



**GITAM**  
(DEEMED TO BE UNIVERSITY)  
(Estd. u/s 3 of the UGC Act, 1956)

VISAKHAPATNAM 🌞 HYDERABAD 🌞 BENGALURU

# **MALICIOUS URL DETECTION USING MACHINE LEARNING**

By

**Shivani Donthi** (222022502019)

**Srinath Tummala** (222022502027)

Under the guidance of

**Dr. Motahar Reza**

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Data Science

Department of Mathematics, School of Science,  
GITAM (Deemed to be) University, Hyderabad.

June 2022.

© GITAM University, 2022.



# GITAM

(DEEMED TO BE UNIVERSITY)

(Estd. u/s 3 of the UGC Act, 1956)

VISAKHAPATNAM ☀ HYDERABAD ☀ BENGALURU

## CERTIFICATE

*This is to certify that the report titled **Malicious URL Detection using Machine Learning** is a bonafide record of work done by **Shivani Donthi (222022502019)** and **Srinath Tummala (222022502027)** of GITAM (Deemed to be University), Hyderabad, in fulfillment of the requirements of IV semester MSc(Data Science) during the year 2022.*

**Head of the Department**

**Guide**

Valued-by:

1.

2.

## **ACKNOWLEDGEMENTS**

We are very much obliged to honorable Pro-Vice-Chancellor, Prof. D. Sambasiva Rao for providing essential infrastructure and assets during our course of study.

We are specially indebted to Dr. Dattatri Jois Nagesha, Principal, School of Science for his kind assistance and continuous monitoring.

We heartfully thank our Academic Coordinator, Dr. K Anil Kumar for providing us constant support and encouragement throughout our Academics.

We would like to express our sincere gratitude and appreciation to the Head of the Department of Mathematics, and our guide, Dr. Motahar Reza, for giving us this opportunity to take up this project, for supporting and helping us to carry out the project. His constant mentoring and encouragement helped us keep up with the project schedule. We sincerely thank him for taking his time to guide us through the project and make it successful.

We would like to thank all faculties of Mathematics and Computer Science departments for being a pillar of support for us. We take extreme pleasure to thank our parents and friends for their affection and encouragement.

**Shivani Donthi**

**Srinath Tummala**

**Signature**

**Signature**

## TABLE OF CONTENTS

<b>Acknowledgements.....</b>	<b>03</b>
<b>Abstract.....</b>	<b>08</b>
<b>List of Figures.....</b>	<b>09</b>
<b>1. Introduction.....</b>	<b>10</b>
<b>1.1 Project Description</b>	
1.1.1. Domain of Research	
1.1.2. Reason for choosing this Domain	
<b>1.2 Problem Statement and Objective</b>	
<b>1.3 Purpose, Scope, and Applicability</b>	
1.3.1 Purpose	
1.3.2 Scope	
1.3.3 Applicability	
<b>1.4 System Requirements</b>	
1.4.1. Functional Requirements	
1.4.2. Non Functional Requirements	
1.4.3. Hardware and Software Requirements	
1.4.4. Constraints	
<b>2. Literature Review.....</b>	<b>15</b>
<b>2.1 Reviews</b>	
<b>2.2 Gaps Identified</b>	
<b>3. Methodology.....</b>	<b>19</b>

### **3.1 Data Design**

#### **3.1.1. Dataset Collection**

#### **3.1.2. Description of Dataset**

##### 3.1.2.1 Name of Dataset

##### 3.1.2.2 Number of Classes

#### **3.1.3. Data Preparation**

##### 3.1.3.1. Handling NAN Values

##### 3.1.3.2. Removing Noise

##### 3.1.3.3. Formatting the Data

##### 3.1.3.4. Handling Imbalanced Data

#### **3.1.4. Feature Extraction**

#### **3.1.5. Algorithms**

##### 3.1.5.1. Adaboost Classifier

##### 3.1.5.2. Random Forest Classifier

##### 3.1.5.3. Voting Classifier

### **3.2 Model Building**

#### **3.2.1. Review of Machine Learning Algorithms**

#### **3.2.2. Machine Learning Model**

##### 3.2.2.1. Model Category and Rationale for choosing the Model Category

##### 3.2.2.2. Model name and Rationale for choosing the Model Name

##### 3.2.2.3. Model Architecture

##### 3.2.2.4. Steps in Building the Model

#### **3.2.3. Model Training**

### **3.2.4. Model Validation**

#### 3.2.4.1. Validation Strategies

### **3.2.5. Model Scoring**

#### 3.2.5.1. Accuracy

## **3.3 Model Implementation**

### **3.3.1. Data Investigation**

### **3.3.2. Data Preparation**

#### 3.3.2.1. Data Visualization

#### 3.3.2.2. Data Splitting

### **3.3.3. Model Training and Validation**

#### 3.3.3.1. Data Balancing

#### 3.3.3.2. Cross-Validation

#### 3.3.3.3. Model Validation Strategies

## **4. Discussion..... 37**

### **4.1. Performance Metrics**

#### 4.1.1. Classification Reports

#### 4.1.2. Confusion Matrix

## **5. Conclusion..... 40**

### **5.1. Research Project Overview**

### **5.2. Evaluation of Results**

### **5.3. Implementation and Issues**

### **5.4. Advantages and Limitations**

#### 5.4.1. Advantages

5.4.2. Limitations

**5.5. Future Work and Recommendations**

**6. Appendix..... 44**

**6.1. References**

## **ABSTRACT**

URLs allow Internet users to move from one website to another. Fully represent access to content stored on servers somewhere in the world. URLs are available by simply clicking on a link or image or typing in our browsers. A favorite method used by attackers and children of text is to deceive the social media because regular users still click on any link or visit any URL they find. Blocking other URLs is a fundamental and essential way to provide a basic level of security. With the advent of internet technology, network security is under various threats. In particular, cybercriminals can spread the same dangerous industries (URLs) to attack as criminals by stealing sensitive and spam information. Searching for the wrong URL is vital in preventing this attack. Cybercriminals use malicious URLs as distribution channels to distribute malicious software on the web. Attackers use browser vulnerabilities to install malicious software so that they can access the victim's computer remotely. A malware program aims to gain access to the network, filter sensitive information, and secretly monitor targeted computer systems. In this project, we compare models and find the best guessing model for classifying spam and ham URLs in a better way. Our proposed prediction model seeks to improve the accuracy of the forecast by using various factors that take into account the interaction effect of different parameters.



## LIST OF FIGURES

Figure No	Figure Name	Page No
3.1.1	Overview of the Dataset	19
3.1.4. (a) - 3.1.4. (i)	Examples of Features Extracted	21 - 24
3.2.2.4	Steps Followed By Machine Learning Model	27
3.2.4.1	Cross-Validation Snippet	29
3.3.2.1	Heat Map for extracted features	31
3.3.3(a)	Predictor and Target Variables	32
3.3.3(b)	Splitting of Train and Test Data	32
3.3.3.1(a)	Imbalance in Dataset	33
3.3.3.1(b)	Size of Dataset before Oversampling Technique	33
3.3.3.1(c)	Oversampling Technique	34
3.3.3.1(d)	Size of Dataset after Oversampling Technique	34
3.3.3.2	Cross-Validation Snippet	35
4.1.1(a)	Training data Accuracy	37
4.1.1(b)	Testing data Accuracy	38
4.1.2	Confusion Matrix of Random Forest Classifier	39
5.2(a), 5.2(b)	Results of Model, Future Application	41
	References	44

