value of feature vector are neglected from data frame, Once the higher value are selected these values are applied to the classifier and fine tuning is done with their results using the Hyper functions. Finally, by using the Hyper tuning technique we get the good accuracy of classifier model and the Phishing websites are detected.

3.3 EXISTING SYSTEM:

In exiting work, they implemented the Decision tree, Random Forest, SVM classifiers with different data spit levels, where the Hyper Tuning Method was not performed so the accuracy was not obtained and the learning process was unable to control

Disadvantage:

- Cross validation is not performed.
- Unable to control the learning process.

3.4 PROPOSED SYSTEM:

But in our model, going to use correction map and feature selection. In feature selection, whose priority of their feature vector is gathered from Random Forest feature selection? In that lower-level value of feature vector are neglected from data frame. A collection of machine learning model like SVM, NB, RF, DTC, KNN are used to predict, based on BAC – balanced accuracy score particular classifier model is fine-tuned by hyper parameters and then classifier predict whether the data is stress or normal. The complete implementation can be done through Google Colab (Python-Jupyter Notebook).

Feature Selection:

- Heat map Correlation using Seaborn library
- Feature selection based on RF

Classifier:

Compare the performance evaluation of different classifiers:

- SVM – Support vector machine
- DTC – Decision tree
- RF – Random Forest
- NB – Naive Bayes
- KNN – k nearest neighbour

Advantage:

- While using the hyper parameters, the learning process is controlled.
- Easy to manage a large set of experiments
- It makes the algorithm more highlighted with high accuracy.

CHAPTER – 4

4.1 Software Requirements:

Environment:

- Cloud Tool: Google colab
- Operating system: 64 Bit OS (Windows 7/ 8 / 10) / Linux
- IDE : Python – Jupyter notebook

4.1.1 Hardware Requirements:

- RAM: Minimum 4GB
- ROM: 128 GB minimum

4.2 OVERVIEW OF GOOGLE COLABS

Google Colab was developed by Google to provide free access to GPU's and TPU's to anyone who needs them to build a machine learning or deep learning model. Google Colab can be defined as an improved version of Jupyter Notebook.

4.2.1 Google Colab Features:

Google Colab provides tons of exciting features that any modern IDE offers, and much more. Some of the most exciting features are listed below.

- Interactive tutorials to learn machine learning and neural networks.
- Write and execute Python 3 code without having a local setup.
- Execute terminal commands from the Notebook.
- Import datasets from external sources such as Kaggle.
- Save your Notebooks to Google Drive.
- Import Notebooks from Google Drive.
- Free cloud service, GPUs and TPUs.
- Integrate with PyTorch, Tensor Flow, Open CV.
- Import or publish directly from/to GitHub.

4.3 PYTHON OVERVIEW:

Python is a general purpose programming language. It is an interpreted language, which makes it suitable for rapid development and prototyping of programming segments or complete small programs.

Python's main advantages:

Open source software, supported by Python Software Foundation Available on all major platforms (Windows, macOS, Linux)

It is a good language for new programmers to learn due to its straightforward, object-oriented style

It is well-structured, which aids readability

It is extensible (i.e., modifiable) and is supported by a large community who provide a comprehensive range of 3rd party packages

Features of Python:

- Easy to Learn and Use
- Expressive Language
- Interpreted Language
- Cross-platform Language
- Free and Open Source
- Object-Oriented Language
- Extensible
- Large Standard Library
- Embeddable

4.3.1 PYTHON ENVIRONMENT:

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

A notebook document consists of rich text elements with HTML formatted text, figures, mathematical equations etc. The notebook is also an executable document consisting of code blocks in Python or other supporting languages.

4.3.2 OVERVIEW OF JUPYTER NOTEBOOK:

What is Jupyter Notebook?

Jupyter Notebook is an application that allows editing and running Notebook documents through a web browser or an Integrated Development Environment (IDE). Instead of files, you will work with Notebooks.

Jupyter notebook is a client-server application. The application starts the server on local machine and opens the notebook interface in web browser where it can be edited and run from. The notebook is saved as ipynb file and can be exported as html, pdf and LaTeX files.

4.4 PYTHON BUILTIN MODULES

- Pandas
- Numpy
- Seaborn
- Matplotlib
- Scikit-Learn
- Plotly

PANDAS:

Pandas is one of the most popular modules in python that can be used for data manipulation and analysis using python. Basically, it provides an easy interface to interact with flowing data and apply transformations to them on the go. This module is covered under the BSD license and can be used for free. You can download this module by visiting the website or by installing it through the python package manager.

NUMPY:

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

SEABORN:

Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.

Seaborn offers the following functionalities:

- Dataset oriented API to determine the relationship between variables.
- Automatic estimation and plotting of linear regression plots.
- It supports high-level abstractions for multi-plot grids.
- Visualizing univariate and bivariate distribution.

MATPLOTLIB:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

There are different types of graphs in Matplotlib, we'll be looking into:

- Bar Graph
- Scatter Plot
- Line Graph
- Histogram
- Box Plot

SCIKIT-LEARN:

In python, scikit-learn library has a pre-built functionality under sklearn. Pre processing.

Next thing is to do feature extraction Feature extraction is an attribute reduction process. Unlike feature selection, which ranks the existing attributes according to their predictive significance, feature extraction actually transforms the attributes. The transformed attributes, or features, are linear combinations of the original attributes. Finally, our models are trained using Classifier algorithm. We use nltk. classify module on Natural Language Toolkit library on Python. We use the labelled dataset gathered . The rest of our labelled data will be used to evaluate the models. Some machine learning algorithms were used to classify pre processed data. The

chosen classifiers were Decision tree, Support Vector Machines and Random Forest. These algorithms are very popular in text classification tasks.

PLOTLY:

The Plotly Python library is an interactive open-source library. This can be a very helpful tool for data visualization and understanding the data simply and easily. plotly graph objects are a high-level interface to plotly which are easy to use. It can plot various types of graphs and charts like scatter plots, line charts, bar charts, box plots, histograms, pie charts, etc.

