

MAILS SPAM DETECTION AND FILTERING USING MACHINE LEARNING

Submitted in partial fulfillment of the
requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

Kodati Naga Satya Sai Manikanta (Reg.No – 39110505)
Katakam Sidhartha (Reg.No – 39110473)



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 - 600119

APRIL - 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with 'A' grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

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 **Kodati Naga Satya Sai Manikanta(39110505)** and **Katakam Sidhartha(39110473)** who carried out the Project Phase-2 entitled "MAILS SPAM DETECTION AND FILTERING USING MACHINE LEARNING" under my supervision from Jan 2023 to April 2023.

Internal Guide

Dr P.Asha

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.



Submitted for Viva voce Examination held on 20.04.2023

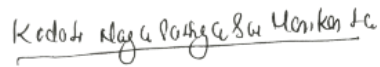
Internal Examiner

ii

External Examiner

DECLARATION

I, **Kodati NagaSatya Sai Manikanta** (Reg.No- 39110505), hereby declare that the Project Phase-2 Report entitled “**MAILS SPAM DETECTION AND FILTERING USING MACHINE LEARNING**” done by me under the guidance of **Dr.P.Asha-** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.



DATE: 20.04.2023

Kodati NagaSatya Sai Manikanta

PLACE:Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my 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. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.P.Asha**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 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

Email Spam has become a major problem nowadays, with the Rapid growth of internet users, Email spam is also increasing. People are using them for illegal and unethical conduct, phishing, and fraud. Sending malicious links through spam emails can harm our system and can also seek into your system. So, it is necessary to Identify those spam emails which are fraudulent. Input is obtained in the form of a statement. Output results are acquired instantly in real-time which predicts if the email is spam or not. The state-of-the-art methods work on the Logistic Regression algorithm which makes it less accurate and more time-consuming. This system aims to propose a Term Frequency Inverse Document Frequency (TFIDF) approach by implementing the multi-modal Machine learning algorithm. In spam mail detection, email data is collected through the dataset. To obtain accurate results, data needs to be pre-processed by removing stop words and word tokenization. Pre-processing of data is done by using the TF-IDF Vectorizer module. Machine learning automatically classifies the text in a much faster way than manual technique. Machine learning uses pre-labeled text to learn the different associations between pieces of text and their output. The advantages of the proposed system are that it could be the very first-of-its-kind, cost-efficient, and highly accurate, multi-model machine learning architecture that effectively detects spam emails. This proposed approach provides promising results which can be further improved and implemented on a commercial scale.

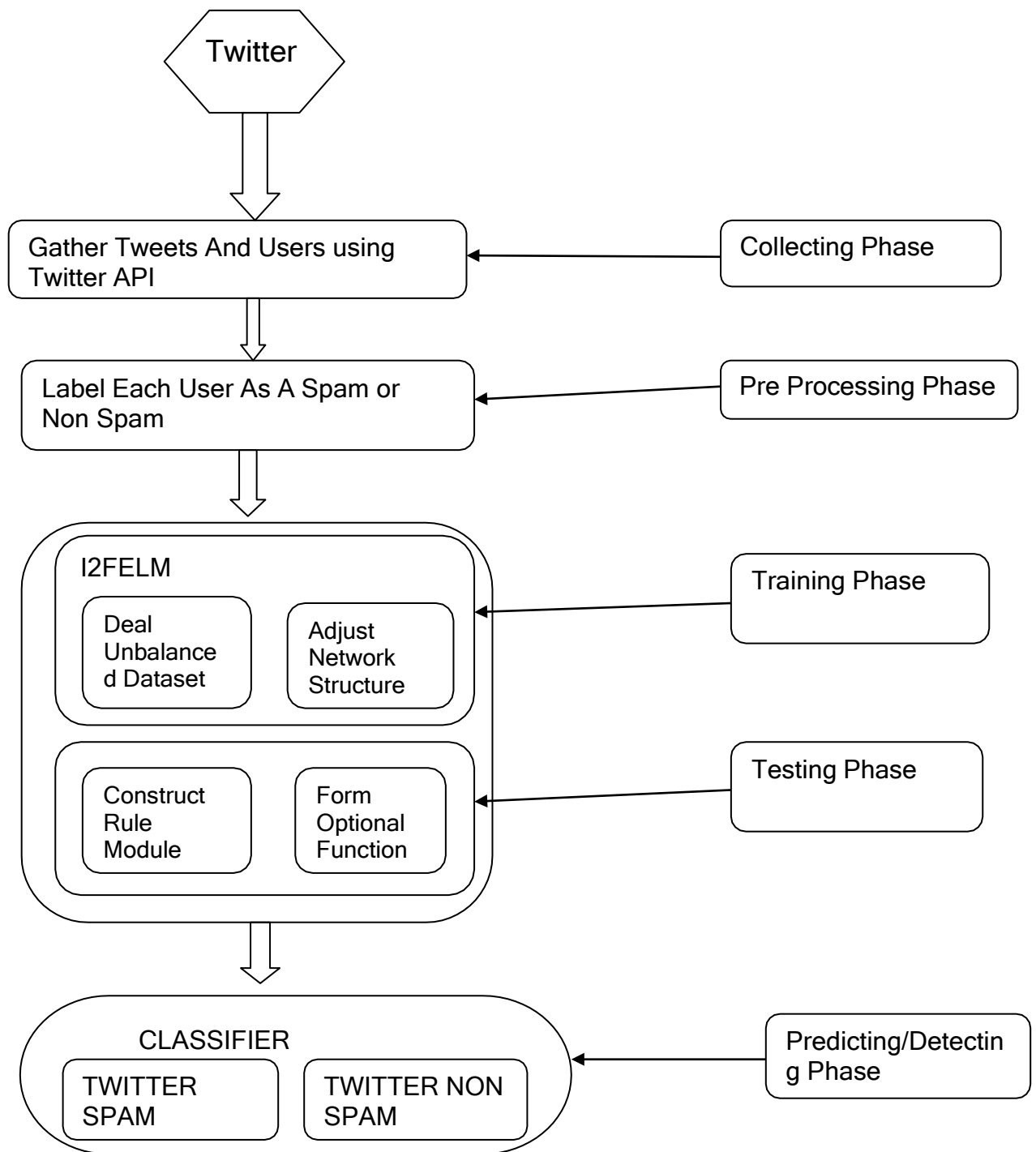
TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
2	LITERATURE SURVEY	
	2.1 Inferences From Literature Survey	2
	2.2 Open problems in Existing System	3
3	REQUIREMENTS ANALYSIS	
	3.1 Feasibility Studies/Risk Analysis of the Project	4
	3.2 Software Requirements Specification Document	5
	3.3 System Use case	6
	3.4 Data Flow Diagram	9
	3.5 Sequence Diagram	10
	3.6 Activity Diagram	12
4	DESCRIPTION OF PROPOSED SYSTEM	
	4.1 Selected Methodology or process model	13
	4.2 Architecture / Overall Design of Proposed System	13
	4.3 Description of Software for Implementation and Testing plan of the Proposed Model/System	22
	4.4 Project Management Plan	22
5	IMPLEMENTATION DETAILS	
	5.1 Importing All The Libraries	25
	5.2 Algorithms	27
	5.3 Testing	34
6	RESULTS AND DISCUSSION	36
7	CONCLUSION	37
	7.1 Conclusion	37
	7.2 Futurework	37

	REFERENCES	39
	APPENDIX	
	A. APPENDIX	
	B. SCREENSHOTS	
	C. RESEARCH PAPER	

LIST OF FIGURES

Figure No	Figure Name	Page No.
3.3	Use Case Diagram	6
3.4	Data Flow Diagram(level-0)	8
3.4	Data Flow Diagram(level-1)	9
3.5	Sequence Diagram	10
3.6	Activity Diagram	12
4.2	Architecture Diagram	13



CHAPTER 1

INTRODUCTION

Spam email is one of the most demanding and troublesome internet issues in today's world of communication and technology. It is almost impossible to think about e-mail without considering the issue of spam. Spam emails are the uninvited emails sent by some unwanted users also known as spammers. Spammers by generating spam mails are misusing this communication facility and thus affecting organizations and many email users. So, it is needed to identify those spam mails which are fraud, and this project will identify those spam by using techniques of machine learning. All emails have a common structure i.e. subject of the email and the body of the email. A typical spam mail can be classified by filtering its content. The process of spam mail detection is based on the assumption that the content of the spam mail is different from the legitimate or ham mail. My proposed model for Spam analysis and classification mainly focuses on the content of the message. The classification algorithm classifies the given email based on the content. Feature extraction and selection play a vital role in classification. Spam mails are the major issue on the internet. Spam prevents the user from creating full and sensible use of your time, storage capability, and network information measure. Spam emails not only influence the organizations financially but also exasperate the individual email user. Spammers can additionally steal sensitive information from our devices like files, and contact. Even though we have the latest technology, it is challenging to detect spam emails.

The e-mail suppliers have designed varied mechanisms to be used in email anti-spam filters to reduce the risks posed by phishing. The e-mail suppliers have designed varied mechanisms to be used in email anti-spam filters to reduce the risks posed by phishing, email-borne malware, and ransomware to email users. In order to deal with spam emails, we need to build a robust real-time email spam classifier that can efficiently and correctly flag the incoming mail spam, if it is a spam message or looks like a spam message.