## CHAPTER 1

### INTRODUCTION

This chapter explains the description of the project and the discovery of a malicious URL using Machine Learning. The authors' previous work on the Internet Security Center and other domains for malicious URL discovery is also mentioned. Online security aims to prevent malicious software and software attacks and protect your users from potential intruders. When we use the internet to make our job easier, at the same time, many attackers try to steal information from our system. There are many ways to attack malicious URLs. Block listing is common in anti-virus programs, blockchain/detection systems, and spam filters. The blocked listing method is straightforward and can provide the best accuracy if the list is updated on time, but this method may not work correctly to find harmful URLs that have just been produced. In present day Machine Learning is being used in various areas and cyber security is one of them. Machine learning plays a vital role in identifying malicious URLs in modern techniques for detecting malicious URLs. The URL represents the exact application location, showing resources at www. There are two parts to the same service acquirer: a) Username which means the domain name or IP address where the application is located. b) Protocol identifier indicating which protocol is used. Methods for Learning Machine, use some percentage of URLs a data for training based on statistical fundamentals, learn the prediction function to classify a URL as malicious or dangerous. This enables to create new URLs as opposed to blocking methods. The major specification for training a machine learning model is availability of training data. In the context of a malicious URL recovery, this will be accompanied by a set of large URLs. Machine learning can be divided as supervised, unsupervised, and semi-supervised, accompanied by having training data labels, no labels, and having labels for a limited portion of training data, respectively. Labels are associated with information that the URL is malicious or harmless. After the data required for training has been collected, we need to remove the instructive features to adequately describe the URL and at the same time, can be mathematically translated by machine learning models. Here, we begin by extracting the length, number, and binary features of the existing URLs in our database and then adding them as columns to our database and analyzing these features. Preliminary Data Processing and Data Visibility are performed to understand and

read the data. Machine learning classes like AdaBoost and Random Forest were used to separate dangerous and dangerous URLs. Voting categories were used for voting among dividers who provided the best possible accuracy. The random forest provides the highest accuracy of about 99%.

### **1.1. PROJECT DESCRIPTION**

Machine learning algorithms are used to implement the purpose of this project. Machine learning algorithms are not enough to work on big data and get a malicious URL in a better way with the best accuracy. The function of machine learning is to use historical data as embedded to predict new output values. AdaBoost and Random Forest algorithms are being used in this project. The voting category is used for voting between dividers which provides the best precision for the best classification of spam and ham URLs.

#### **1.1.1. DOMAIN OF RESEARCH**

Cyber security is a mechanism that protects electronic devices which are connected to internet from cyber attacks. Today internet technology has become an integral part of our education, entertainment, gaming, banking, and communications. In this modern digital age, it is much easier to have any information with a single click. But everything has its pros and cons, as we know in our tips but the internet is a platform for attack too. When we use the internet to make our job easier, at the same time many attackers try to steal information from our system. There are many ways to attack, malicious URLs. When a user visits a website, it is malicious and triggers a maliciously planned activity. Therefore, there are various ways to find harmful URLs online. Machine learning is one of them.

#### **1.1.2. REASON FOR CHOOSING THIS DOMAIN**

Cyber security is a practice that protects systems, networks, and systems from digital attacks. Malicious URLs or malicious websites are a serious problem for online security. In modern techniques for detecting malicious URLs, machine learning plays a vital role in identifying malicious URLs. The URL represents the same application location, showing resources at www. Many attackers are trying to steal information from our system. There are many ways to attack,

malicious URLs. The use of malicious URLs may result in illegal access to user data by the enemy. Given the rise spam URL distribution from the past decades, it is evident that there is a necessity to learn and implement strategies or mechanism to identify and avoid these spam URLs.

### 1.2. PROBLEM STATEMENT AND OBJECTIVE

- Detection of malicious URLs and classifying them as malicious and benign using various machine learning classifiers. Several feature extraction methods such as lexical features and host-based features are extracted better improve performance of the classifiers.

#### **Primary objective:**

- To build a binary classification model for detection and classification of malicious and benign URLs by improving the accuracy and processing time.

#### **Secondary objectives:**

1. Understanding the characteristics of spam and ham URLs.
2. To build a Machine Learning model to classify spam and ham URLs.
3. To select the best suited feature extraction techniques and classification algorithms for detection and classifying the malicious URLs and devise an improved method.

### 1.3. PURPOSE, SCOPE AND APPLICABILITY

#### 1.3.1. PURPOSE

The aim of this project is to compare models and find a better predictor model for better classification of malicious and benign URLs. The proposed predictor model attempts to improve the accuracy of prediction by using various features which take into account the interaction effect of different parameters.

### **1.3.2. SCOPE**

The scope of the malicious URL classification can be further extended to different domains like security, banking sector, safe and security purposes. In the banking sector to protect confidential information, transaction details, account details, user details and so on. Precautions can be taken if the malicious URLs are known prior.

### **1.3.3. APPLICABILITY**

The work can be applied to detect the various URLs on the internet and prevention measures can be taken.

## **1.4. SYSTEM REQUIREMENTS**

### **1.4.1. FUNCTIONAL REQUIREMENTS**

- Feature extraction and feature analysis techniques are carried to identify the most relevant features.
- Splitting the dataset into 80:20 ratios and training the model on the chosen data.
- Binary classification algorithms are used to predict the URLs are benign and malicious URLs.
- The requirement is to find best features which give better results compared to other models. So end users can find in the model to guess whether the URL is safe or not.

### **1.4.2. NON FUNCTIONAL REQUIREMENTS**

- Scalability -Even if the number of URLs and parameters increases, the model is expected to perform with the same level of efficiency.
- Reliability- Increase in the size of the dataset should not affect the accuracy of prediction.

### **1.4.3. HARDWARE AND SOFTWARE REQUIREMENTS**

- Hardware: The model can be implemented in I3 processors with 4GB Random Access Memory (RAM).

- Software: The project is implemented using Python 3.0 and higher. The implementation requires the use of following libraries in python:
- NumPy
- Pandas
- Scikit Learn
- Matplotlib
- Seaborn
- URL parser

### **1.4.4. CONSTRAINTS**

The system configured operating system should be Windows 8 or a higher version and the Python programming language should be version 3.0 or a higher version.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 REVIEWS

Cho Do Xuan et al.,[1] proposed A method for malicious URL detection using machine learning based on URL behaviors and attributes as well as the big data technology. The combination between easy-to-calculate attributes and big data processing technologies to ensure the balance of the two factors is the processing time and accuracy of the system. The results of this research can be applied and implemented in information security technologies in information security systems. The results of this article have been used to build a free tool to detect malicious URLs on web browsers. They used two supervised learning algorithms namely RF & SVM. RF algorithm achieves a high accuracy of about 96.28% and also maintains the balance between the processing time and accuracy of the system.

Anjali B et al.,[2] proposed A URL classification and identification model based on Naive Bayes classifier to detect and prevent cybercrime. Benign, Malware, Spamming and Phishing URLs were collected from the various sources for the experiment. Experiment is performed using Naive Bayes and is compared with SVM. It is observed that the proposed Naive Bayes model is more accurate than SVM for detection and identification of type of malicious URLs with the help of probability estimation tasks. Experiments on a large number of datasets show that these two algorithms take almost the same time to learn, but Naive Bayes reasoning is relatively faster. Experiment is performed only using Naive Bayes and is compared with SVM.

Vanitha et al.,[3] proposed The approach that classifies URLs automatically by using a Machine Learning algorithm called logistic regression that is used for binary classification. Feature extraction technique is used to detect the malicious URLs. Features Considered are \* Blacklist Queries \*Lexical Features. Feature Vectorizing methods were used. Tf-idf machine learning text feature extraction approach from the python module of sklearn is used. After Vectorizer data are arranged and distributed onto the term-frequency and inverse document frequency, which is

called as text extraction approach. Feature extraction and feature vectorization methods were used to detect the malicious URLs. It is observed that logistic regression obtains maximum learning accuracy compared to other algorithms such as Naïve Bayes, Random forest. Logistic regression achieves accuracy of about 98%.

Divya Kapil et al.,[4] experiments 4 algorithms using the weka tool and also compares the different detection techniques. ISC URL 2016 dataset was used for the experiment, which shows the results using performance metrics TPR, FPR, Precision, Recall and F-measure. Random numbers of samples were taken. Sample dataset contains 47 attributes. some features were rejected so that overfitting problems can be avoided. The dataset is in 'Malware', 'Spam', 'Benign', 'Defacement' and 'Phishing' form. Malware URLs are a critical issue for the researchers and machine learning techniques are very helpful in various areas. Malicious URLs detection using machine learning is a better idea than conventional techniques. Dataset is divided into 80% for training and 20% for testing. J48, Random forest, Bayes-Net and lazy classifiers were used and multi-class classification was performed, where classes were labeled as Defacement, Phishing, Spam, Malware and benign. It is observed that Random Forest achieves the highest TPR about 96% followed by Lazy classifier with 95% TPR.