So you all must be wondering why plotly over other visualization tools or libraries? Here's the answer –

Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in a large number of data points.

It is visually attractive that can be accepted by a wide range of audiences.

It allows us for the endless customization of our graphs that makes our plot more meaningful and understandable for others.

Ok, enough theory let's start.

Installation:

To install this module, type the below command in the terminal.

pip install plotly

4.5 METHODOLOGY

System consists of three modules:

- Classification (General)
- RF Feature Selection
- Hyper tuning

Classification (General)

Classification is used to compare between the different types of algorithms such as Random Forest (RF), Decision Tree (DT), Naive Bayes (NB), K Nearest Neighbour (KNN), Supervised Machine Learning Algorithm (SVM) and to choose the maximum Accuracy among them

RF Feature Selection

Feature selection using Random Forest comes under the category of Embedded methods. Embedded methods combine the qualities of filter and wrapper methods. They are implemented by algorithms that have their own built-in feature selection methods.

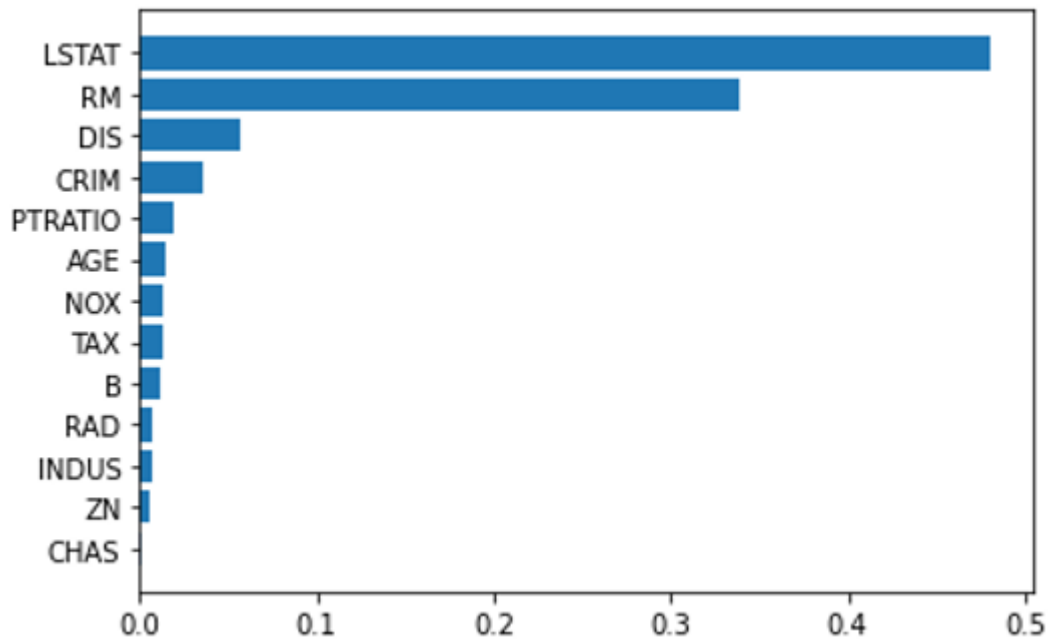
Some of the benefits of embedded methods are:

- They are highly accurate.
- They generalize better.
- They are interpretable

How does Random Forest select features?

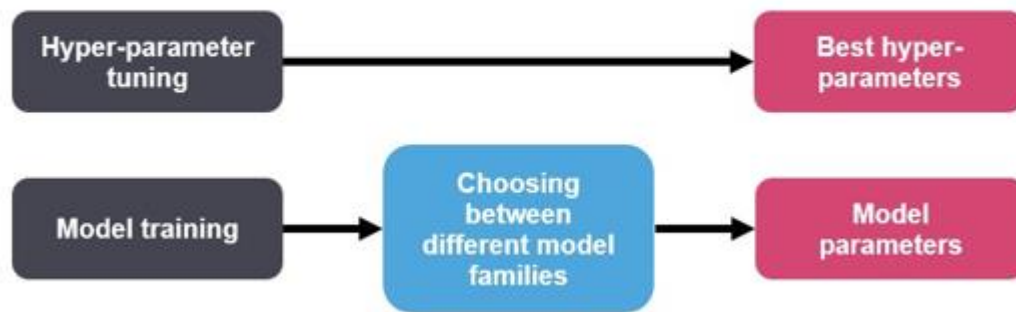
Random forests consist of 4 –12 hundred decision trees, each of them built over a random extraction of the observations from the dataset and a random extraction of the features. Not every tree sees all the features or all the observations, and this guarantees that the trees are de-correlated and therefore less prone to over-fitting. Each tree is also a sequence of yes-no questions based on a single or combination of features. At each node (this is at each question), the tree divides the dataset into 2 buckets, each of them hosting observations that are more similar among themselves and different from the ones in the other bucket. Therefore, the importance of each feature is derived from how “pure” each of the buckets is.

Sample Outcome for RF Feature Selection:



Hyper Tuning

- Hyper parameter tuning is choosing a set of optimal hyper parameters for a learning algorithm. A hyper parameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyper parameter tuning.
- Hyper-parameters: Model parameters are learned from data and hyper-parameters are tuned to get the best fit. Searching for the best hyper-parameter can be tedious, hence search algorithms like grid search and random search are used.



4.6 ALGORITHMS USED

Random Forest

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Naive Bayes

Naive Bayes is a supervised machine learning classification algorithm based on the Bayes theorem. It supports continuous data. It is a simple classification technique, but has high functionality.

K Nearest Neighbour

KNN algorithm is one of the simplest classification algorithms and it is one of the most used learning algorithms. KNN is a non-parametric, lazy learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

Supervised Machine Learning Algorithm

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs.