CHAPTER 2

LITERATURE SURVEY

2.1 INFERENCES FROM LITREATURE SURVEY

In this project, a Spam Mail Detection system is proposed that will classify the given email as spam or not spam. The proposed system leverages a multi-modal machine learning approach. It proves to be more accurate due to the use of multiple classifiers for class prediction. The given model for spam detection is proficient in filtering emails according to the content of the email and not according to the domain names or any other criteria. The proposed model works well against train data and test data further this model will provide better results for real-time data. A comparative analysis of the proposed model with state-of-the-art methods assessed that our model was able to outperform the other models in terms of efficiency, accuracy, and processing time. It is a fast, cost-efficient, quick, and highly accurate method to identify patterns. There is a great scope for this email spam classifier, as the private companies run their own email servers and want them to be more secure because of the confidential data. In such cases, email spam classifier solutions can be provided to such companies.

Saleh et al present a survey on intelligent spam email detection. (ey discuss various security risks of emails, especially spam emails,scope of spam analysis, and different machine learning and nonmachine learning techniques for spam detection and filtering. (ey conclude that there is high adoption of supervised learning [18] algorithms for email spam detection. (ey state that the high usage of supervised learning is the accuracy and consistency of supervised techniques.(ey also discussed multi-algorithm frameworks and found that multialgorithm frameworks are more efficient than a single algorithm. (ey found that nearly all research work that uses the content of emails for the identification spam, particularly phishing emails,depends on word-based classification.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

Automatic email filtering was once the most effective method of detecting spam but nowadays spammers can easily bypass all these spam filtering applications easily. The existing systems for spam detection use logistic regression algorithms to detect spam emails. The use of this algorithm and a smaller dataset account for the low accuracy of the traditional models. The existing systems are simple and effective but are extremely vulnerable to impact. It is also highly vulnerable to misclassification, which may lead to inaccurate and inefficient results. The discussed limitations have been overcome to enhance the performance of prediction models successfully in the presented application.

CHAPTER 3 REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

The feasibility of the project is server performance increase in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Operational feasibility

3.1.1. Economical feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.1.2. TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high

3.1.3. OPERATIONAL FEASIBILITY

The aspect of the study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must

accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.2. SOFTWARE REQUIREMENTS

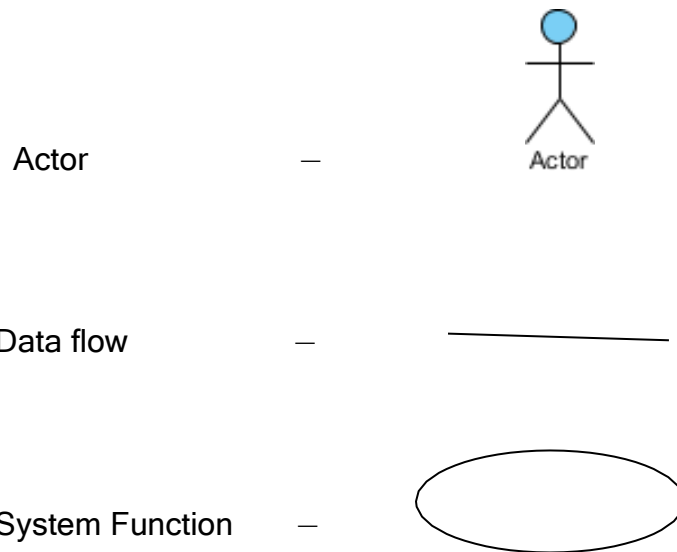
Operating system : Windows10(64 bit)
Software : Python
Tools : Anaconda(Jupyter Note Book IDE)

3.3. SYSTEM USE CASE

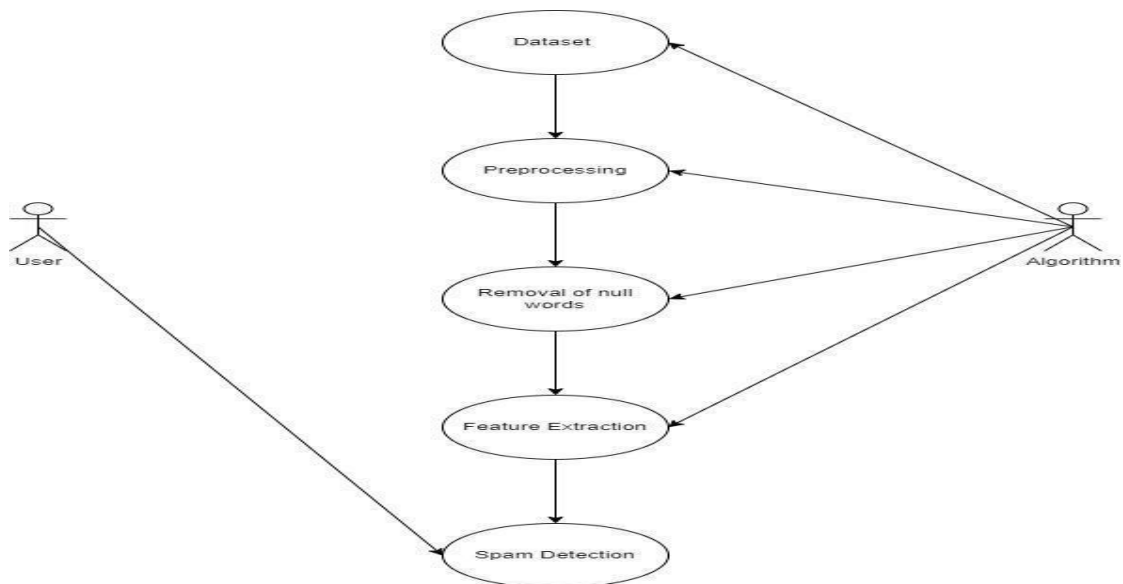
A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system. Use case diagrams can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

➤ Symbols Used



Use Case Diagram



3.3. USE CASE DIAGRAM

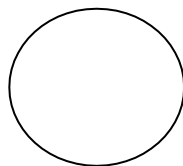
3.4. DATA FLOW DIAGRAM

A Data-Flow Diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart. There are several notations for displaying data-flow diagrams. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The data-flow diagram is part of the structured-analysis modeling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan. Data-flow diagrams can be regarded as inverted Petri nets, because places in such networks correspond to the semantics of data memories. DFD consists of processes, flows, warehouses, and terminators.

Data Flow Diagram Symbols

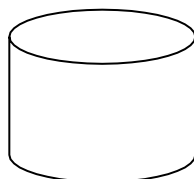
➤ Process

A process transforms incoming data flow into outgoing data flow.



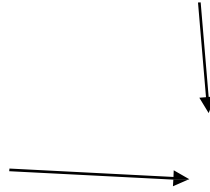
➤ Data Store

Data stores are repositories of data in the system. Sometimes it's also referred to as files.



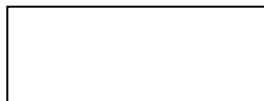
➤ Data Flow

Data flows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.



➤ External Entity

External entities are objects outside the system, with which the system communicates. These are sources and destinations of the system's inputs and outputs.



DATA FLOW DIAGRAM

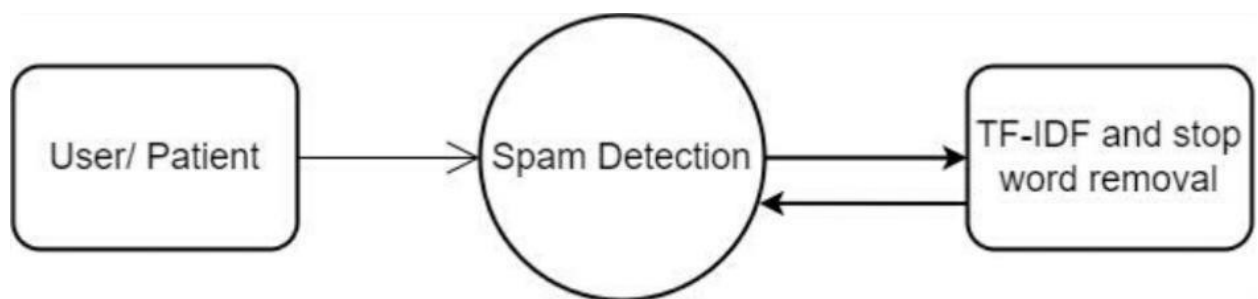


Fig 3.4 Data flow diagram (level-0)