Jino S Ganesh et al.,[5] identified Phishing URLs under weak supervision, which requires a small amount of labeled data to start the learning process. They implemented a new hybrid model which combined NLP (natural language processing-based features) and word vector. Two parallel modules are used that extract functional representations of URLs. The first is the character level CNN module. The other is an attention-based hierarchical RNN module that is proposed to find phishing URL detection.

The Detecting process is based on: Finding the data, retrieving and summarizing the data.

Making the prediction based on the analysis data. Calculating the probabilities of the specific results. Adapting to certain development autonomously. Optimizing the process based on the recognized pattern. Four different machine learning algorithms are used, like logistic regression, decision tree, random forest, and multilayer perceptron neural networks. Random forest achieves the highest accuracy about 98.6%.



R. Naresh et al.,[6] identified Malicious uniform resource locators employing a combination of URL lexical options, payload size, and python supply options. Feature extraction techniques such as Host-based features, lexical based features and popularity based features were used to detect the malicious URLs. Feature extraction technique is used to detect the malicious URLs. Detecting process: Web Crawling, Feature extraction and processing, Training of classifiers and Running classification to detect malicious URLs. URLs were collected from the Alexa ranking website. Numbers of URL (400,000) out of which 80,000 were malicious and others clean, this makes our data set. The research targets mainly on domain-name and URL's attributes. Classifiers used are SVM and Logistic Regression. They used a Support Vector Machine with a polynomial kernel and logistic regression to attain maximum accuracy. Logistic regression achieves accuracy about 98%.

Rajesh Kumar et al.,[7] used Black and white list technology and machine learning algorithms and formed a multilayer filtering model for detection of malicious URLs. The model was trained for each machine learning algorithm i.e., naive Bayesian classification and decision tree classifier threshold and this threshold is used to refer to guide two classifiers for filtering URL. Naive Bayesian classifier, Decision Tree classifier and SVM classifiers were combined in one multi-layer model to improve the malicious URL detection system in terms of accuracy. The Multi-layer filter model performs better than all the three classifier models achieving the accuracy about 79.55%.

Gopinath Palaniappan et al.,[8] explored An active DNS analysis approach for classifying a domain name as benign or malicious by including the web-based features of the domain name in addition to the usually used lexical-based and DNS-based features. They extracted features of a domain name under DNS-based, web-based, blacklisting and lexical-based categories, and trained a logistic regression classifier and tested the classifier to classify unlabeled dataset of domain names and got an accuracy of about 60% using a small dataset of about 10000 domain names. The usage of web-based features of domain names in addition to using blacklists, DNS data, and lexical features to identify malicious domains has been shown.

Apoorva Joshi et al.,[9] proposed A static lexical feature-based Random Forest Classification approach to classify malicious and benign URLs. Ensemble machine learning approach is used. A Random Forest model for URL classification using purely static lexical features extracted from the URL string. Methodology used is: Business Understanding, Data Understanding, Data Preparation, Data Modeling, Evaluation and Deployment. The results of the Random Forest model were compared against simple machine learning classifiers such as Naïve Bayes, SVM and Logistic Regression, as well as other ensemble classifiers like AdaBoost and Gradient Boost. Random Forest was the best choice for the classification, given that it has the highest accuracy about 92% and lowest FNR among the models compared.

### **2.2. GAPS IDENTIFIED**

1. There are very few ensemble models built on detection of malicious URLs.
2. Some data sets have incorrect URLs and may result in data imbalance and some of the works did not perform over sampling strategies to measure imbalanced data.
3. In most of the research, the Random Forest algorithm is used and compared which gives high accuracy.

## CHAPTER 3

### METHODOLOGY

#### 3.1. DATA DESIGN

##### 3.1.1. DATASET COLLECTION

The malicious URL dataset used in this model is imported from Kaggle. It comprises 450176 rows and 4 columns. Column names are Unnamed, URL, Label and Result.

Unnamed: 0		url	label	result
0	0	https://www.google.com	benign	0
1	1	https://www.youtube.com	benign	0
2	2	https://www.facebook.com	benign	0
3	3	https://www.baidu.com	benign	0
4	4	https://www.wikipedia.org	benign	0
...	...	...	...	...
450171	450171	http://ecct-it.com/docmmnn/aptgd/index.php	malicious	1
450172	450172	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1
450173	450173	http://faboleena.com/js/infortis/jquery/plugin...	malicious	1
450174	450174	http://atualizapj.com/	malicious	1
450175	450175	http://writeassociate.com/test/Portal/inicio/l...	malicious	1

50176 rows × 4 columns

**Fig 3.1.1: Overview of the dataset**

This dataset has been taken from Kaggle for analysis purposes. It doesn't consist of any null values and the Unnamed column will be dropped in the later stage of data preprocessing as it is not required for any kind of analysis.

##### 3.1.2. DESCRIPTION OF DATASET

###### 3.1.2.1. NAME OF DATASET

The dataset named urldata.csv has been taken from Kaggle for our experimental purpose. This

dataset was created to tackle the problem of malicious URLs on the cyber world. The malicious URLs can be detected using the lexical features along with tokenization of the URL strings.

### **3.1.2.2. NUMBER OF CLASSES**

The dataset has 7043 450176 rows and 4 attributes. There are two types of classes in the dataset. They are benign and malicious URLs. The size of the dataset is 34,595 KB and each URL can be identified with a unique id. A URL in the dataset can belong to either benign or malicious class.

### **3.1.3. DATA PREPARATION**

#### **3.1.3.1. HANDLING NAN VALUES**

There are no null values in any of the columns in the dataset.

#### **3.1.3.2. REMOVING NOISE**

The exact features have the values in 1's and 0's so there is no need to convert them into standard format.

#### **3.1.3.3. FORMATTING THE DATA**

The exact features have the values in 1's and 0's so there is no need to convert them into standard format. The classes benign and malicious have corresponding values 1 and 0 in the result column.

#### **3.1.3.4. HANDLING IMBALANCED DATA**

The target column for data retrieval of malicious URL data contains imbalance categories. The data therefore needs to be balanced using the oversampling method. Otherwise, the model will learn more in the benign majority class.

### **3.1.4. FEATURE EXTRACTION**

To extract lexical features, count features and binary features URL parsing is done with the help of URL parser library and then applied lambda function. To extract binary features we have used

regular expressions. In total 18 features are extracted.

**Lexical features are:** Length Features - 'URL Length', 'Hostname Length', 'Path Length', 'Top Level Directory Length', 'First Directory Length'

- **Length of URL:** This feature extracts the length of the URL using the 'len' function of String.

```
#Length of URL
df['url_length'] = df['url'].apply(lambda i: len(str(i)))

print(len(str("https://www.youtube.com")))

23
```

Fig. 3.1.4.(a): Length of URL

- **Length of Host-name:** This feature extracts the hostname of the URL. Host name is the Domain name of a URL.

```
#Hostname Length
df['hostname_length'] = df['url'].apply(lambda i: len(urlparse(i).netloc))

len(urlparse("https://www.youtube.com").netloc)

15
```

Fig. 3.1.4.(b): Length of Hostname

- **Length Of Path:** This feature extracts the length of the path. The URL path is the string of information that comes after the top-level domain name.

```
#Path Length = No. of Directories
df['path_length'] = df['url'].apply(lambda i: len(urlparse(i).path))

len(urlparse("http://www.rosespa.com.sg/ipic/Dirk/index.php").path)

20
```

Fig. 3.1.4.(c): Length of path


- **Length Of First Directory:** This function extracts the length of the first directory in the path of the URL. The first directory is usually found after the top-level domain separated by a '/'.  
The screenshot shows a Python function named fd\_length that takes a URL as input. It uses urlparse to get the path and then splits it by '/' to find the first directory. For the URL 'http://www.rosespa.com.sg/ipic/Dirk/index.php', it returns 4, which is the length of the first directory 'ipic'.

Fig. 3.1.4.(d): Length of First Directory

- **Length of TLD:** This feature extracts the length of Top-Level Domain of a given URL. A Top-Level Domain is the ending component of a domain name. It is the suffix of extension linked to a website.