Dataflow Diagram:

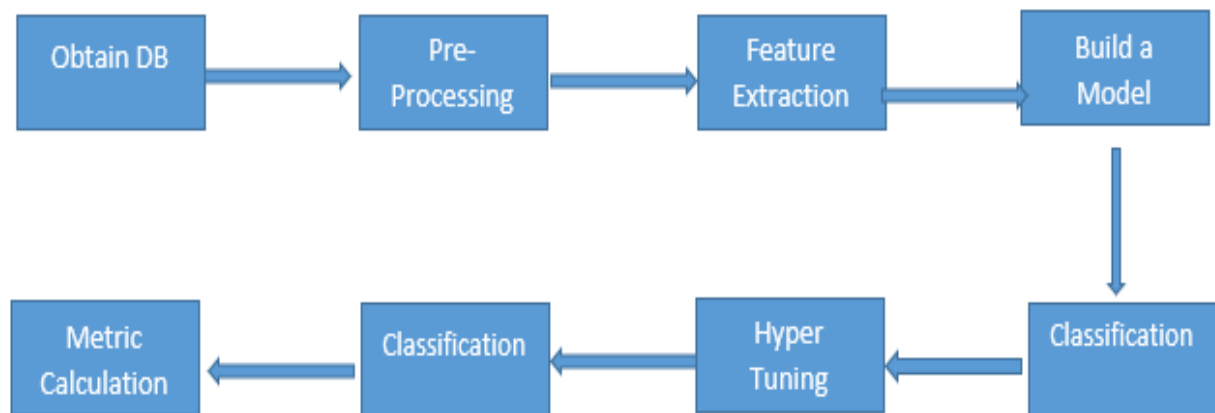


Fig 1: Dataflow Diagram

General Architecture Diagram:

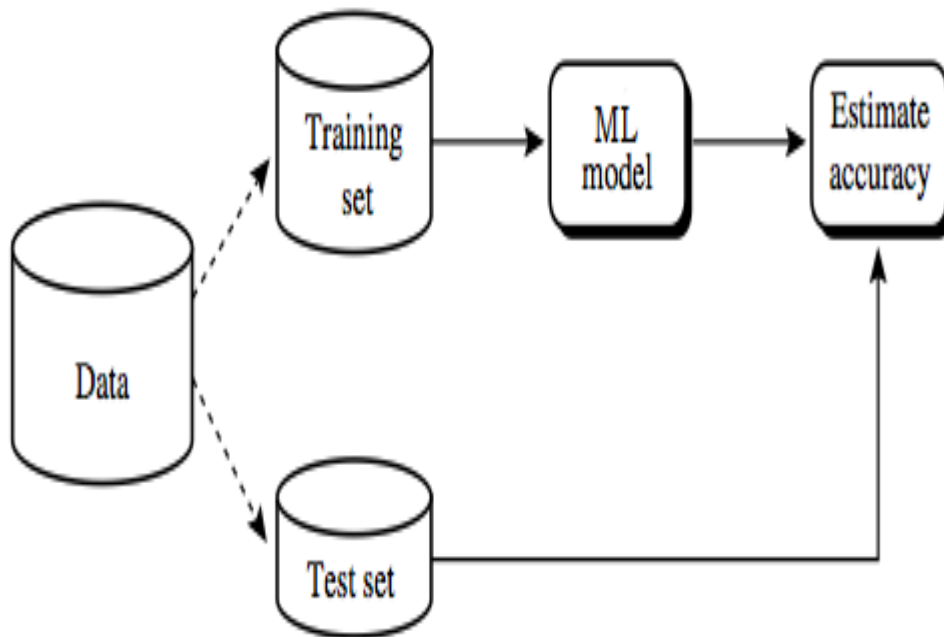


Fig 2: General Architecture Diagram

Architecture Diagram with Hyper Tuning:

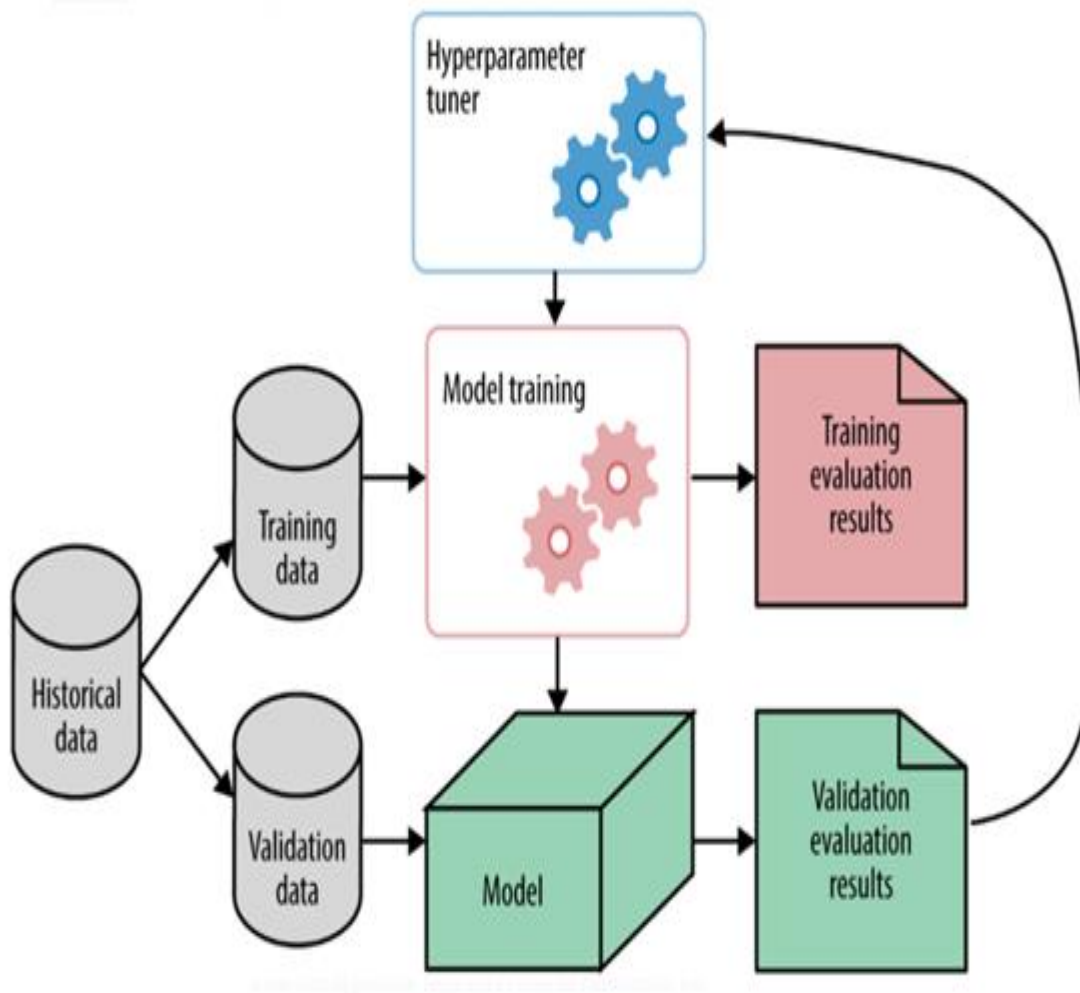


Fig 3: Architecture Diagram with Hyper Tuning

Splitting The Dataset

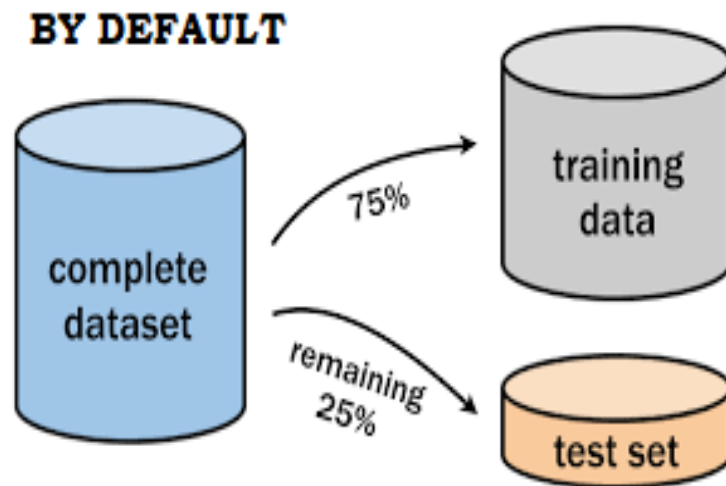


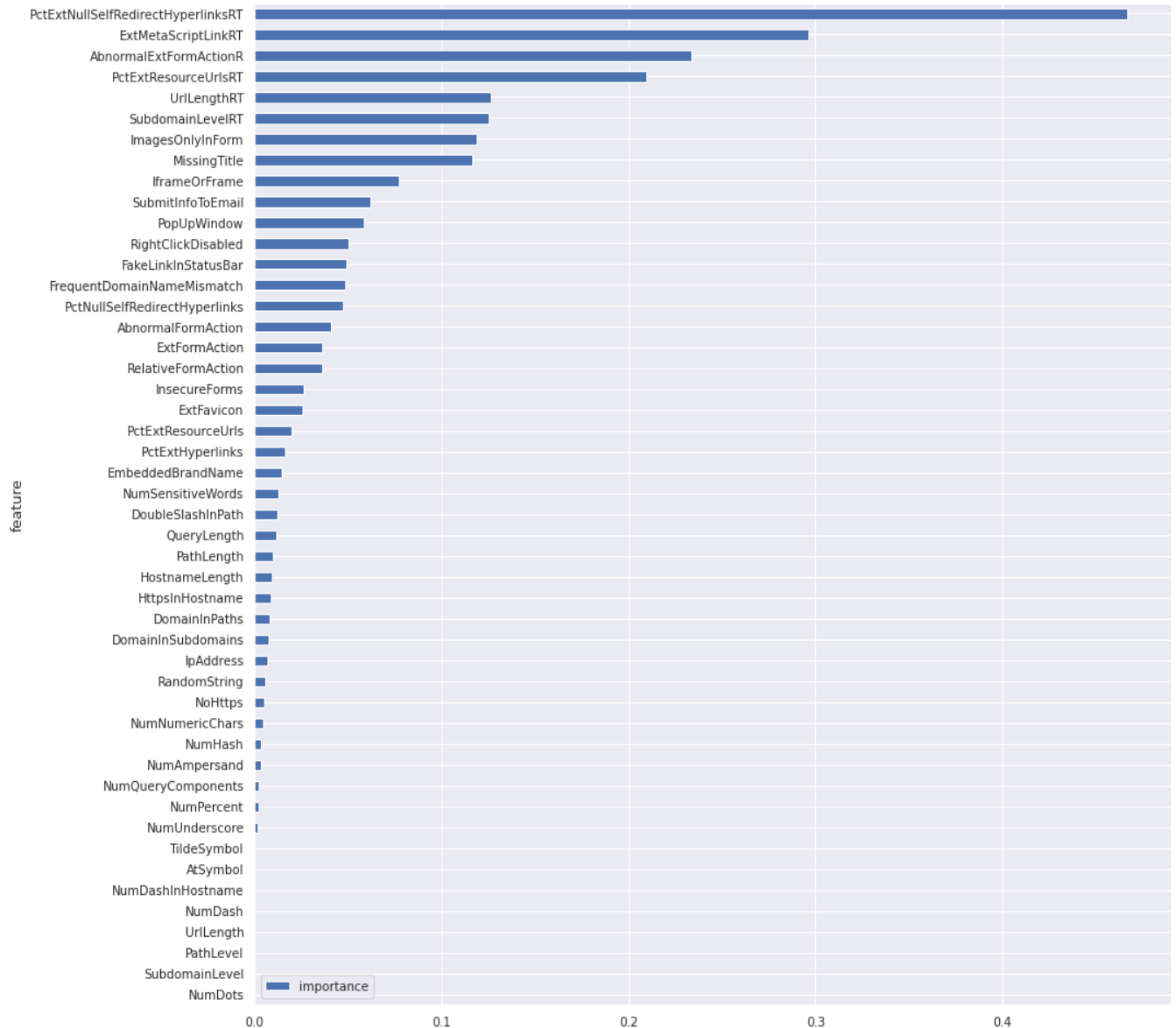
Fig 4: Splitting Dataset Diagram

CHAPTER 5

RESULT AND DISCUSSION

The dataset used for the training is obtained from Internet sources which is further a manual collection of URLs . We created a dataset containing an equal number of labelled phishing and non-phishing websites . We further performed random mixing of the dataset to remove any order in the dataset. The dataset contained 100000 examples and is further split in the ratio 7:3 for training and testing. We trained the classifiers based on the features extracted from the lexical structure of the URL.