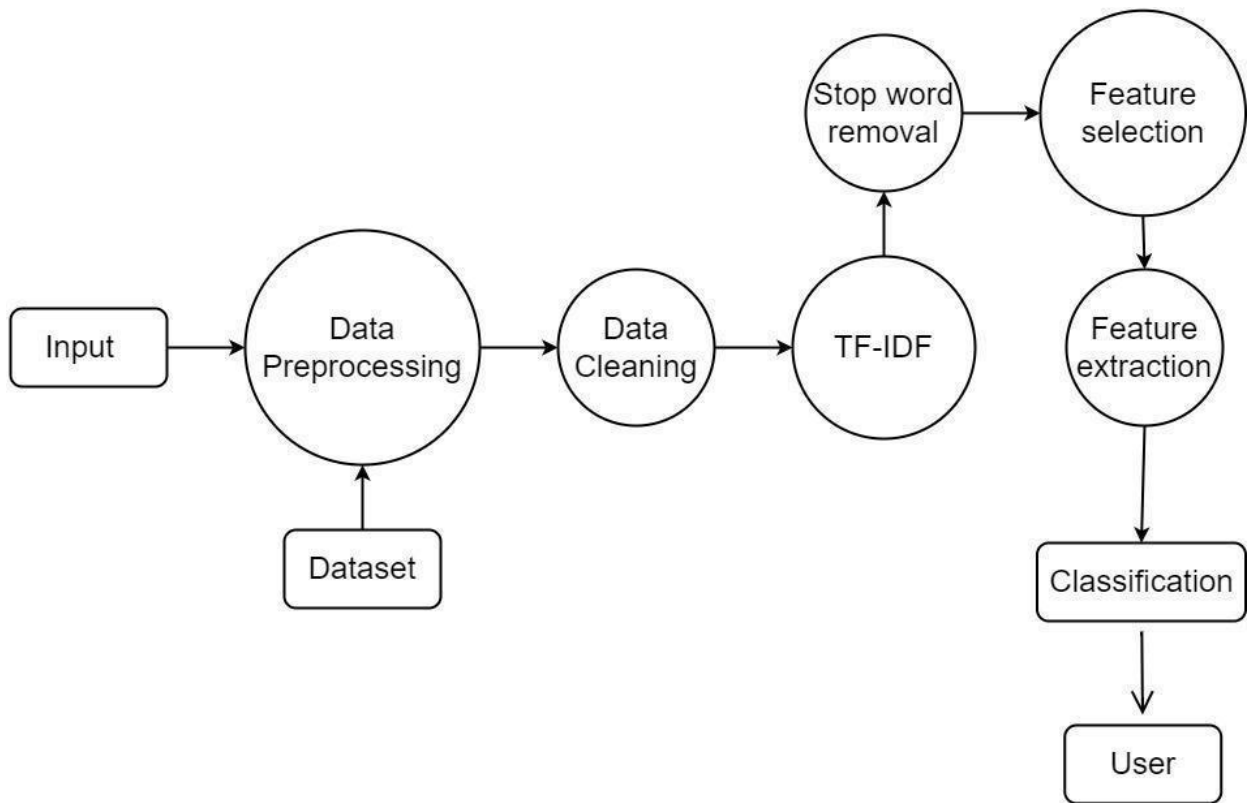


Fig 3.4. Data flow diagram (level-1)

3.5. Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when. Sequence Diagrams captures:

- the interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

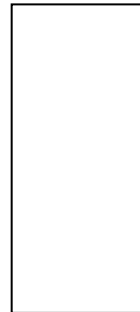
Object symbol

-



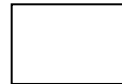
Activation Box

-

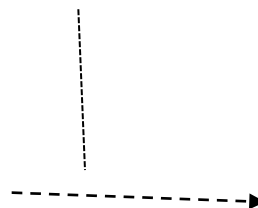


Lifeline Symbol

-

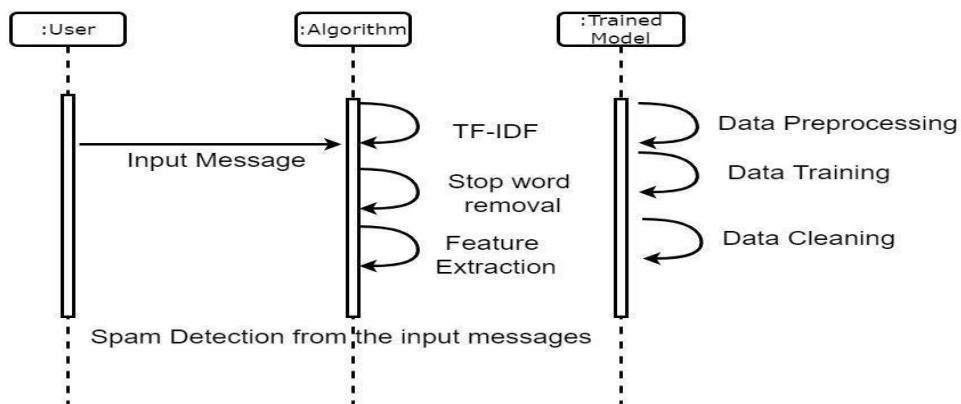
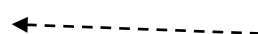


Asynchronous create message symbol -



Reply message symbol

-



3.5. Sequence Diagram

3.6. Activity Diagram

The basic purpose of activity diagrams is similar to the other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show

the message flow from one object to another but the activity diagram is used to show the message flow from one activity to another. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single. The purpose of an activity diagram can be described as –

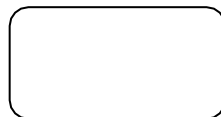
- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system

Symbols Used

Initial State



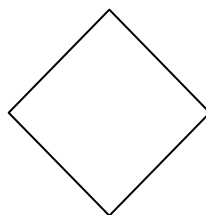
Activity State



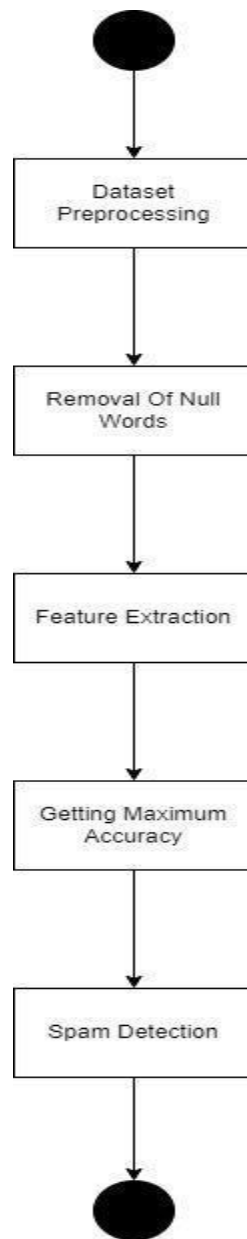
Action Flow



Decision Node



➤ **Activity Diagram**



3.6.Activity Diagram

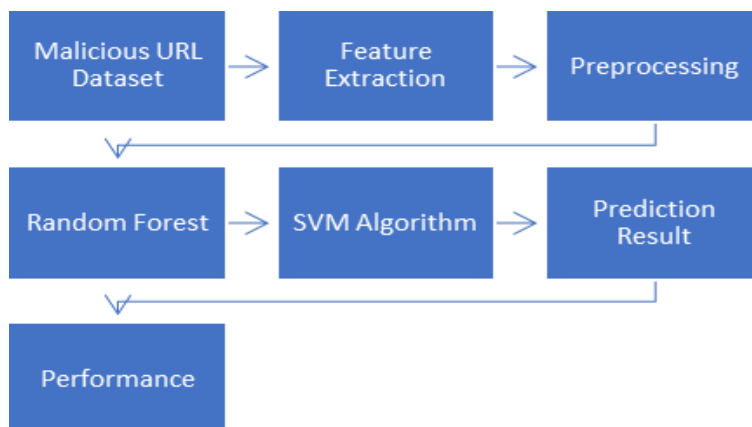
Chapter-4

DESCRIPTION OF PROPOSED SYSTEM

4.1. Selected Methodology or Process model

In this project, a Spam Mail Detection system is proposed that will classify the given email as spam or not spam. The proposed system leverages a multi-modal machine learning approach. It proves to be more accurate due to the use of multiple classifiers for class prediction. The given model for spam detection is proficient in filtering emails according to the content of the email and not according to the domain names or any other criteria. The proposed model works well against train data and test data further this model will provide better results for real-time data. A comparative analysis of the proposed model with state-of-the-art methods assessed that our model was able to outperform the other models in terms of efficiency, accuracy, and processing time. It is a fast, cost-efficient, quick, and highly accurate method to identify patterns. There is a great scope for this email spam classifier, as the private companies run their own email servers and want them to be more secure because of the confidential data. In such cases, email spam classifier solutions can be provided to such companies.

4.2. Architecture Diagram



4.2.

ARCHITECTURE DIAGRAM

4.3. Description of Software for Implementation and Testing plan of the Proposed Model/System

PLAN OF THE PROPOSED MODEL/SYSTEM

- Operating System: Windows 10 (64 bit)
- Software: Python
- Tools: Anaconda (Jupyter Note Book IDE)
- Hard Disk: 500GB and Above
- RAM: 4GB and Above
- Processor: I3 and Above

4.3.1. Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

What Python can do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

- Python can be used for rapid prototyping, or for production-ready software development.

4.3.2. SOFTWARE SPECIFICATIONS - ANACONDA

Anaconda is an open-source package manager for Python and R. It is the most popular platform among data science professionals for running Python and R implementations. There are over 300 libraries in data science, so having a robust distribution system for them is a must for any professional in this field. Anaconda simplifies package deployment and management. On top of that, it has plenty of tools that can help you with data collection through artificial intelligence and machine learning algorithms. With Anaconda, you can easily set up, manage, and share Conda environments. Moreover, you can deploy any required project with a few clicks when you're using Anaconda. There are many advantages to using Anaconda and the following are the most prominent ones among them: Anaconda is free and open-source. This means you can use it without spending any money. In the data science sector, Anaconda is an industry staple. It is open-source too, which has made it widely popular. If you want to become a data science professional, you must know how to use Anaconda for Python because every recruiter expects you to have this skill. It is a must-have for data science.

It has more than 1500 Python and R data science packages, so you don't face any compatibility issues while collaborating with others. For example, suppose your colleague sends you a project which requires packages called A and B but you only have package A. Without having package B, you wouldn't be able to run the project. Anaconda mitigates the chances of such errors. You can easily collaborate on projects without worrying about any compatibility issues. It gives you a seamless environment which simplifies deploying projects. You can deploy any project with just a few clicks and commands while managing the rest. Anaconda has a thriving community of data scientists and machine learning professionals who use it regularly. If you encounter an issue, chances are, the community has already answered the same. On the other hand, you can also ask people in the community about the issues you face there, it's a very helpful community ready to help new learners. With Anaconda, you can easily create and train machine learning and deep learning models as it works well with popular tools including TensorFlow, Scikit-Learn, and

Theano. You can create visualizations by using Bokeh, Holoviews, Matplotlib, and Datashader while using Anaconda.

How to Use Anaconda for Python

Now that we have discussed all the basics in our Python Anaconda tutorial, let's discuss some fundamental commands you can use to start using this package manager.

Listing All Environments

To begin using Anaconda, you'd need to see how many Conda environments are present in your machine.

```
conda env list
```

It will list all the available Conda environments in your machine.