The screenshot shows a Python function named tld\_length that takes a TLD as input. It uses len to get the length of the TLD. For the TLD 'com.sg', it returns 6. The output shows the top level directory of the URL and the length of the TLD.

```
url = "http://www.rosespa.com.sg/ipic/Dirk/index.php"
tld = get_tld(url, fail_silently=True)
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1
print("Top Level Directory of 'http://www.rosespa.com.sg/ipic/Dirk/index.php' :", tld)
print("Length of tld:", tld_length(tld))
```

Top Level Directory of 'http://www.rosespa.com.sg/ipic/Dirk/index.php' : com.sg  
Length of tld: 6

Fig. 3.1.4.(e): Length of Top-Level Domain

**Count Features are:** Count of : '-', '@', '?', '%', '.', '=', 'http', 'www', 'Digits', 'Letters', 'Number of Directories'.

- As the name of the features suggest, each of these features extracts the count of particular characters / digits / letters in a given URL.
- **Count of Number of Directories:** This feature extracts the number of directories available in the URL. It is simply calculated by the count of '/' in the path of the URL as the directories are separated by a '/'.

```
def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')
df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))

no_of_dir("http://www.rosespa.com.sg/opic/Dirk/index.php")

3
```

Fig. 3.1.4.(f): Count of Directories

**Binary features are:** ‘If IP address is used in URL’ and ‘If URL shortening is used in URL’.

- **Use of IP or not:** This is a binary feature which extracts whether an IP address is used in a particular URL or not. It is filtered based on the general IP characters used in present day scenarios.

```
#Use of IP or not in domain
def having_ip_address(url):
    match = re.search(
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.',
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\./' # IPv4
        '([0x[0-9a-fA-F]{1,2})\\.([0x[0-9a-fA-F]{1,2})\\.([0x[0-9a-fA-F]{1,2})\\.([0x[0-9a-fA-F]{1,2})\\./' # IPv4 in hexadecimal
        '([a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url) # IPv6
    if match:
        #print match.group()
        return -1
    else:
        print ("No matching pattern found")
        return 1
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))

having_ip_address("http://www.rosespa.com.sg/opic/Dirk/index.php")

No matching pattern found

1
```

Fig. 3.1.4.(g): Presence of IP Address

- **Use of Shortening URL or not:** This feature extracts information whether the URL is shortened or not. A URL is shortened to hide the path and track activities after the shortened URL is clicked. Some of the most used URL shortening services are used for extracting this feature.

```
def shortening_service(url):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
        'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
        'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'
        'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
        'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
        'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'
        'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|'
        'tr\.im|link\.zip\.net',
        url)

    if match:
        print("Url Shortened")
        return -1
    else:
        return 1
# df['short_url'] = df['url'].apply(lambda i: shortening_service(i))

shortening_service("https://tinyurl.com/4swk97wy")

Url Shortened

-1
```

**Fig. 3.1.4.(h) : Shortening of URL**

### Dataset after Feature Extraction:

url	label	result	url_length	hostname_length	path_length	fd_length	tid_length	count-	count@	...	count%	count.	count=	count- http	count- https
://www.google.com	benign	0	22	14	0	0	3	0	0	...	0	2	0	1	1
/www.youtube.com	benign	0	23	15	0	0	3	0	0	...	0	2	0	1	1
www.facebook.com	benign	0	24	16	0	0	3	0	0	...	0	2	0	1	1
is://www.baidu.com	benign	0	21	13	0	0	3	0	0	...	0	2	0	1	1
/www.wikipedia.org	benign	0	25	17	0	0	3	0	0	...	0	2	0	1	1

11 columns

**Fig 3.1.4(i): Overview of the dataset after feature extraction**

### 3.1.5. ALGORITHMS

The classifiers like AdaBoost and Random Forest algorithms were used in model building which were later given into the voting classifier to check the best model for our dataset.

#### 3.1.5.1. Adaptive Boosting Algorithm (AdaBoost):

Adaptive Boosting Algorithm, popularly known as AdaBoost Algorithm is one of the Machine Learning methods used as an Ensemble Method. The most common algorithm used with AdaBoost is single-level decision trees which means for Decision trees with only 1 division. These trees are also called Decision Stumps. This algorithm creates a model and provides equal



weights for all data points. It then gives us high weights on points that are poorly organized. Now all the points with the highest weight are given extra value in the next model. It will retain the training models until further notice without a minor error.

### **3.1.5.2. Random Forest Algorithm:**

The Random Forest is a compiler that contains a multiple decision trees for multiple subsets of the original dataset and takes averages known as aggregations to improve the performance metrics of that dataset. These subsets are called bootstrapped datasets as they are made into subsets using bootstrapping method. Rather than depending on a single decision tree, the random forest algorithm takes an output from decision trees of each bootstrapped dataset and gives the final output based on many predictable votes. Since the random forest includes many trees to predict the class of the given input, there is a possibility that some decision trees might give the appropriate output, while others may not give the appropriate output. But, all the trees predict the appropriate outcome. Therefore, the two assumptions for an ideal Random forest classifier are as follows:

- Dataset should contain appropriate values so that the classifier can predict accurate results rather than the guessed result.
- The outcomes from each tree must be very weakly correlated.

### **3.1.5.3. Voting Classifier:**

A voting classifier is a Machine Learning ensemble method which is trained on multiple models and gives an output(class) which is based on the probability of that class being predicted is high. The prediction from each classifier are sent to voting classifier which combines the predictions and gives final predictions based on majority of votes. In this way, we can eliminate use of multiple models to predict multiple outcomes by passing the predictions from multiple classes to the voting classifier which predicts based on majority of votes. There are two types of votes which are supported by voting classifier.

1. **Hard Voting:** In hard voting, the final predicted outcome is the outcome which has majority of votes, i.e., the outcome which has highest probability of being predicted by individual

classifiers. Suppose there are three classifiers which predicted the outcome class as (2,1,2) respectively. So, here the majority of classifiers predicted 2 as outcome. Hence 2 will be the final prediction outcome.

2. **Soft Voting:** In soft voting, the final prediction outcome is based on the average of the probabilities which are given to that class by each of the classifiers. Suppose, for the inputs provided to the classifiers, the prediction probability for class 1 = (0.20, 0.37) and class 2 = (0.11, 0.53). So the average for class 1 is 0.285 and class 2 is 0.32. So, it is evident that class 2 has highest average probability. Hence, the prediction outcome of the voting classifier through soft voting is class 2.

## 3.2. MODEL BUILDING

### 3.2.1. REVIEW OF MACHINE LEARNING ALGORITHMS

Machine Learning is a branch of Artificial Intelligence that deals with the software applications to predict the outcomes more accurately without any manual intervention. They use historical/previous data as an input to predict the possible outputs. Machine learning has supervised, unsupervised and semi-supervised learning algorithms. In this project, we have used supervised Machine Learning Algorithms like AdaBoost, Random Forest and Voting Classifiers to identify and classify the URLs as spam or ham.

### 3.2.2. MACHINE LEARNING MODEL

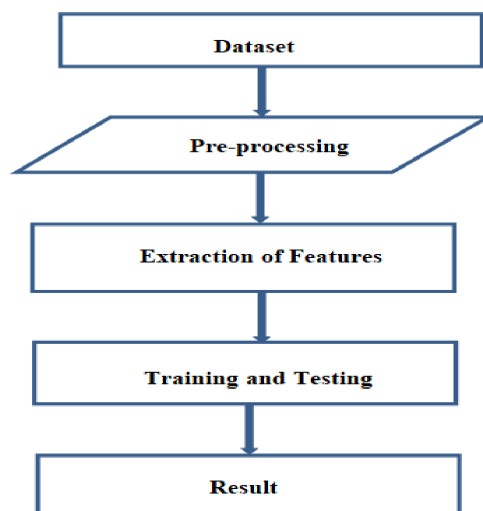
#### 3.2.2.1. MODEL CATEGORY & RATIONALE FOR CHOOSING THE MODEL CATEGORY

The model category of AdaBoost and Random Forest model is a supervised technique. The data which is passed into the model has labels with malicious and benign features in the target column. The model is being trained on the labeled data provided as input during the training phase. The supervised model analyzes the training data and predicts the class of data provided during the test phase.