Calculating the feature importance



Output after calling the multiple classifier

	model	feature_count	acc	prc	rec	f1
0	rfc_model_1	48	0.980000	0.980000	0.980000	0.980000
1	rfc_model_2	48	0.981667	0.981667	0.981667	0.981667
2	dtc_model_1	48	0.965667	0.965667	0.965667	0.965667
3	nbc_model_1	48	0.839667	0.839667	0.839667	0.839667
4	svc_model_1	48	0.850667	0.850667	0.850667	0.850667
5	knc_model_1	48	0.857333	0.857333	0.857333	0.857333

After Removing Less Priority Features

	model	feature_count	acc	prc	rec	f1
0	rfc_model_1	48	0.980000	0.980000	0.980000	0.980000
1	rfc_model_2	48	0.981667	0.981667	0.981667	0.981667
2	dtc_model_1	48	0.965667	0.965667	0.965667	0.965667
3	nbc_model_1	48	0.839667	0.839667	0.839667	0.839667
4	svc_model_1	48	0.850667	0.850667	0.850667	0.850667
5	knc_model_1	48	0.857333	0.857333	0.857333	0.857333
6	rfc_model_1a	22	0.976000	0.976000	0.976000	0.976000
7	rfc_model_2a	22	0.977333	0.977333	0.977333	0.977333
8	dtc_model_1a	22	0.960667	0.960667	0.960667	0.960667
9	nbc_model_1a	22	0.839333	0.839333	0.839333	0.839333
10	svc_model_1a	22	0.841333	0.841333	0.841333	0.841333

Output after Hyper Tuning

	model	feature_count	auc	f1	rec
0	rfc_model_X_A	22	0.995666	0.973509	0.9698
1	sdg_model_1	22	0.958120	0.868192	0.8950
2	lrc_model_X	22	0.924233	0.849691	0.8900
3	dtc_model_X	22	0.942800	0.942476	0.9378

CHAPTER 6

SUMMARY AND CONCLUSIONS

In this paper, we have explored how well to classify phishing URLs from the given set of URLs containing benign and phishing URLs. We have also discussed the randomization of the dataset, feature engineering, feature extraction using lexical analysis host-based features and statistical analysis. We have also used different classifiers for the comparative study and found that the findings are almost consistent across the different classifiers. We also observed dataset randomization yielded a great optimization and the accuracy of the classifier improved significantly. We have adopted a simple approach to extract the features from the URLs using simple regular expressions. There could be more features that can be experimented and that might lead to improving further the accuracy of the system. The dataset used in this paper contains the URLs list which may be a little old, hence regular continuous training along with a new dataset would enhance the model accuracy and performance significantly.

In our work, we used correlation heat map random forest feature selection process to get the important features, and hyper tuning it's applied to the classifier. Here it's a comparative analysis of website phishing classification, classifiers like RF, DTC, KNN, SVM, NB etc. To identify the performance of different classifier.

APPENDIX

A. Code of the System

```
import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import plotly.express as px

import plotly.graph_objects as go

import warnings

warnings.filterwarnings('ignore')

import string
```

Read the File through Pandas Library

```
y1 = pd.read_csv("Phishing.csv" )

display(y1.shape)

display(y1.head())

display("")

display(y1.tail())
```

Information about Dataset

```
def display_file_info(df,df_name):

    col_name_list    = list(df.columns)

    col_data_type    = [type(col) for col in df.iloc[0,:]]
```

```

null_count_list = [df[col].isnull().sum() for col in col_name_list]

unique_count_list = [df[col].nunique() for col in col_name_list]

memory_usage_list = [df[col].memory_usage() for col in col_name_list]

total_memory_usage = sum(memory_usage_list)

df_info = pd.DataFrame({'column_name': col_name_list,
                        'column_type': col_data_type,
                        'null_count': null_count_list,
                        'unique_count': unique_count_list})

print('----- {} Summary --- {} x rows | {} x cols | approx {:.2f} MB:'
      .format(df_name , df.shape[0], df.shape[1], total_memory_usage))

print( df_info.to_string())

```

#####

```
display_file_info(y1,'Input Data')
```

```
y1.info()
```

Changing the column data type

```
float_dataType = y1.select_dtypes('float64').columns
```

```
for i in float_dataType:
```

```
    y1[i] = y1[i].astype('float32')
```

```
int_dataType = y1.select_dtypes('int64').columns
```

```
for j in int_dataType:
```

```
    y1[j] = y1[j].astype('int32')
```

```
y1.info()
```

Target Class

```
from plotly.offline import iplot
```

```
pred_class = ['Yes', 'No' ]
```

```
trace = go.Pie(labels = pred_class, values = predict_class_count)
```

```
data = [trace]
```

```
fig = go.Figure(data = trace)
```

```
iplot(fig)
```

our dataset contains 50 columns

splitting the data as 15, 15,20 (fto view the statistical information)

```
yx1 = y1.iloc[:,17]
```

```
yx2 = y1.iloc[:,18:35]
```

```
yx3 = y1.iloc[:,36:]
```

```
display(yx1.shape)
```

```
display(yx2.shape)
```

```
display(yx3.shape)
```

```
%matplotlib inline
```

```
sns.set(rc={'figure.figsize':(15,15)})
```

```
sns.heatmap(yx1.corr(),annot=True,vmin=-1,vmax=1,cmap='coolwarm')
```

```
%matplotlib inline
```

```
sns.set(rc={'figure.figsize':(15,15)})
```

```
sns.heatmap(yx2.corr(),annot=True,vmin=-1,vmax=1,cmap='coolwarm')
```

```
%matplotlib inline
```

```
sns.set(rc={'figure.figsize':(15,15)})
```

```
sns.heatmap(yx3.corr(),annot=True,vmin=-1,vmax=1,cmap='coolwarm')
```