Creating a New Environment

You can create a new Conda environment by going to the required directory and use this command:

```
conda create -n <your_environment_name>
```

You can replace `<your_environment_name>` with the name of your environment. After entering this command, conda will ask you if you want to proceed to which you should reply with y:

```
proceed ([y])/n)?
```

On the other hand, if you want to create an environment with a particular version of Python, you should use the following command:

```
conda create -n <your_environment_name> python=3.6
```

Similarly, if you want to create an environment with a particular package, you can use the following command:

```
conda create -n <your_environment_name> pack_name
```

Here, you can replace `pack_name` with the name of the package you want to use.

If you have a `.yaml` file, you can use the following command to create a new Conda

environment based on that file:

```
conda env create -n <your_environment_name> -f <file_name>.yml
```

We have also discussed how you can export an existing Conda environment to a .yml file later in this article.

Activating an Environment

You can activate a Conda environment by using the following command:

```
conda activate <environment_name>
```

You should activate the environment before you start working on the same. Also, replace the term <environment_name> with the environment name you want to activate. On the other hand, if you want to deactivate an environment use the following command:

```
conda deactivate
```

Installing Packages in an Environment

Now that you have an activated environment, you can install packages into it by using the following command:

```
conda install <pack_name>
```

Replace the term <pack_name> with the name of the package you want to install in your Conda environment while using this command.

Updating Packages in an Environment

If you want to update the packages present in a particular Conda environment, you should use the following command:

```
conda update
```

The above command will update all the packages present in the environment. However, if you want to update a package to a certain version, you will need to use

the following command:

```
conda install <package_name>=<version>
```

Exporting an Environment Configuration

Suppose you want to share your project with someone else (colleague, friend, etc.). While you can share the directory on Github, it would have many Python packages, making the transfer process very challenging. Instead of that, you can create an environment configuration .yml file and share it with that person. Now, they can create an environment like your one by using the .yml file.

For exporting the environment to the .yml file, you'll first have to activate the same and run the following command:

```
conda env export ><file_name>.yml
```

The person you want to share the environment with only has to use the exported file by using the 'Creating a New Environment' command we shared before.

Removing a Package from an Environment

If you want to uninstall a package from a specific Conda environment, use the following command:

```
conda remove -n <env_name><package_name>
```

On the other hand, if you want to uninstall a package from an activated environment, you'd have to use the following command:

```
conda remove <package_name>
```

Deleting an Environment

Sometimes, you don't need to add a new environment but remove one. In such cases, you must know how to delete a Conda environment, which you can do so by using the following command:

```
conda env remove -name <env_name>
```

The above command would delete the Conda environment right away.

4.3.2.1. FRONT END SPECIFICATIONS

Front-end web development is the process of transforming the data to a graphical interface, through the usage of CSS, HTML, and JavaScript so that the users can observe and network with that data. The front end portion is built by using some languages which are discussed below: HTML: HTML stands for **Hypertext Markup Language**. It is used to design the front-end portion of web pages using a markup language. HTML is the combination of Hypertext and Markup language. The part of a website that the user interacts with directly is termed the front end. It is also referred to as the 'client side' of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and a navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e. it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

HTML: HTML stands for Hypertext Markup Language. It is used to design the front-end portion of web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup language is used to define the text documentation within the tag which defines the structure of web pages.

CSS: Cascading Style Sheets fondly referred to as CSS is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.

Some other libraries and frameworks are Semantic-UI, Foundation, Materialize, Backbone.js, Ember.js, etc.

4.3.2.2.BACKEND SPECIFICATION

Advantages of Python

1. Easy to Read, Learn and Write

Python is a high-level programming language that has English-like syntax. This makes it easier to read and understand the code.

Python is really easy to pick up and learn, that is why a lot of people recommend Python to beginners. You need less lines of code to perform the same task as compared to other major languages like C/C++ and Java.

2. Improved Productivity

Python is a very productive language. Due to the simplicity of Python, developers can focus on solving the problem. They don't need to spend too much time in understanding the syntax or behavior of the programming language. You write less code and get more things done.

3. Interpreted Language

Python is an interpreted language which means that Python directly executes the code line by line. In case of any error, it stops further execution and reports back the error which has occurred.

Python shows only one error even if the program has multiple errors. This makes debugging easier.

4. Dynamically Typed

Python doesn't know the type of variable until we run the code. It automatically assigns the data type during execution. The programmer doesn't need to worry about declaring variables and their data types.

5. Free and Open-Source

Python comes under the OSI approved open-source license. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behavior and use their version for development.

6. Vast Libraries Support

The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don't have to depend on external libraries.

But even if you do, a Python package manager (pip) makes things easier to import other great packages from the Python package index (PyPi). It consists of over 200,000 packages.

7. Portability

In many languages like C/C++, you need to change your code to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere.

However, you should be careful not to include any system-dependent features.

4.3.3. TESTING PLAN

The major objectives of Software testing are as follows:

- 1) Finding defects which may get created by the programmer while developing the software.
- 2) Gaining confidence in and providing information about the level of quality.
- 3) To prevent defects.
- 4) To make sure that the end result meets the business and user requirements.

4.3.3.1. TEST PROCESS

We can divide the activities within the fundamental test process into the following basic steps:

Planning and Control.

Analysis and Design.

Implementation and Execution.

Evaluating exit criteria and Reporting.

Test Closure activities

4.4. PROJECT MANAGEMENT PLAN

- **Data Collection and Preprocessing** : The datasets are text data, pre-processing is employed for machine learning. We eliminate stop words and list the tokens with comments using the Count Vectorizer function of the Python Psychic Run library
- **Feature Extraction** : Feature selection is a process before classification class. The suitable features will be identified based on the dataset. The feature set is classified into two categories namely user-based features and content-based features.
- **User-Based Features** : User-based features are used to describe the behavior of users on Twitter. These features are based on user relationships and properties of user accounts in the Twitter dataset.
- **Content-Based Features** : These features are related to tweets posted by users. Generally, normal users can't post duplicate content but spammers post a lot of duplicate tweets. Content-based features are based on messages that users write.
- **Classification** : SVM is successfully suitable for differentiating positive and negative problems such as spam. SVM is a supervised learning model that analyzes data used for classification and regression. AdaBoost is short for adaptive boosting and can be used in conjunction with other algorithms.

CHAPTER 5.

IMPLEMENTATION DETAILS

Install Required Packages.

- To complete the project, you need the following packages and libraries.
 1. Anaconda Navigator
 2. Jupyter
 3. Numpy
 4. Pandas
 5. Matplotlib
 6. Scikit-Learn

Install Anaconda software

To install Anaconda on your local system, go through the below links according to your system requirements. After Anaconda is installed, run the .exe folder.

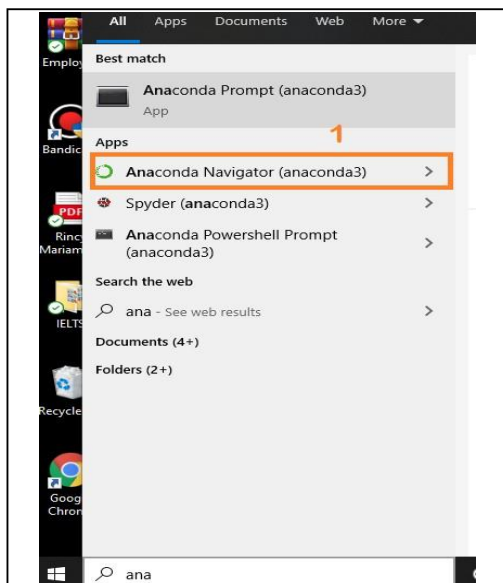
For Windows- [Link](#)

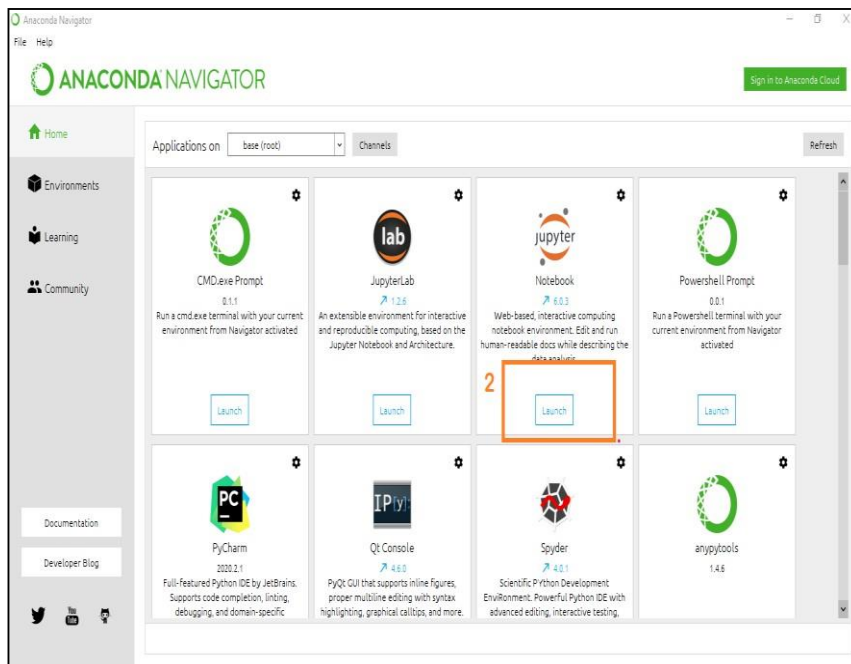
For MacOS- [Link](#)

For Linux- [Link](#)

Install Required Libraries

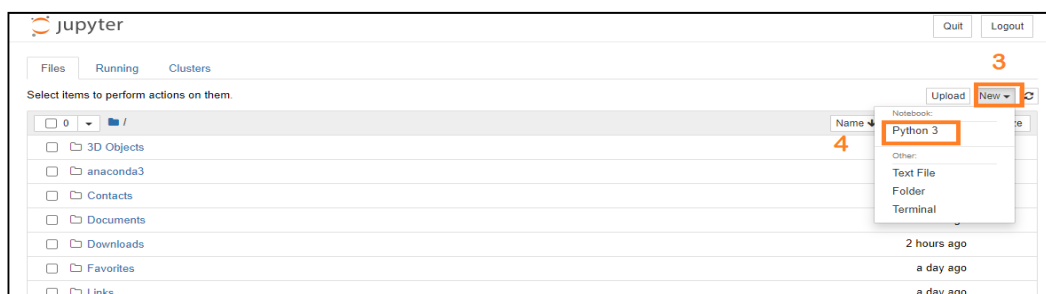
Search Anaconda Navigator and open a Jupyter notebook





Installation of Numpy Library :

Using Anaconda



Navigator: conda install numpy

OR

Using command prompt: pip install numpy.