### 3.2.2.2. MODEL NAME & RATIONALE FOR CHOOSING THE MODEL NAME

The malicious URL detection and classification model is a binary classification model because the target column has two classes, viz., benign and malicious. The malicious URL data set used for the research includes around 3 predictor variables and a target variable, later feature extraction is carried out in which 17 features are taken from the provided URLs depending on the behavior of the URLs. The target is to detect and classify the binary value viz., malicious or benign for a particular observation. The purpose of the study is to build a Machine learning model which is capable of predicting the class of a new observation as either spam or ham based on the knowledge gathered during the learning phase. Since there are only two target classes, this binary classification model is appropriate to achieve the objective of the research.

### 3.2.2.3. MODEL ARCHITECTURE



**Fig 3.2.2.4.: Steps followed by a machine learning model.**

### 3.2.2.4. STEPS IN BUILDING THE MODEL

(i) Data Encoding: This is the initial step in building the model, the feature extraction carried in such a way that the output are binary values (0 and 1).

(ii) Feature Extraction: Length, count and binary features have been extracted and added to our dataset using lambda function.

(iii) URL parser: This is a freely available tool/parser which is used in this project to split the given URL into separate parts like hostname length, path length, TLD length, etc.,.

(iv) Tokenizing URL: “In Python tokenization basically means separating text into small lines, words or even creating non-English language words. Various token functions are built into the nltk module itself and can be used in programs. Here URLs are subdivided into smaller fields for data analysis”.

(v) Data Scaling: No need for data scaling as there were no such columns or data was present.

(vi) Training and Test Set: The fifth step is to split the data into train data and test data using the ‘train\_test\_split’ function available at the ‘sklearn’ library. The classification rate is selected at 80:20, which means that 80% of the data is selected for training and the remaining 20% is selected for the evaluation of new comments and class decisions. A high percentage of training data makes the model train better.

(vii) Balancing Data: The dataset is highly imbalanced with 76.80% of benign URLs and 23.20% of malicious URLs. The dataset has to be balanced before training the model. Hence as the next step, balancing of training data is done using the oversampling technique. If the data is imbalanced, the model will be biased towards the majority variable.

(viii) Feature Selection: The fifth step is to select the key features of the model. This step plays a crucial role as it is very important to find the most relevant features related to the machine learning model.

(ix) Model\_Building: The last step is to build a binary classification model to detect and classify the benign and malicious URLs. AdaBoost classifier, Random Forest classifier are used, compared and voted using the voting classifier which clearly proves that Random Forest Classifier gives better performance metrics.

(x) Cross-Validation: A 5-fold cross-validation strategy is implemented to validate the Machine Learning model to check the validity of the results.

(xi) Detection: Function has been created and different URLs were provided as user input to

detect and classify whether the given URL belongs to a benign class or malicious class. If the given URL is identified as malicious, then an alert message box is displayed, and a "Safe URL" message is printed.

### 3.2.3. MODEL TRAINING

In the training phase, we train the algorithms to find spam or malicious websites found in the browsers we use. When we give a web address as an input to the model, the trained algorithms check whether the website or URL is safe or not and display a message. If the URL is safe, "Safe URL, Happy Browsing!!" message will be displayed. If the URL is spam or malicious, "Malicious Alert, proceed with caution!" message will be displayed. The types of Machine Learning we used to identify harmful URLs are described in section algorithm 3.1.5.

### 3.2.4. MODEL VALIDATION

#### 3.2.4.1. VALIDATION STRATEGIES

```
# 5-fold cross-validation before oversampling
from sklearn.model_selection import cross_val_score
print("before oversampling:")
print(cross_val_score(est1, x1, y1, cv=5, scoring='accuracy').mean())
```

```
before oversampling:
0.9910568751649425
```

```
# 5-fold cross-validation after oversampling
from sklearn.model_selection import cross_val_score
print("after oversampling:")
print(cross_val_score(est, x, y, cv=5, scoring='accuracy').mean())
```

```
after oversampling:
0.9941559986118156
```

**Fig 3.2.4.1: Validation Snippet**

The model is validated through cross-validation technique. 'cross\_val\_score' is imported from 'sklearn.model\_selection' library. 5-fold cross-validation has been implemented on the machine learning model to check the validity of the model, that is to check whether the model is predicting the correct output. We have scored 99% validation accuracy and hence our model is good to go.

### 3.3. MODEL IMPLEMENTATION

#### 3.3.1. DATA INVESTIGATION

Machine Learning is a branch of Artificial Intelligence that deals with the software applications to predict the outcomes more accurately without any manual intervention. They use historical/previous data as an input to predict the possible outputs. Machine learning has supervised, unsupervised and semi-supervised learning algorithms. Machine learning activities allow us to work with your data set at different stages of the data analysis process:

- Preparing models
- Training models
- Evaluating models
- Applying models
- Managing models

The dataset is collected from Kaggle which comprises of 4,50,176 rows and 4 columns. Machine Learning algorithms proved that efficient classification of malicious and benign URLs can be done using machine learning classifiers like Random Forest algorithm and AdaBoost algorithm. Random Forest algorithm achieves highest accuracy of 99.8% when compared to AdaBoost classifier which yields the accuracy of 99.5%. Voting classifier was used to check the model which is giving the highest accuracy.

#### 3.3.2. DATA PREPARATION

In the data preparation part, the behavior and features of URLs are studied and noted. Feature extraction is done based on the length, count and binary features. In total 18 features were extracted and features are as follows:

**Lexical features are:** 'URL Length', 'Hostname Length', 'Path Length', 'Top Level Directory Length', 'First Directory Length'

**Count Features are:** Count of: '-', '@', '?', '%', '.', '=', 'http', 'www', 'Digits', 'Letters', 'Number of Directories'.