To obtain the mutual info b/t all the columns

```
target_class = y1['CLASS_LABEL']
```

```
y1_data = y1.drop(['id', 'CLASS_LABEL'], axis=1)
```

getting discrete features

```
from sklearn.feature_selection import mutual_info_classif
```

```
mutualInfo_scores = mutual_info_classif(y1_data, target_class, discrete_features=feat_1)
```

```
mutualInfo_scores = pd.Series(mutualInfo_scores, name='MI Scores', index=y1_data.columns)
```

```
mutualInfo_scores = mutualInfo_scores.sort_values(ascending=False)
```

```
mutualInfo_scores
```

calculating the feature importance

```
mutualInfo_scores_1 = mutualInfo_scores.sort_values(ascending=True)
```

```
feature_importances = pd.DataFrame({'feature': list(y1_data.columns),
```

```
                                     'importance': list(mutualInfo_scores_1)})
```

```
feature_importances.sort_values(by='importance', inplace=True)
```

```
feature_importances.set_index('feature', drop=True, inplace=True)
```

```
feature_importances.plot(kind='barh')
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import GridSearchCV as GSCV
```

```
from sklearn.model_selection import cross_validate as CV
```

```
from sklearn.ensemble import RandomForestClassifier as RFC
```

```
from sklearn.tree import DecisionTreeClassifier as DTC
```

```

from sklearn.naive_bayes import GaussianNB as NBC

from sklearn.linear_model import LogisticRegression as LR

from sklearn.linear_model import SGDClassifier as SDC

from sklearn.neighbors import KNeighborsClassifier as KNN

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, balanced_accuracy_score, precision_score, recall_score, f1_score

X_train, X_test, y_train, y_test = train_test_split(y1_data , target_class, random_state=0 ,
test_size=0.30,stratify=target_class)

display("for training")

print(X_train.shape, y_train.shape)

display("for testing")

print(X_test.shape, y_test.shape)

for training

(7000, 48) (7000,)

for testing

(3000, 48) (3000,)

creating an empty list to save the results (classifier)

model_names = []

feature_counts = []

model_acc_scores = []

model_prc_scores = []

model_rec_scores = []

model_f1_scores = []

calling multiple classifier

```

```

# RFC -- Model 1

rfc_model_1 = RFC(random_state=0)

rfc_model_1.fit(X_train, y_train)

rfc_model_1_y_hat = rfc_model_1.predict(X_test)

model_names.append('rfc_model_1')

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, rfc_model_1_y_hat))

model_prc_scores.append(precision_score(y_test, rfc_model_1_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, rfc_model_1_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, rfc_model_1_y_hat, average='micro'))


# RFC -- Model 2

rfc_model_2 = RFC(random_state=42, class_weight='balanced')

rfc_model_2.fit(X_train, y_train)

rfc_model_2_y_hat = rfc_model_2.predict(X_test)

model_names.append('rfc_model_2')

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, rfc_model_2_y_hat))

model_prc_scores.append(precision_score(y_test, rfc_model_2_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, rfc_model_2_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, rfc_model_2_y_hat, average='micro'))


# DTC Model

dtc_model_1 = DTC(random_state=0)

```

```

dtc_model_1.fit(X_train, y_train)

dtc_model_1_y_hat = dtc_model_1.predict(X_test)

model_names.append('dtc_model_1')

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, dtc_model_1_y_hat))

model_prc_scores.append(precision_score(y_test, dtc_model_1_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, dtc_model_1_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, dtc_model_1_y_hat, average='micro'))


# Navie Bayes

nbc_model_1 = NBC()

nbc_model_1.fit(X_train, y_train)

nbc_model_1_y_hat = nbc_model_1.predict(X_test)

model_names.append('nbc_model_1')

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, nbc_model_1_y_hat))

model_prc_scores.append(precision_score(y_test, nbc_model_1_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, nbc_model_1_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, nbc_model_1_y_hat, average='micro'))


# Support vector

svc_model_1 = SVC(random_state=0)

svc_model_1.fit(X_train, y_train)

svc_model_1_y_hat = svc_model_1.predict(X_test)

model_names.append('svc_model_1')

```

```

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, svc_model_1_y_hat))

model_prc_scores.append(precision_score(y_test, svc_model_1_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, svc_model_1_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, svc_model_1_y_hat, average='micro'))

# kNN

knc_model_1 = KNN()

knc_model_1.fit(X_train, y_train)

knc_model_1_y_hat = knc_model_1.predict(X_test)

model_names.append('knc_model_1')

feature_counts.append(X_train.shape[1])

model_acc_scores.append(accuracy_score(y_test, knc_model_1_y_hat))

model_prc_scores.append(precision_score(y_test, knc_model_1_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test, knc_model_1_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test, knc_model_1_y_hat, average='micro'))

f_model_eval = pd.DataFrame({'model': model_names, 'feature_count': feature_counts,

                             'acc': model_acc_scores,

                             'prc': model_prc_scores, 'rec': model_rec_scores,

                             'f1': model_f1_scores})

df_model_eval

```

removing less priority features

```

from sklearn.feature_selection import mutual_info_classif

mutualInfo_scores = mutual_info_classif(y1_data, target_class, discrete_features=feat_1)

mutualInfo_scores = pd.Series(mutualInfo_scores, name='MI Scores', index=y1_data.columns)