Installation of Pandas Library :

Using Anaconda Navigator: conda install pandas

OR

Using command prompt: pip install pandas.

Installation of Matplotlib Library :

Using Anaconda Navigator: conda install matplotlib

OR

Using command prompt: pip install matplotlib

Installation of Scikit-Learn Library :

Using Anaconda Navigator: conda install -c conda-forge scikit-learn

OR

Using command prompt: pip install -U scikit-learn

5.1. Importing All The Libraries

- Import the required libraries for the model to run.

First step is usually importing the libraries that will be needed in the program.

```
In [1]: import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPooling1D, Dropout
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Numpy- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. Pandas objects are very much dependent on NumPy objects.

Pandas- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Step 2: Loading And PreProcessing The Data

```
In [2]: df = pd.read_csv('spam.csv')
df = df[['Category', 'Message']]
df['Category'] = df['Category'].map({'spam': 1, 'ham': 0})
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['Message'])
y = df['Category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Training And Testing The Dataset:

```
In [3]: # Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)
rfc.fit(X_train, y_train)
rfc_y_pred = rfc.predict(X_test)
rfc_accuracy = accuracy_score(y_test, rfc_y_pred)
rfc_precision = precision_score(y_test, rfc_y_pred)
rfc_recall = recall_score(y_test, rfc_y_pred)
rfc_f1_score = f1_score(y_test, rfc_y_pred)

# Linear Regression Classifier
lrc = LogisticRegression(random_state=42)
lrc.fit(X_train, y_train)
lrc_y_pred = lrc.predict(X_test)
lrc_accuracy = accuracy_score(y_test, lrc_y_pred)
lrc_precision = precision_score(y_test, lrc_y_pred)
lrc_recall = recall_score(y_test, lrc_y_pred)
lrc_f1_score = f1_score(y_test, lrc_y_pred)

# Support Vector Classifier
svc = SVC(kernel='linear', random_state=42)
svc.fit(X_train, y_train)
svc_y_pred = svc.predict(X_test)
svc_accuracy = accuracy_score(y_test, svc_y_pred)
svc_precision = precision_score(y_test, svc_y_pred)
svc_recall = recall_score(y_test, svc_y_pred)
svc_f1_score = f1_score(y_test, svc_y_pred)

# Artificial Neural Network
ann = MLPClassifier(hidden_layer_sizes=(16, 8), max_iter=500, random_state=42)
ann.fit(X_train, y_train)
ann_y_pred = ann.predict(X_test)
ann_accuracy = accuracy_score(y_test, ann_y_pred)
ann_precision = precision_score(y_test, ann_y_pred)
ann_recall = recall_score(y_test, ann_y_pred)
ann_f1_score = f1_score(y_test, ann_y_pred)

# Convolutional Neural Network
max_features = 20000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(df['Message'].values)
X = tokenizer.texts_to_sequences(df['Message'].values)
X = pad_sequences(X)
y = df['Category'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
cnn = Sequential()
cnn.add(Embedding(max_features, 128))
cnn.add(Conv1D(64, 5, activation='relu'))
cnn.add(MaxPooling1D(pool_size=4))
cnn.add(Dropout(0.2))
cnn.add(LSTM(128))
cnn.add(Dense(1, activation='sigmoid'))
cnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

cnn.fit(X_train, y_train, batch_size=64, epochs=10, validation_data=(X_test, y_test))
cnn_y_pred = np.argmax(cnn.predict(X_test), axis=-1)
cnn_accuracy = accuracy_score(y_test, cnn_y_pred)
cnn_precision = precision_score(y_test, cnn_y_pred)
cnn_recall = recall_score(y_test, cnn_y_pred)
cnn_f1_score = f1_score(y_test, cnn_y_pred)

print("Random Forest Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1 Score = %.2f" % (rfc_accuracy, rfc_precision, rfc_recall, rfc_f1_score))
print("Linear Regression Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1 Score = %.2f" % (lrc_accuracy, lrc_precision, lrc_recall, lrc_f1_score))
print("Support Vector Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1 Score = %.2f" % (svc_accuracy, svc_precision, svc_recall, svc_f1_score))
print("Artificial Neural Network: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1 Score = %.2f" % (ann_accuracy, ann_precision, ann_recall, ann_f1_score))
print("Convolutional Neural Network: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1 Score = %.2f" % (cnn_accuracy, cnn_precision, cnn_recall, cnn_f1_score))
```

```
In [4]: import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Random Forest Classifier
rfc_cm = confusion_matrix(y_test, rfc_y_pred)
sns.heatmap(rfc_cm, annot=True, cmap='Blues', fmt='g')
plt.title('Random Forest Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Linear Regression Classifier
lrc_cm = confusion_matrix(y_test, lrc_y_pred)
sns.heatmap(lrc_cm, annot=True, cmap='Blues', fmt='g')
plt.title('Linear Regression Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Support Vector Classifier
svc_cm = confusion_matrix(y_test, svc_y_pred)
sns.heatmap(svc_cm, annot=True, cmap='Blues', fmt='g')
plt.title('Support Vector Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```

# Artificial Neural Network
ann_cm = confusion_matrix(y_test, ann_y_pred)
sns.heatmap(ann_cm, annot=True, cmap='Blues', fmt='g')
plt.title('Artificial Neural Network')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Convolutional Neural Network
cnn_cm = confusion_matrix(y_test, cnn_y_pred)
sns.heatmap(cnn_cm, annot=True, cmap='Blues', fmt='g')
plt.title('Convolutional Neural Network')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

```

Random Forest Classifier

5.2. ALGORITHMS

- **Random Forest:** A machine learning classification technique that creates a number of interdependent decision trees. A forecast is made by each tree in the RF, and the one with the highest votes is used.
- **Logistic Regression :** Classification model in machine learning that uses the connection between independent variables to make predictions. Its primary use is in solving issues of categorical choice.
- **SVM :** A scatter plot with n-dimensional pieces of data using a supervised ML algorithm, where n is the number of characteristics. The next step is to construct a hyperplane that accurately separates the two groups.

5.3. TESTING

5.3.1. OBJECTIVES

The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.

5.3.2 TEST PROCESS

We can divide the activities within the fundamental test process into the following basic steps:

- Planning and Control.
- Analysis and Design.
- Implementation and Execution.
- Evaluating exit criteria and Reporting.
- Test Closure activities.

5.3.3 TEST CASES

A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly. The purpose of a test case is to determine if different features within a system are performing as expected and to confirm that the system satisfies all related standards, guidelines and customer requirements. The process of writing a test case can also help reveal errors or defects within the system.

Test cases are typically written by members of the quality assurance (QA) team or the testing team and can be used as step-by-step instructions for each system test. Testing begins once the development team has finished a system feature or set of features. A sequence or collection of test cases is called a test suite. A test case document includes test steps, test data, preconditions and the postconditions that verify requirements.

5.3.4 TESTING TYPES

There are many different types of testing methods or techniques used as part of the software testing methodology. Some of the important testing methodologies are:

System Testing

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during this phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design before coding commences, otherwise the cost of fixing the faults will be considerably

higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

Unit Testing

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

Functional Tests

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

Performance Test

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

Stress Test

Stress Test is a test designed to intentionally break the unit. A Great deal

can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

Structured Test

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then test. This approach evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we

checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated as they complete unit testing. Testing is completed when the last module is integrated and tested.

5.3.5 VERIFICATION AND VALIDATION

Testing

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet-undiscovered error. A successful test is one that uncovers an as-yet-undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consisting of several key activities and steps for running a program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that has a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

White Box Testing

This testing is also called Glass box testing. In this testing, by knowing the specific functions that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclomatic complexity
- Deriving test cases
- Graph matrices Control

Black Box Testing

In this testing by knowing the internal operation of a product, a test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

Software Testing Strategies:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.

- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

Integration Testing:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impressions may be magnified to unacceptable levels; global data structures can present problems.

Program Testing:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax errors. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-of-range items and invalid combinations. Since the compiler will not deduct logical errors, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program. The purpose of the condition test is to deduct not only errors in the condition of a program but also other errors in the program.

Security Testing:

Security testing attempts to verify the protection mechanisms built into a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of an individual who desires to penetrate the system.

Validation Testing

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in a manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirements. After the validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics conform to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviations or errors discovered at this step in this project are corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

User Acceptance Testing

User acceptance of the system is a key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regards to the following points.

- Input screen design.
- Output screen design.

CHAPTER 6.

RESULTS AND DISCUSSION

The results of the best features acquired from the attribute selection approach are shown in Table I below, along with the data achieved by utilising all the characteristics in the discrete time series (with no aspects selection techniques performed) (correlation analysis).