**Binary features are:** 'If IP address is used in URL ' and 'If URL shortening is used in URL '.

## 3.3.2.1. DATA VISUALIZATION

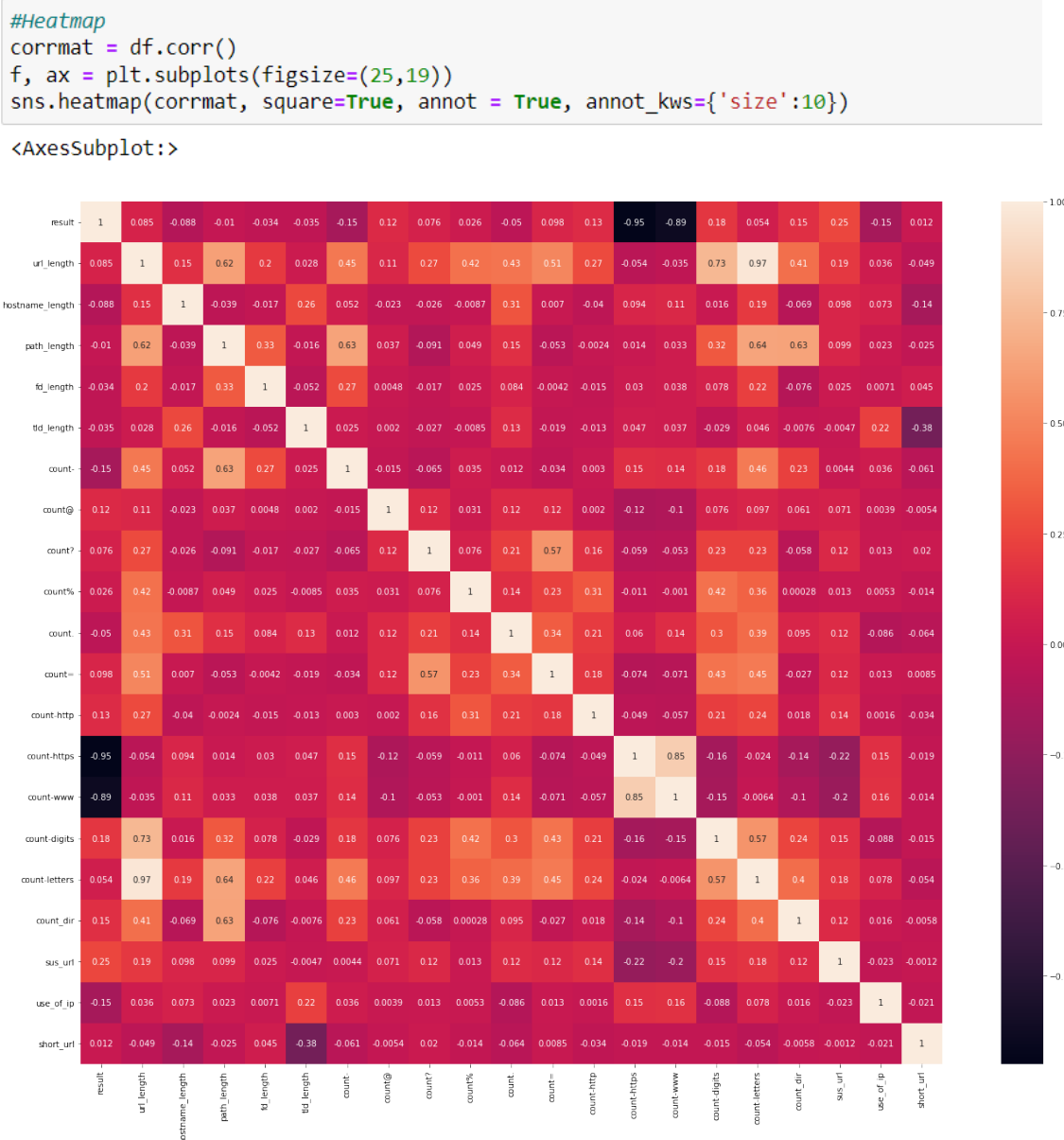


Fig 3.2.2.1: Heatmap of the extracted features

**Heat Map:** “A heatmap is colorful representation of information in Multi-dimensional formation. It is used in visualization of simple or complex information. Heatmaps are widely used across various fields such as defense, customer behavior analysis and marketing platforms”. The correlation between the features can be visualized and understood from the heat map that has been drawn which has been extracted from the URLs.

### 3.3.2.2. DATA SPLITTING

The entire data was taken into consideration to train the model in which the rows size increased from 4,50,176 to 6,91,476 after doing oversampling to balance the dataset. The dataset was split into train and test data in 80:20 ratio. The smaller part, 20% of data remains unseen which is used for testing the model and the majority of data 80% is used for training the Machine Learning Model.

### 3.3.3. MODEL TRAINING & VALIDATION

The dataset was split into train and test data in 80:20 ratio. 20% of data remains unseen which is used for testing the model and 80% of data is used for training the Model.

```
#Predictor Variables
x = test_over[['hostname_length',
               'path_length', 'fd_length', 'tld_length', 'count-', 'count@', 'count?',
               'count%', 'count.', 'count=', 'count-http', 'count-https', 'count-www', 'count-digits',
               'count-letters', 'count_dir', 'use_of_ip']]

#Target Variable
y = test_over['result']
```

**Fig.3.3.3 (a): Predictor and Target Variables**

```
Before oversampling:
Shape of x1_train: (360140, 17)
Shape of x1_test: (90036, 17)
Shape of y1_train: (360140,)
Shape of y1_test: (90036,)

#Splitting the data into Training and Testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print("After oversampling:")
print("Shape of x_train: ", x_train.shape)
print("Shape of x_test: ", x_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_test: ", y_test.shape)

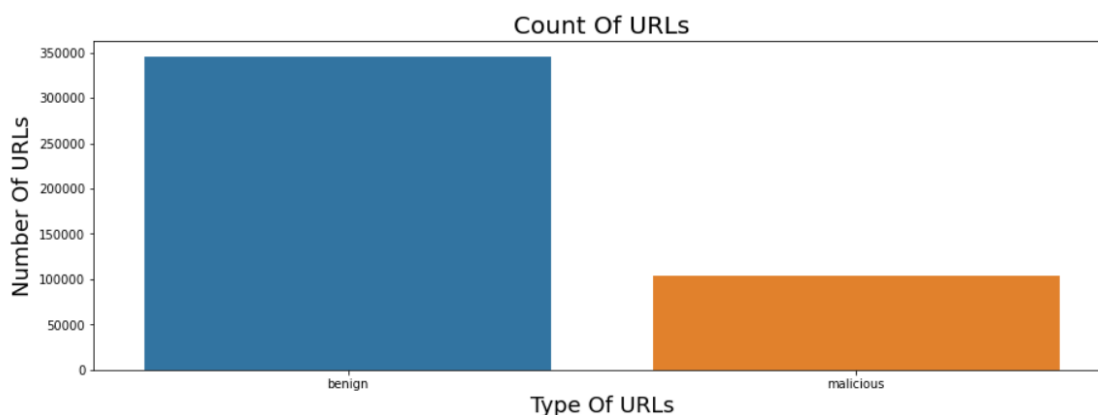
After oversampling:
Shape of x_train: (553180, 17)
Shape of x_test: (138296, 17)
Shape of y_train: (553180,)
Shape of y_test: (138296,)
```

**Fig 3.3.3.(b): Splitting of train and test data**



### 3.3.3.1. DATA BALANCING

The malicious URL dataset that has been taken from kaggle had an imbalance in classes such as benign and malicious. The Percentage of malicious URLs was 23.20 %, and the percentage of benign URLs was 76.80 %. So to balance the dataset oversampling technique has been applied.



```
print("Percent Of Malicious URLs:{:.2f} %".format(len(df[df['label']=='malicious'])/len(df['label'])*100))
print("Percent Of Benign URLs:{:.2f} %".format(len(df[df['label']=='benign'])/len(df['label'])*100))
```

```
Percent Of Malicious URLs:23.20 %
Percent Of Benign URLs:76.80 %
```

**Fig. 3.3.3.1 (a):** Figure shows imbalance in the dataset

```
X = np.array(df.drop('result',axis=1))
y = np.array(df['result'])

print(X.shape,y.shape)

(450176, 21) (450176,)
```

**Fig 3.3.3.1 (b):** size of dataset before oversampling

```
class_0 = df[df['result']==0]
class_1 = df[df['result']==1]

print(class_0.shape, class_1.shape)

(345738, 22) (104438, 22)

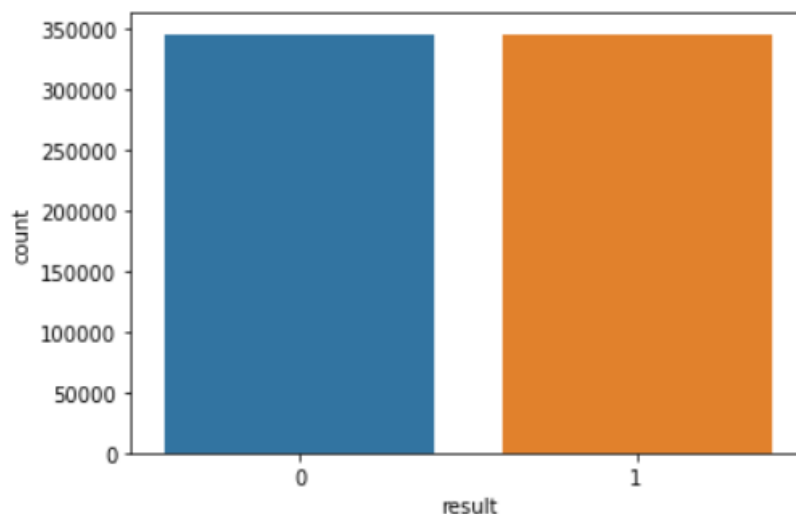
class_1_over = class_1.sample(class_count_0, replace=True)
class_1_over.shape

(345738, 22)

test_over = pd.concat([class_0, class_1_over], axis=0)
test_over.shape

(691476, 22)
```

**Fig 3.3.3.1 (c): Oversampling technique**



**Fig 3.3.3.1 (d): Data after oversampling**

**Oversampling:** If the data is imbalanced, i.e. size of one class is more than the size of another class in the data, we apply oversampling technique. Oversampling involves introducing a bias to select samples the minority class more than the majority class to make the data balanced. In our case, the malicious URLs are less in number (23%) when compared to benign URLs (74%). So, we have performed Oversampling technique which has increased the size of malicious URLs as the same size of Benign URLs by randomly selecting samples from malicious class and replacing them with multiple copies in the dataset. Therefore, it is possible that a single instance may be

selected multiple times.