```

```

mutualInfo_scores = mutualInfo_scores.sort_values(ascending=False)

mutualInfo_scores

df_train_features = y1_data_new.drop(columns=
['DoubleSlashInPath','SubdomainLevel','FakeLinkInStatusBar','RandomString',
'ImagesOnlyInForm','SubdomainLevelRT', 'HttpsInHostname', 'NumHash', 'NoHttps',
'RelativeFormAction',
'RightClickDisabled','ExtFavicon', 'AtSymbol','PopUpWindow','SubdomainLevel','TildeSymbol',
'NumDashInHostname','DomainInSubdomains','DomainInPaths','NumAmpersand',
'NumPercent', 'MissingTitle','AbnormalFormAction','NumUnderscore','ExtFormAction',
'EmbeddedBrandName','IpAddress'])

df_train_features.head(3)

display(df_train_features.shape)

train test split -- new (after removing the less priority features

X_train1, X_test1, y_train1, y_test1 = train_test_split(df_train_features, target_class, random_state=0 ,
test_size=0.30,stratify=target_class)

display("for training")

print(X_train1.shape, y_train1.shape)

display("for testing")

print(X_test1.shape, y_test1.shape)

# RFC -- Model 1

rfc_model_1a = RFC(random_state=0)

rfc_model_1a.fit(X_train1, y_train1)

rfc_model_1a_y_hat = rfc_model_1a.predict(X_test1)

model_names.append('rfc_model_1a')

feature_counts.append(X_train1.shape[1])

```

```

model_acc_scores.append(accuracy_score(y_test1, rfc_model_1a_y_hat))

model_prc_scores.append(precision_score(y_test1, rfc_model_1a_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test1, rfc_model_1a_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test1, rfc_model_1a_y_hat, average='micro'))


# RFC -- Model 2

rfc_model_2a = RFC(random_state=42, class_weight='balanced')

rfc_model_2a.fit(X_train1, y_train1)

rfc_model_2a_y_hat = rfc_model_2a.predict(X_test1)

model_names.append('rfc_model_2a')

feature_counts.append(X_train1.shape[1])

model_acc_scores.append(accuracy_score(y_test1, rfc_model_2a_y_hat))

model_prc_scores.append(precision_score(y_test1, rfc_model_2a_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test1, rfc_model_2a_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test1, rfc_model_2a_y_hat, average='micro'))


# DTC Model

dtc_model_1a = DTC(random_state=0)

dtc_model_1a.fit(X_train1, y_train1)

dtc_model_1a_y_hat = dtc_model_1a.predict(X_test1)

model_names.append('dtc_model_1a')

feature_counts.append(X_train1.shape[1])

model_acc_scores.append(accuracy_score(y_test1, dtc_model_1a_y_hat))

model_prc_scores.append(precision_score(y_test1, dtc_model_1a_y_hat, average='micro'))

```

```

model_rec_scores.append(recall_score(y_test1, dtc_model_1a_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test1, dtc_model_1a_y_hat, average='micro'))


# Navie Bayes

nbc_model_1a = NBC()

nbc_model_1a.fit(X_train1, y_train1)

nbc_model_1a_y_hat = nbc_model_1a.predict(X_test1)

model_names.append('nbc_model_1a')

feature_counts.append(X_train1.shape[1])

model_acc_scores.append(accuracy_score(y_test1, nbc_model_1a_y_hat))

model_prc_scores.append(precision_score(y_test1, nbc_model_1a_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test1, nbc_model_1a_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test1, nbc_model_1a_y_hat, average='micro'))


# Support vector

svc_model_1a = SVC(random_state=0)

svc_model_1a.fit(X_train1, y_train1)

svc_model_1a_y_hat = svc_model_1a.predict(X_test1)

model_names.append('svc_model_1a')

feature_counts.append(X_train1.shape[1])

model_acc_scores.append(accuracy_score(y_test1, svc_model_1a_y_hat))

model_prc_scores.append(precision_score(y_test1, svc_model_1a_y_hat, average='micro'))

model_rec_scores.append(recall_score(y_test1, svc_model_1a_y_hat, average='micro'))

model_f1_scores.append(f1_score(y_test1, svc_model_1a_y_hat, average='micro'))

df_model_eval = pd.DataFrame({'model': model_names, 'feature_count': feature_counts,

```

```

        'acc': model_acc_scores,

        'prc': model_prc_scores, 'rec': model_rec_scores,

        'f1': model_f1_scores})