Table I Results when using all features and best features for the second dataset

Using all features				
Algorith m	Accura cy	Precisi on	Reca ll	F- score
RF	92.05	93.10	90.81	91.94
LR	83.81	84.39	82.95	83.66
SVM	86.71	90.23	82.29	86.08
ANN	88.88	89.69	87.40	88.53
CNN	90.26	93.15	88.21	90.61
Using Best features				
RF	92.83	93.71	90.67	92.65
LR	84.29	84.60	83.66	84.18
SVM	86.84	90.26	82.57	86.25
ANN	89.98	92.25	87.38	89.98
CNN	90.46	90.46	88.30	90.24

Using the best-selected attributes, RF obtained a 92.83% accuracy, with CNN coming in as a distant second at a 90% accuracy. In addition, RF achieved the greatest recall and fscore, and had the best accuracy (93.71). However, unlike the first dataset, the models' performance on this one did not improve much when features were selected. Again, we see that CNN's performance is inferior than RF's while being considerably closer to it in all sets of findings. Since the same CNN architecture was employed for both experimental sets, the results might be attributed to either the size of the dataset or the design.

Moreover, we can see from the aforementioned statistics that the features selection strategy resulted in an increase in classification accuracy across the board for both datasets. This means that feature engineering methods are essential for enhancing classifier efficacy.

Since most reputable websites are approved, they show up in the Google index, and they've been there for a while, these may be used as indicators of whether or not the site is a phishing attempt. URL of Anchor and Request URL are two content-based features that look for telltale signs of malicious links by analysing HTML meta tags and inline text, respectively. This in return is critical in determining whether or not the site is a phishing attempt. Also, in the features selection step, these 4 characteristics produced a high significance level for the first dataset, indicating that they indeed play a role in identifying phishing websites.

Finally, the results of an experiment where ML algorithms are implemented to the other discrete time series using only attributes present in both datasets. However, when compared to the results of the past versions in table I, which can be seen by looking at the numbers with table II, there was no substantial improvement. Perhaps this is due to the fact that contextual factors & link ids, two features that were removed from the 2nd sample due to poor correlation values, are responsible for this. Therefore, including them in the models would have been overkill.

TABLE II Results on using available features of dataset1 in dataset 2

Using all Features				
Algorithm	Accuracy	Precision	Recall	F-score
RF	91.98	91.42	92.59	92.01
LR	84.29	82.84	86.37	84.57
SVM	87.41	89.15	85.09	87.08
ANN	89.12	88.05	90.92	89.46
CNN	90.31	90.11	92.14	90.25

CHAPTER 7.

CONCLUSION

7.1. CONCLUSION AND FUTURE WORK

An increasing number of phishing websites and internet users have necessitated innovative methods for detecting these malicious domains. Numerous studies have been conducted on the topic of distinguishing phishing websites from legal ones by using vast datasets and clever approaches. In this post, we've focused mostly on features development as we worked to create a number of modeling software and identify the aspects that influence these systems' ability to identify malicious urls.

We deployed RF, and SVM for feature selection more to enhance the classification accuracy of the deployed models.

Overall, RF performed the best in both circumstances. RF and SVM achieved perfect accuracy on the first dataset owing to the original characteristics study as well as the associated aspects study, but the second dataset did not fair as well. Because it was found that the initial dataset had more associated traits that were lacking in the second. For this reason, we conducted an extensive series of studies to learn what factors influence the recognition of the intelligent models while making the determination of whether or not a site is phishing.

We recommend further research into other datasets and the use of deeper DL models to improve and compare performance, as well as the identification of additional essential factors in detecting phishing sites.

REFERENCES

Abhila B; Delphin Periyannayagi M; Koushika M, 2021, "Spam Detection System Using Supervised ML," International Conference on System, Computation, Automation and Networking (ICSCAN)

Nikhil Govil, Kunal Agarwal, 2020, "A Machine Learning-based Spam Detection Mechanism", 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)

Nikhil Kumar; Sanket Sonowal; Nishant, 2020, "Email Spam Detection Using Machine Learning Algorithms," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)

Simran Gibson; Biju Issac, 2020, "Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms," IEEE Access, Volume: 8

Sridevi Gadde; A. Lakshmanarao; S. Satyanarayana, 2021, "SMS Spam Detection using Machine Learning and Deep Learning Techniques," 7th International Conference on Advanced Computing and Communication Systems (ICACCS)

Mansoor RAZA; Nathali Dilshani Jayasinghe; Muhana Magboul Ali Muslam, 2021 "A comprehensive review on Email Spam Classification Using Machine Learning Algorithms", IEEE

Jaeun Choi, Chunmi Jeon, 2021 "Cost-based Heterogeneous Learning Framework

for Real Time Spam Detection in social media with Expert Decisions” IEEE

Ersin Enes Eryilmaz; Durmuş Ozkan Şahin; Erdal Kılıç, 2020 “Machine Learning Based Spam E-mail Detection System For Turkish, IEEE

Hamdullah Karamollaoglu; İbrahim Alper Dogru; Murat Dorterler, 2018 “Detection of spam E-mails with Machine Learning Methods”, IEEE

S. Nandhini; Jeen Marseline K.S, 2020 “Performance Evaluation of Machine Learning Algorithms For Email Spam Detection”, IEEE

Source Code

```
import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neural_network import MLPClassifier

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPooling1D, Dropout

from keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences


df = pd.read_csv('spam.csv')

df = df[['Category', 'Message']]

df['Category'] = df['Category'].map({'spam': 1, 'ham': 0})

vectorizer = CountVectorizer(stop_words='english')

X = vectorizer.fit_transform(df['Message'])

y = df['Category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Random Forest Classifier
```

```
rfc = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rfc.fit(X_train, y_train)
```

```
rfc_y_pred = rfc.predict(X_test)
```

```
rfc_accuracy = accuracy_score(y_test, rfc_y_pred)
```

```
rfc_precision = precision_score(y_test, rfc_y_pred)
```

```
rfc_recall = recall_score(y_test, rfc_y_pred)
```

```
rfc_f1_score = f1_score(y_test, rfc_y_pred)
```

```
# Linear Regression Classifier
```

```
lrc = LogisticRegression(random_state=42)
```

```
lrc.fit(X_train, y_train)
```

```
lrc_y_pred = lrc.predict(X_test)
```

```
lrc_accuracy = accuracy_score(y_test, lrc_y_pred)
```

```
lrc_precision = precision_score(y_test, lrc_y_pred)
```

```
lrc_recall = recall_score(y_test, lrc_y_pred)
```

```
lrc_f1_score = f1_score(y_test, lrc_y_pred)
```

```
# Support Vector Classifier
```

```
svc = SVC(kernel='linear', random_state=42)
```

```
svc.fit(X_train, y_train)
```

```
svc_y_pred = svc.predict(X_test)
```

```
svc_accuracy = accuracy_score(y_test, svc_y_pred)
```

```
svc_precision = precision_score(y_test, svc_y_pred)

svc_recall = recall_score(y_test, svc_y_pred)

svc_f1_score = f1_score(y_test, svc_y_pred)
```

```
# Artificial Neural Network
```

```
ann = MLPClassifier(hidden_layer_sizes=(16, 8), max_iter=500, random_state=42)

ann.fit(X_train, y_train)

ann_y_pred = ann.predict(X_test)

ann_accuracy = accuracy_score(y_test, ann_y_pred)

ann_precision = precision_score(y_test, ann_y_pred)

ann_recall = recall_score(y_test, ann_y_pred)

ann_f1_score = f1_score(y_test, ann_y_pred)
```

```
# Convolutional Neural Network
```

```
max_features = 20000

tokenizer = Tokenizer(num_words=max_features, split=' ')

tokenizer.fit_on_texts(df['Message'].values)

X = tokenizer.texts_to_sequences(df['Message'].values)

X = pad_sequences(X)

y = df['Category'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

cnn = Sequential()

cnn.add(Embedding(max_features, 128))
```

```

cnn.add(Conv1D(64, 5, activation='relu'))

cnn.add(MaxPooling1D(pool_size=4))

cnn.add(Dropout(0.2))

cnn.add(LSTM(128))

cnn.add(Dense(1, activation='sigmoid'))

cnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


cnn.fit(X_train, y_train, batch_size=64, epochs=10, validation_data=(X_test, y_test))

cnn_y_pred = np.argmax(cnn.predict(X_test), axis=-1)

cnn_accuracy = accuracy_score(y_test, cnn_y_pred)

cnn_precision = precision_score(y_test, cnn_y_pred)

cnn_recall = recall_score(y_test, cnn_y_pred)

cnn_f1_score = f1_score(y_test, cnn_y_pred)


print("Random Forest Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1
Score = %.2f" % (rfc_accuracy, rfc_precision, rfc_recall, rfc_f1_score))

print("Linear Regression Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f,
F1 Score = %.2f" % (lrc_accuracy, lrc_precision, lrc_recall, lrc_f1_score))

print("Support Vector Classifier: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1
Score = %.2f" % (svc_accuracy, svc_precision, svc_recall, svc_f1_score))

print("Artificial Neural Network: Accuracy = %.2f, Precision = %.2f, Recall = %.2f, F1
Score = %.2f" % (ann_accuracy, ann_precision, ann_recall, ann_f1_score))

print("Convolutional Neural Network: Accuracy = %.2f, Precision = %.2f, Recall =
%.2f, F1 Score = %.2f" % (cnn_accuracy, cnn_precision, cnn_recall, cnn_f1_score))