### 3.3.3.2. CROSS-VALIDATION

The opposite validation is a mathematical method used to measure the ability of machine learning models. It is often used in machine learning to compare and select a given predictive model because it is easier to understand, easier to use, and results in skill measurements that are often less biased than alternatives. The K-fold cross-validation is a process used to measure model ability in new data. The process has one parameter called k which refers to the number of groups the sample data provided to be divided into. Thus, the process is often called k-fold cross-validation. If a certain number of k is selected, it can be used instead of k in the reference, as k = 5 becomes 5-fold cross-validation.

```
# 5-fold cross-validation before oversampling
from sklearn.model_selection import cross_val_score
print("before oversampling:")
print(cross_val_score(est1, x1, y1, cv=5, scoring='accuracy').mean())
```

```
before oversampling:
0.9910568751649425
```

```
# 5-fold cross-validation after oversampling
from sklearn.model_selection import cross_val_score
print("after oversampling:")
print(cross_val_score(est, x, y, cv=5, scoring='accuracy').mean())
```

```
after oversampling:
0.9941559986118156
```

**Fig 3.3.3.2: Cross-validation code snippet**

5-fold cross-validation has been applied to validate the model, and the cross-validation achieved 99% accuracy this shows that our model is not over fitted and is good to go.

### 3.3.3.3. MODEL VALIDATION STRATEGIES

Malicious URL dataset is split in 80 to 20 ratio, i.e., 80% of data is used for training the model and 20% of the data is used for testing which the model is being trained on 80% of the data and

20% of unseen data is provided as test data. The binary classification is done using Random Forest algorithm and AdaBoost algorithm later on which the voting classifier is applied to check the model with the highest accuracy. The Random Forest algorithm gives about 99.8% accuracy and the AdaBoost classifier gives about 99.5% accuracy. Hence the Random Forest algorithm performs better than the AdaBoost classifier. Train accuracy and test accuracy has also been checked where the model produced 99% of accuracy from each of training and testing data. A 5-fold cross-validation technique has been applied to validate the model in which the cross-validation accuracy is observed to be 99%. As the oversampling technique is used to balance the dataset and cross-validation is applied to validate the model the over fitting problem doesn't occur.

## CHAPTER 4

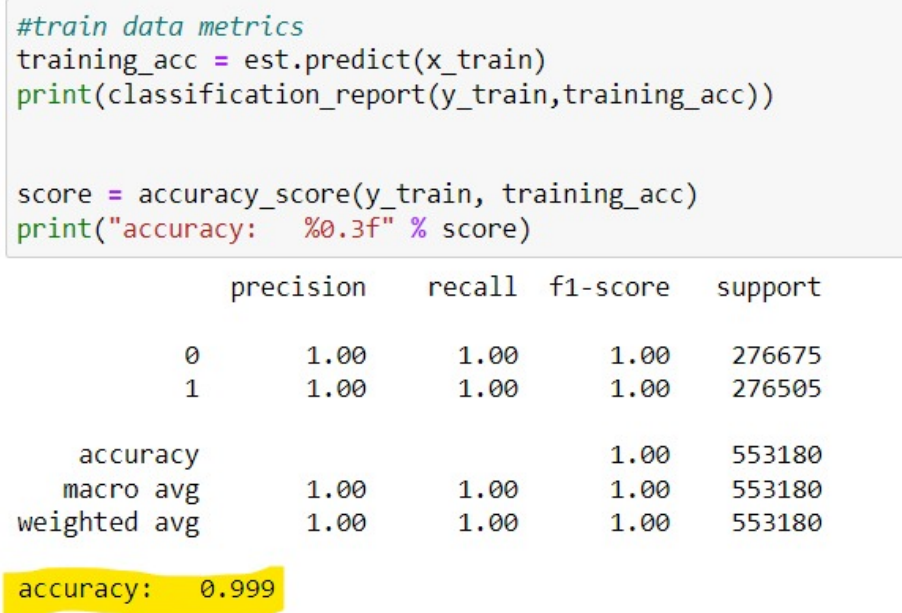
### DISCUSSION / ANALYSIS OF RESULTS

After model building and training on known data, the machine learning model is to be tested on new/unseen data. Then the model is evaluated according to the performance on unseen data. The metrics used are accuracy, precision, recall and f1-score.

#### 4.1. PERFORMANCE METRICS

##### 4.1.1. CLASSIFICATION REPORTS

For the first model, after training, the models evaluated on unseen data. As shown in Fig. 4.1.1(a) & (b), the training accuracy is 99 percent and test accuracy is 99 percent.



**Fig 4.1.1 (a): Shows the training accuracy**

```
#test data metrics
y_pred = est.predict(x_test)
print(classification_report(y_test,y_pred))

score = accuracy_score(y_test, y_pred)
print("accuracy:  %.3f" % score)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	69063
1	1.00	1.00	1.00	69233
accuracy			1.00	138296
macro avg	1.00	1.00	1.00	138296
weighted avg	1.00	1.00	1.00	138296

```
accuracy:  0.999
```

**Fig 4.1.1 (b): Shows the test accuracy**

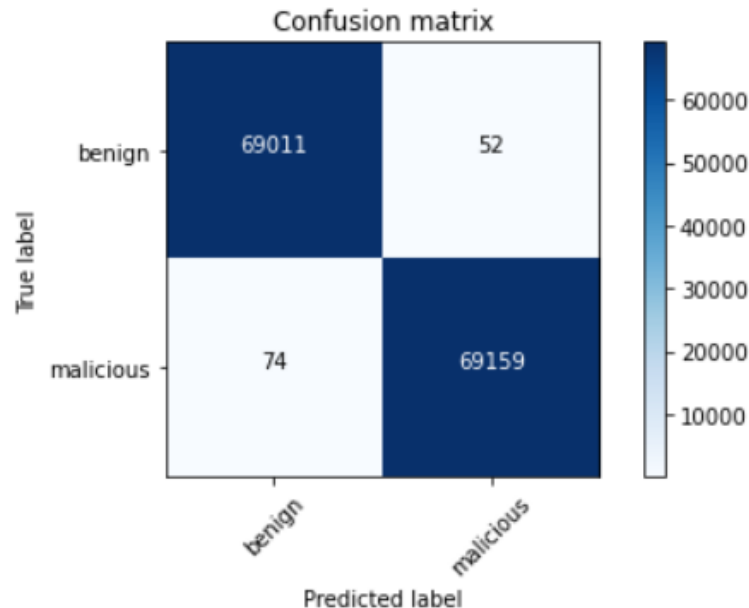
The training accuracy of the model is 99% and the test accuracy is also 99%. Hence we can ensure that there is no overfitting in the model. Also, we have applied a 5-fold cross-validation strategy to cross check the validity of the model. Both malicious and benign URLs were given as user input to the model to validate the model. The model is observed to give the best accuracy and predicts better when tested.

### 4.1.2. CONFUSION MATRIX

A confusion matrix is a multi-dimensional square matrix which is used to test the performance of the given classification model. The size of matrix is the number of target classes. The confusion matrix compares the actual target values to the values that are predicted by the model.