```

```
df_model_eval
```

Hyper Tunning based on GridSearch CV

```
XXX = df_train_features
```

```
yyy = target_class
```

```
model_names_A = []
```

```
feature_counts_A = []
```

```
model_rec_scores_A = []
```

```
model_auc_scores_A = []
```

```
model_f1_scores_A = []
```

```
hp_grid = {'n_estimators': [10, 50, 100, 200, 500],
```

```
        'max_features': ['sqrt','log2'],
```

```
        'class_weight': [{0: 0.005, 1: 0.995}, 'balanced']}]
```

```
gscv_rfc = GSCV(estimator=RFC(random_state=0), param_grid=hp_grid, cv=5, n_jobs=5, verbose=20,
scoring='recall')
```

```
gscv_rfc.fit(XXX, yyy)
```

```
gscv_rfc.best_params_
```

```
from sklearn.metrics import roc_auc_score
```

```
cv_scoring = ['balanced_accuracy', 'f1', 'roc_auc', 'recall']
```

```
rfc_model_A = RFC(random_state=0,max_features='sqrt',n_estimators=100,class_weight='balanced')
```

```
cv_results_rfc_model_A = CV(rfc_model_A,XXX,yyy, cv=5, n_jobs=5 , verbose=10 , scoring=cv_scoring)

display(cv_results_rfc_model_A)
```

```
rfc_model_A_auc = cv_results_rfc_model_A['test_roc_auc'].mean()
```

```
rfc_model_A_f1 = cv_results_rfc_model_A['test_f1'].mean()
```

```
rfc_model_A_rec = cv_results_rfc_model_A['test_recall'].mean()
```

```
print('\t\tModel Training Evaluation (Cross-Validation) Results for rfc_model_X_1: ')
```

```
print(' \n\t\t\ttauc: {} \n\t\t\ttf1: {} \n\t\t\trec: {}'
```

```
      .format( rfc_model_A_auc,rfc_model_A_f1,rfc_model_A_rec))
```

```
model_names_A.append('rfc_model_X_A')
```

```
feature_counts_A.append(XXX.shape[1])
```

```
model_auc_scores_A.append(rfc_model_A_auc)
```

```
model_f1_scores_A.append(rfc_model_A_f1)
```

```
model_rec_scores_A.append(rfc_model_A_rec)
```

```
sdc_model_1 = SDC(random_state=0,class_weight='balanced',max_iter=1000,loss='log')
```

```
cv_results_sdc_model_1 = CV(sdc_model_1,XXX,yyy, cv=5, n_jobs=5 , verbose=10 , scoring=cv_scoring)
```

```
display(cv_results_sdc_model_1)
```

```
sdc_model_1_auc = cv_results_sdc_model_1['test_roc_auc'].mean()
```

```
sdc_model_1_f1 = cv_results_sdc_model_1['test_f1'].mean()
```

```

sdc_model_1_rec = cv_results_sdc_model_1['test_recall'].mean()

print('\t\tModel Training Evaluation (Cross-Validation) Results for sdc_model_1: ')

print('\n\t\t\ttauc: {} \n\t\t\ttf1: {} \n\t\t\ttrec: {}'.format( sdc_model_1_auc,sdc_model_1_f1,sdc_model_1_rec))

model_names_A.append('sdc_model_1')

feature_counts_A.append(XXX.shape[1])

model_auc_scores_A.append(sdc_model_1_auc)

model_f1_scores_A.append(sdc_model_1_f1)

model_rec_scores_A.append(sdc_model_1_rec)

from sklearn.linear_model import LogisticRegression as LRC

lrc_model_X = LRC(random_state=0, max_iter=1000, multi_class='ovr', class_weight='balanced', solver='saga',
C=0.001)

cv_results_lrc_model_X = CV(lrc_model_X, XXX,yyy, cv=5, n_jobs=5 , verbose=10 , scoring=cv_scoring)

lrc_model_X_auc = cv_results_lrc_model_X['test_roc_auc'].mean()

lrc_model_X_f1 = cv_results_lrc_model_X['test_f1'].mean()

lrc_model_X_rec = cv_results_lrc_model_X['test_recall'].mean()

print('\t\tModel Training Evaluation (Cross-Validation) Results for lrc_model_X: ')

print('\n\t\t\ttauc: {} \n\t\t\ttf1: {} \n\t\t\ttrec: {}'.format( lrc_model_X_auc,lrc_model_X_f1, lrc_model_X_rec))

model_names_A.append('lrc_model_X')

```

```

feature_counts_A.append(XXX.shape[1])

model_auc_scores_A.append(lrc_model_X_auc)

model_f1_scores_A.append(lrc_model_X_f1)

model_rec_scores_A.append(lrc_model_X_rec)

dtc_model_X = DTC(random_state=0, class_weight='balanced', max_features='sqrt')

cv_results_dtc_model_X = CV(dtc_model_X, XXX,yyy, cv=5, n_jobs=5 , verbose=10 , scoring=cv_scoring)


dtc_model_X_auc = cv_results_dtc_model_X['test_roc_auc'].mean()

dtc_model_X_f1 = cv_results_dtc_model_X['test_f1'].mean()

dtc_model_X_rec = cv_results_dtc_model_X['test_recall'].mean()


print('\t\tModel Training Evaluation (Cross-Validation) Results for dtc_model_X: ')

print('\n\t\t\ttauc: {} \n\t\t\ttf1: {} \n\t\t\ttrec: {}'.format( dtc_model_X_auc,dtc_model_X_f1, dtc_model_X_rec))


model_names_A.append('dtc_model_X')

feature_counts_A.append(XXX.shape[1])

model_auc_scores_A.append(dtc_model_X_auc)

model_f1_scores_A.append(dtc_model_X_f1)

model_rec_scores_A.append(dtc_model_X_rec)

df_model_eval_1 = pd.DataFrame({'model': model_names_A, 'feature_count': feature_counts_A,
                                'auc': model_auc_scores_A,

```

```
'f1':model_f1_scores_A,'rec':model_rec_scores_A})
```

```
df_model_eval_1
```

```
display("for training")
```

```
print(X_train1.shape, y_train1.shape)
```

```
display("for testing")
```

```
print(X_test1.shape, y_test1.shape)
```

References

1. Mahmoud Khonji, Youssef Iraqi, "Phishing Detection: A Literature Survey IEEE, and Andrew Jones, 2013
2. Y. Sönmez, T. Tuncer, H. Gökal, and E. Avci, "Phishing web sites features classification based on extreme learning machine," 6th Int. Symp. Digit. Forensic Secur. ISDFS 2018 - Proceeding, vol. 2018–Janua, pp. 1–5, 2018.
3. T. Peng, I. Harris, and Y. Sawa, "Detecting Phishing Attacks Using Natural Language Processing and Machine Learning," Proc. - 12th IEEE Int. Conf. Semant. Comput. ICSC 2018, vol. 2018–Janua, pp. 300–301, 2018
4. S. Parekh, D. Parikh, S. Kotak, and P. S. Sankhe, "A New Method for Detection of Phishing Websites: URL Detection," in 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, vol. 0, no. Iccict, pp. 949–952.
5. L. MacHado and J. Gadge, "Phishing Sites Detection Based on C4.5 Decision Tree Algorithm," in 2017 International Conference on Computing, Communication, Control and Automation, ICCUBEA 2017, 2018, pp. 1–5
6. M. Aydin and N. Baykal, "Feature extraction and classification phishing websites based on URL," 2015 IEEE Conf. Commun. NetworkSecurity, CNS 2015, pp. 769–770, 2015
7. N. Abdelhamid, A. Ayesh, F. Thabtah, "Phishing Detection based Associative Classification", Data Mining. Expert Systems with Applications (ESWA), vol. 41, pp 5948-5959, 2014