```
import itertools
cm = confusion_matrix(y_test, y_pred, labels=[0,1])
plot_confusion_matrix(cm, classes=['benign', 'malicious'])
```

Confusion matrix, without normalization



**Fig 4.1.2: Confusion Matrix**

The confusion matrix for the random forest classifier is as shown in Fig. 4.1.2

- **True positive(s) (TP):** These are the links in which we have predicted benign and are actually benign.
- **True negative(s) (TN):** The links which we have predicted spam/malicious, and they are actually spam/malicious.
- **False Positive(s) (FP):** The links that we have predicted malicious, but they are actually benign. (Also known as “Type I error”)
- **False negative(s) (FN):** The links which we have predicted benign, but they are actually malicious. (Also known as “Type II error”)

## CHAPTER 5

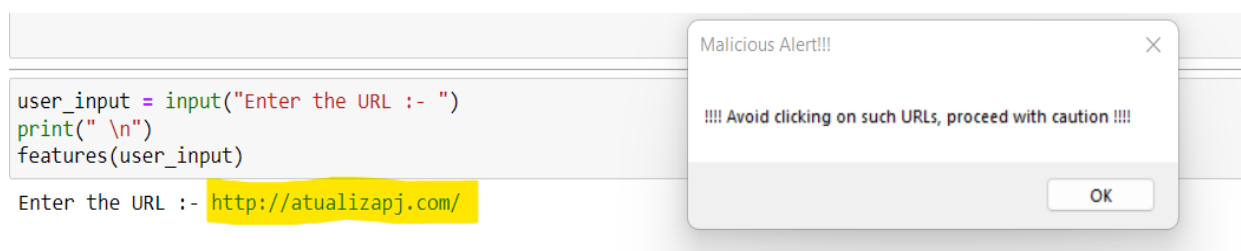
### CONCLUSION

In this chapter, results of models, future recommendations and the issues faced while building the model are explained.

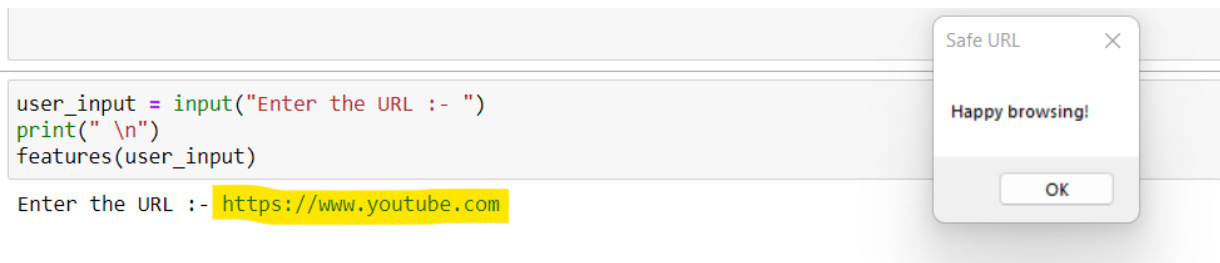
#### 5.1. RESEARCH PROJECT OVERVIEW

The first chapter describes the project description, scope of work, applicability and system requirements that are required to implement the project. The chapter also includes the functional requirements such as feature selection, non-functional requirements such as scalability and reliability and the hardware and software requirements to implement the model. In the second chapter, We have discussed previous work done by the authors in malicious URLs detection and other domains of cyber security and also stated the gaps identified. The third chapter discusses methodologies such as how the required data is acquired and the number of classes such as benign and malicious URLs. Also, the data pre-processing techniques such as features extraction, formatting of data and data balancing are discussed. The model architecture, steps in building the model, processes such as data scaling, splitting, training and validation of data are explained in this chapter. The fourth chapter involves discussion and analysis of results of the model built in the previous chapter. The fifth chapter discusses the advantages and limitations of the project and future recommendations. The final chapter of the project is the Appendix, which consists of References used in the above chapters.

#### 5.2. EVALUATION OF RESULTS







**Fig 5.2(a): Results of the model**

Malicious URL detection model does the binary classification using machine learning classifiers namely Random Forest and AdaBoost classifiers. The voting classifier is used to check the model that is giving highest accuracy amongst all the models. The results show that the Random Forest algorithm performs well compared to AdaBoost classifier. The Random Forest classifier gives accuracy about 99.8% whereas AdaBoost classifier gives accuracy about 99.5% and hence voting classifier predicts that the Random Forest algorithm performs well.

The function has been created which identifies the URL as a spam or ham URL. The URLs are accepted as user input and detected whether they are benign or malicious ones. If the URL is found malicious then an alert message box will be displayed showing that “Avoid clicking on such URLs” else “Safe URL” message will be printed.

For the URLs that are found malicious, some precautionary measures can be carried out, such as blacklisting of URLs and red-listing the URLs so that they don't appear anymore in the future.

```
import re
def suspicious_words(url):
    match = re.search('PayPal|login|signin|bank|account|update|free|lucky|service|bonus|ebayisapi|webscr',
                      url)
    if match:
        return 1
    else:
        return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))
```

**Fig 5.2(b): Future Implementation**

This is an example of how this project can be implemented to specific domains. The above function shows how to customize the project depending on the trigger words which are malicious

to particular domains. The terms like free, lucky, bonus, etc. are most commonly used trigger words in phishing attacks.

### **5.3. IMPLEMENTATION & ISSUES**

- Availability of large amount of data is difficult.
- To get better performance, selection of features has a crucial role.
- It is very difficult to trace the botnets which are made by malware due to the large size of the web.
- Reducing the processing time of the model.
- Imbalance dataset can be a classification issue.

### **5.4. ADVANTAGES & LIMITATIONS**

#### **5.4.1. ADVANTAGES:**

- Machine Learning algorithms are efficient to do binary classification and to detect the malicious URLs.
- Advantages of malicious URL detection using a machine learning model is that it accepts the URL as user input and detects and classifies it as benign or malicious one. If the URL is malicious then the alert message box opens stating that “Avoid clicking on such URLs”, else “Safe URL” is printed.
- Model does binary classification with 99% accuracy. This model can be used in the cyber security domain and also to avoid digital attacks by knowing the malicious and benign URLs in prior. Safety measures can be taken if the URL is found malicious.

#### **5.4.2. LIMITATIONS:**

- Every application requires that it should be trained specifically.
- Need huge quantity of customized, structured training data.

- The Learning model is required to be supervised learning i.e. Training data must be labelled/structured data.
- Needs rigorous training.

### **5.5. FUTURE WORK & RECOMMENDATIONS**

- To build ready-to-use machine learning models for detection of Malicious URLs.
- Blacklisting of malicious URLs that have been detected as malicious ones.
- To make domain specific applications based on suspicious words or keywords.
- To try out deep learning algorithms.

## CHAPTER 6

### APPENDIX

#### 6.1. REFERENCES

- [1] C.D.N.N.T.V. Xuan, "Malicious URL detection based on machine learning," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, pp. 148-153, 2020.
- [2] F. W. L. Douksieh Abdi, "Malicious URL Detection and Identification," *International Journal of Computer Science, Engineering and Information Technology*, vol. 7, no. 6, pp. 01-08, 2017.
- [3] V. V. V. N, "Malicious-URL Detection using Logistic Regression Technique," *International Journal of Engineering and Management Research*, vol. 09, no. 06, pp. 108-113, 2019.
- [4] D.B.A.N.J. Kapil, "Machine Learning-Based Malicious URL Detection," 2, vol. 8, no. 4S, pp. 22-26, 2020.
- [5] S. L. R. R. L. A. V. M. G. L. M. R. J. Jany Shabu, "Machine Learning-Based Malicious Website Detection," *Journal of Computational and Theoretical Nanoscience*, vol. 17, no. 8, pp. 3468-3472, 2020.
- [6] R.A.G.S. Naresh, "Malicious URL detection system using combined SVM and logistic regression model," *International Journal of Advanced Research in Engineering and Technology*, vol. 11, no. 4, pp. 63-73, 2020.
- [7] A. L. L. W. P. S. S. Joshi, "Using Lexical Features for Malicious URL Detection -- A Machine Learning Approach," 2019.
- [8] G.S.S.S.B.S.S.B.B.W. Palaniappan, "Malicious Domain Detection Using Machine Learning on Domain Name Features, Host-Based Features and Web-Based Features," *Procedia Computer Science*, vol. 171, no. 2019, pp. 654-661, 2020.

- [9] Apoorva Joshi, Levi Lloyd, Paul Westin and Srini Seethapathy “Using Lexical Features for Malicious URL Detection - A Machine Learning Approach” International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, 2015, pp. 195-202
- [10] <https://www.rapid7.com/fundamentals/types-of-attacks/>
- [11] <https://www.hindawi.com/journals/cmmm/2015/756842/>
- [12] <https://www.cynet.com/blog/a-guide-to-malware-detection-techniques-av-ngav-and-beyond/>
- [13] <https://www.analyticsinsight.net/security-analytics-a-tool-against-rising-cyberattacks/>
- [14] <https://cheapsslsecurity.com/blog/what-is-a-malicious-url/>
- [15] <https://pipelinesecurity.net/threat-intelligence/phishing-and-malicious-url-data/>
- [16] <https://medium.com/@ismaelbouarfa/malicious-url-detection-with-machine-learning-d57890443dec>