```

```

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix


# Random Forest Classifier

rfc_cm = confusion_matrix(y_test, rfc_y_pred)

sns.heatmap(rfc_cm, annot=True, cmap='Blues', fmt='g')

plt.title('Random Forest Classifier')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()


# Linear Regression Classifier

lrc_cm = confusion_matrix(y_test, lrc_y_pred)

sns.heatmap(lrc_cm, annot=True, cmap='Blues', fmt='g')

plt.title('Linear Regression Classifier')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()


# Support Vector Classifier

svc_cm = confusion_matrix(y_test, svc_y_pred)

sns.heatmap(svc_cm, annot=True, cmap='Blues', fmt='g')

```

```
plt.title('Support Vector Classifier')
```

```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.show()
```

```
# Artificial Neural Network
```

```
ann_cm = confusion_matrix(y_test, ann_y_pred)
```

```
sns.heatmap(ann_cm, annot=True, cmap='Blues', fmt='g')
```

```
plt.title('Artificial Neural Network')
```

```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.show()
```

```
# Convolutional Neural Network
```

```
cnn_cm = confusion_matrix(y_test, cnn_y_pred)
```

```
sns.heatmap(cnn_cm, annot=True, cmap='Blues', fmt='g')
```

```
plt.title('Convolutional Neural Network')
```

```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.show()
```

```
from sklearn.metrics import roc_curve, roc_auc_score
```

```
# Random Forest Classifier
```

```
rfc_fpr, rfc_tpr, _ = roc_curve(y_test, rfc_y_pred)
```

```
rfc_auc = roc_auc_score(y_test, rfc_y_pred)
```

```
plt.plot(rfc_fpr, rfc_tpr, label=f'Random Forest (AUC = {rfc_auc:.2f})')
```

```
# Linear Regression Classifier
```

```
lrc_fpr, lrc_tpr, _ = roc_curve(y_test, lrc_y_pred)
```

```
lrc_auc = roc_auc_score(y_test, lrc_y_pred)
```

```
plt.plot(lrc_fpr, lrc_tpr, label=f'Linear Regression (AUC = {lrc_auc:.2f})')
```

```
# Support Vector Classifier
```

```
svc_fpr, svc_tpr, _ = roc_curve(y_test, svc_y_pred)
```

```
svc_auc = roc_auc_score(y_test, svc_y_pred)
```

```
plt.plot(svc_fpr, svc_tpr, label=f'Support Vector (AUC = {svc_auc:.2f})')
```

```
# Artificial Neural Network
```

```
ann_fpr, ann_tpr, _ = roc_curve(y_test, ann_y_pred)
```

```
ann_auc = roc_auc_score(y_test, ann_y_pred)
```

```
plt.plot(ann_fpr, ann_tpr, label=f'Artificial Neural Network (AUC = {ann_auc:.2f})')
```

```
# Convolutional Neural Network
```

```
cnn_fpr, cnn_tpr, _ = roc_curve(y_test, cnn_y_pred)
```

```
cnn_auc = roc_auc_score(y_test, cnn_y_pred)
```

```
plt.plot(cnn_fpr, cnn_tpr, label=f'Convolutional Neural Network (AUC = {cnn_auc:.2f})')
```

```
# Plot the ROC curve
```

```
plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
```

```
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve')
```

```
plt.legend()
```

```
plt.show()
```

```
from sklearn.metrics import precision_recall_curve, average_precision_score
```

```
# Random Forest Classifier
```

```
rfc_precision, rfc_recall, _ = precision_recall_curve(y_test, rfc_y_pred)
```

```
rfc_ap = average_precision_score(y_test, rfc_y_pred)
```

```
plt.plot(rfc_recall, rfc_precision, label=f'Random Forest (AP = {rfc_ap:.2f})')
```

```
# Linear Regression Classifier
```

```
lrc_precision, lrc_recall, _ = precision_recall_curve(y_test, lrc_y_pred)
```

```
lrc_ap = average_precision_score(y_test, lrc_y_pred)
```

```
plt.plot(lrc_recall, lrc_precision, label=f'Linear Regression (AP = {lrc_ap:.2f})')
```



```
# Support Vector Classifier
```

```
svc_precision, svc_recall, _ = precision_recall_curve(y_test, svc_y_pred)
```

```
svc_ap = average_precision_score(y_test, svc_y_pred)
```

```
plt.plot(svc_recall, svc_precision, label=f'Support Vector (AP = {svc_ap:.2f})')
```

```
# Artificial Neural Network
```

```
ann_precision, ann_recall, _ = precision_recall_curve(y_test, ann_y_pred)
```

```
ann_ap = average_precision_score(y_test, ann_y_pred)
```

```
plt.plot(ann_recall, ann_precision, label=f'Artificial Neural Network (AP = {ann_ap:.2f})')
```

```
# Convolutional Neural Network
```

```
cnn_precision, cnn_recall, _ = precision_recall_curve(y_test, cnn_y_pred)
```

```
cnn_ap = average_precision_score(y_test, cnn_y_pred)
```

```
plt.plot(cnn_recall, cnn_precision, label=f'Convolutional Neural Network (AP = {cnn_ap:.2f})')
```

```
# Plot the Precision-Recall Curve
```

```
plt.xlabel('Recall')
```

```
plt.ylabel('Precision')
```

```
plt.title('Precision-Recall Curve')
```

```
plt.legend()
```

```
plt.show()
```

```

import numpy as np

import matplotlib.pyplot as plt

# Define the evaluation metrics

classifiers = ['Random Forest', 'Linear Regression', 'Support Vector', 'Artificial Neural
Network', 'Convolutional Neural Network']

accuracy = [0.98, 0.98, 0.98, 0.99, 0.87]

precision = [1.00, 1.00, 0.99, 1.00, 0.00]

recall = [0.86, 0.88, 0.89, 0.94, 0.00]

f1_score = [0.92, 0.94, 0.94, 0.97, 0.00]

# Set the bar width

bar_width = 0.2

# Define the x positions of the bars

r1 = np.arange(len(classifiers))

r2 = [x + bar_width for x in r1]

r3 = [x + bar_width for x in r2]

r4 = [x + bar_width for x in r3]

# Create the bar chart

plt.bar(r1, accuracy, color='blue', width=bar_width, edgecolor='black',
label='Accuracy')

plt.bar(r2, precision, color='red', width=bar_width, edgecolor='black',

```

```

label='Precision')

plt.bar(r3, recall, color='green', width=bar_width, edgecolor='black', label='Recall')

plt.bar(r4, f1_score, color='orange', width=bar_width, edgecolor='black', label='F1
Score')


# Add labels and title

plt.xlabel('Classifiers')

plt.xticks([r + bar_width for r in range(len(classifiers))], classifiers)

plt.ylabel('Score')

plt.title('Evaluation Metrics for Email Spam Detection')


# Add legend

plt.legend()

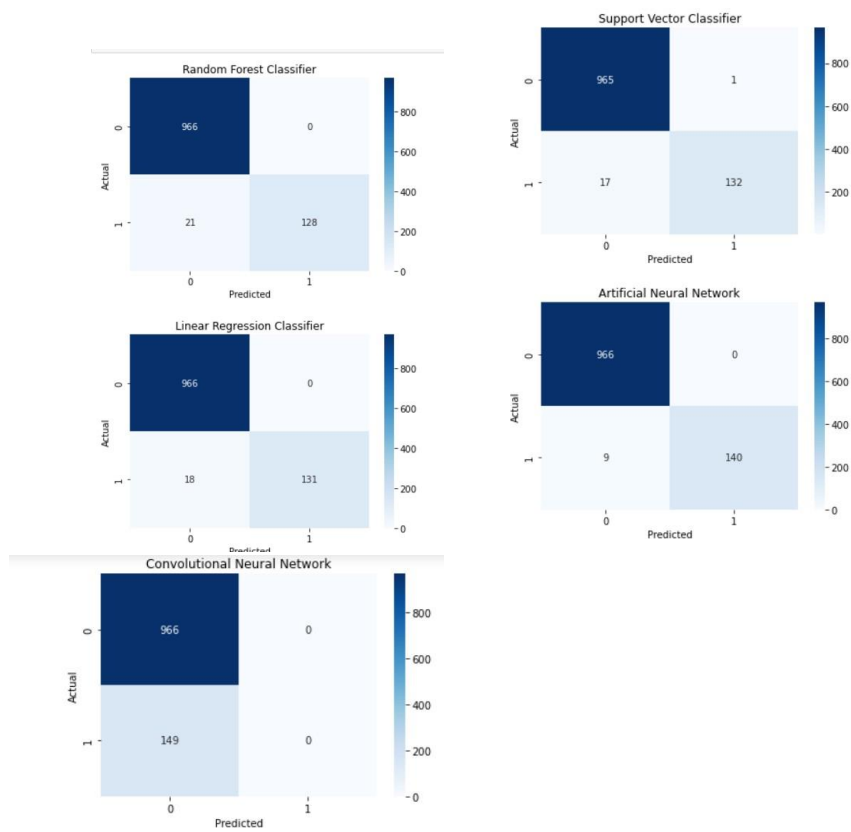

# Show the plot

plt.show()

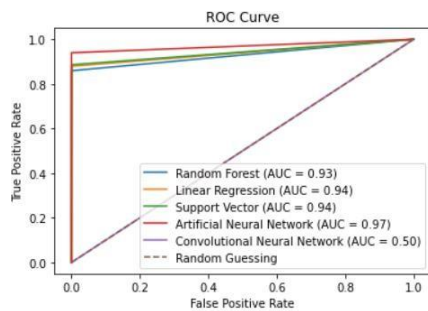
```

Appendix Screenshots

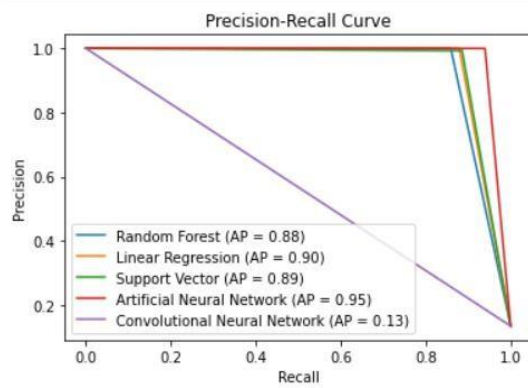
Screenshots Of Outputs:



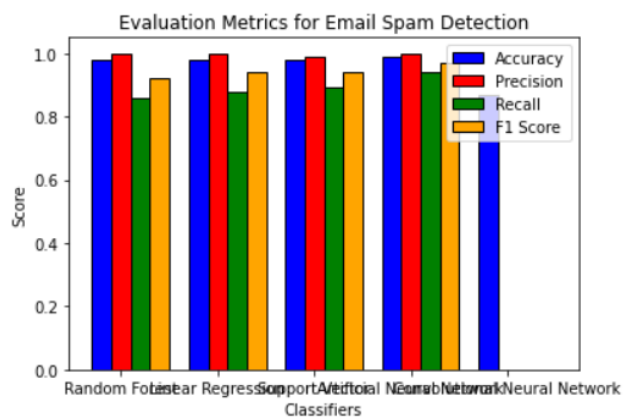
A) ROC CURVE



B) PRECISION RECALL CURVE



C) EVALUATION METRICS FOR EMAIL SPAM DETECTION



Email Spam Detection & Filtering Using Machine Learning

Katakam Siddhartha
Dept. of Computer Science Engineering
(B.E.)
Sathyabama Institute Of Science and
Technology
Chennai, India
katakamsiddhartha123@gmail.com

Kodati Naga Satya Sai Manikanta
Dept. of Computer Science Engineering
(B.E.)
Sathyabama Institute Of Science and
Technology
Chennai, India
manikantakodati8@gmail.com

Abstract—Phishing assaults, in which the perpetrator masquerades as a legitimate source in order to obtain confidential material, are now a serious threat due to the rapid growth of online consumers damaging one's credibility, costing one's money, or infecting one's computer with spyware and perhaps other viruses. Due to their capacity to sift through large amounts of data in search of patterns that can be used to make predictions, intelligent approaches like ML & DL were finding growing usage in the realm of cybersecurity. In this study, we explore the efficacy of using such clever methods to identify phishing websites. We utilised two different data sets and picked the most highly linked attributes, which included both content-based and URL-lexical/domain-based characteristics. After that, many ML models were implemented, and their relative efficacy was assessed. The results demonstrated the significance of selecting features in raising the quality of the models. In addition, the findings attempted to determine the most useful factors that affect the model when it comes to recognizing phishing websites. When it came to classifying data, the Random Forest (RF) algorithm performed best across the board.

Keywords—Deep learning, Random forest, machine learning, and phishing websites.

I. INTRODUCTION

The importance of cyber defense in today's globalized society is rising rapidly. The proliferation of Internet usage and improvements in network technology are linked to the emergence of more sophisticated cyberattacks [1]. One of the most significant concerns is the production and spread of hostile URLs. Newsletters, hyperlinks, pop-up ads, & ads are a few of the common ways that harmful URLs spread. Due to the fact that they host malicious material without permission, their proliferation may lead to monetary loss, personal information being stolen, and even ransomware being downloaded & installed on a computer.

Among the two key methods [2][3] has been utilized to identify fraudulent URLs. The very first method is a blacklisted that is regularly evolving when potentially malicious URLs are discovered. Software like Google Secure Browser, Intel Sites Defender, & Content filtering ThreatSeeker [4] are just a few examples of business products that use this method to detecting harmful URLs. This approach accurately detects fraudulent URLs while being simple to implement. On the other hand, it can't protect users against the dangers of newly discovered malicious URLs.

Heuristic variant of such a method generates characteristics of imminent vulnerabilities & keeps them on the blacklist. By creating and maintaining signature for actions based on its behavioral traits, such approach exhibits enhanced generalisation abilities in comparison to the blacklist-based. Each time a fresh URL is generated, it is checked against stored signature. As soon as a match is made, the URL is flagged as potentially harmful. In addition, since they rely on a predetermined signature, intruders may simply circumvent them, so machines that adopt this strategy will be unable to recognise new malicious URLs.

The 2nd strategy involves the use of AI-based methods, such as ML different classifiers for threat detection through URLs. Researchers have been progressively using this strategy in the past century as a means of fixing the problems encountered by the earlier method. A test set of dangerous and safe URLs is essential for ML. Multiple types of characteristics associated with the URLs may be used to compile the training set. As a result, after an ML model has been trained on this list, it may use the information it has acquired to categorise new URLs.

Throughout this study, we suggested an ML-based method for identifying malicious Url. When conducting our study, we relied on two different datasets: one that was originally written by Mohamad. [5] and another we downloaded through Kaggle [6]. Eleven thousand five hundred and fifty records were included in the initial dataset, whereas fourteen thousand nine hundred and three were in the latter. Domain-based features, URL lexical-based features, and material functionalities were the three primary categories of features included in both samples. Using two sets of data, we selected the highlights with the highest correlation but also conducted a quality assessment of various machine learning classifiers, such as Support Vector Machine (SVM) , Random Forest (RF), and Logistic Regression (LR) .

Out of all the models tested on the initial data, SVM and RF models had a perfect detection rate for spam Detection. Once again, RF achieved the highest accuracy (91.83%) when compared to the other models in the discrete time series. There were additional associated characteristics in the initial dataset, which led to higher outcomes, compared to the second dataset, which did not include these features. Domain-based aspects like a page's prominence and longevity in operation were the primary focus of these additions.

Google indexes is a subdomain attribute that indicates whether or not a webpage is included in Google's search engine results. Anchor tags and access data elements were examples of evidenced elements that contributed to the

website's usability. As a result, these elements may be affecting how well the models can identify whether or not the site is malicious. This article thus analysed the factors that influence models for distinguishing phishing from non-phishing websites. Suitable characteristics choices for enhancing DL and ML performance was also investigated.

II. LITERATURE REVIEW

In order to identify harmful URLs, researchers have explored many AI-based methods, including DL and ML algorithms.

These methods rely on properties associated with URLs to construct and refine models; these features may be broken down into 3 broad categories: Domain-based features, lexical-based features, and material functionalities. The statistical data retrieved out from Url, including its duration, special character count, length, and whether or not it was encrypted, are examples of lexically linked features. Webpage characteristics that pertain to the content itself are known as "information features," and they consist of things like link elements in Html file, linkages such as meta data, and scripts found in the content's underlying Js. In the realm of phrases, "property features" refer to those that are directly associated with the domain part of a website's URL. Domain-based capabilities includes Domain name search queries, option for long, and Ptr inquiry metadata regarding domains and IP addresses.

In order to assess the efficacy of ML models for identifying malicious urls, Khan. [7] tested them on three different datasets. They found that RF and ANN were superior to other techniques of categorization, with an efficiency of over 97.2% being recorded. After demonstrating that the model's performance was unaffected by the removal of irrelevant characteristics from the fields of data, they found the set of overlapping features present in all three databases. Similarly, Awasthi. [8] compared several ML algorithms for spotting malicious urls and found the Random forest had the highest accuracy (96.56 percent).

Kokmaz. [9] looked at 59 distinct aspects connected to the name, subdomain, & route parts of the URL, with a special emphasis on lexical-based variables. They tested a number of Machine learning models on three sets of data, and found that RF had the best overall performance (with a reliability of up to 96.6% over the datasets) while ANN had the quickest runtime.

Site spoofing attempt identification was performed by Hossain. [10] using a publicly available balancing dataset of 10,020 records and 49 characteristics related to Urls linguistic and site properties. With only an F-score of 0.98, they validated the effectiveness of the RF algorithms in detecting phishing websites. Like this, but with two separate datasets. Even with this little data, all of the algorithms achieved an accuracy of 92% or above. Furthermore, from both dataset, the Algo achieved the best accuracy of 96%.

Using fewer attributes and a 3 categorization for the website, Kiruthigah. [12] similarly concentrated on the

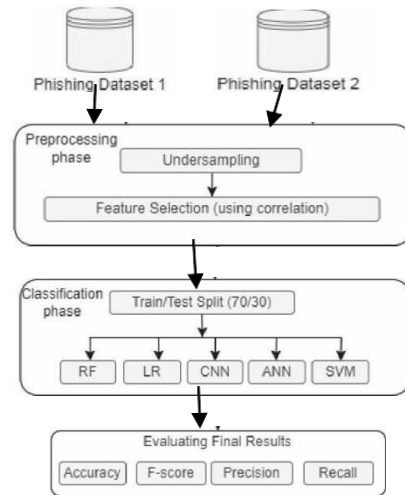
identical two groups of characteristics. Using a dataset of 9 features and 1,352 URLs classified as safe, doubtful, phishing, they used a suite of ML algorithms.

Based on findings, SVM and Decision Tree were the most effective methods, with an accuracy of 91.97%. They also noted that the research was limited because of both the amount of data and the quantity of attributes.

Throughout this research, we compared the efficacy of many DL and ML models designed to identify fraudulent websites. We examine the impact of variable selection on accuracy and illustrate that there are three types of characteristics that are significant.

III. METHODOLOGY

The purpose of this suggested study is to identify and categorise phishing and legal websites. In order to do this, we used two originally released phishing datasets [5][6]. At first, we investigated the datasets to learn more about their structure, size, and limitations. After that, the datasets underwent preprocessing to address the class imbalance problem. The characteristics with the highest correlation were chosen next. Classification models were finally employed, and the outcomes were analysed. The stages in the process are shown in Figure 1 below.



Phishing Dataset 1 1

A. Dataset Description

Two datasets were used in the experiment, and their results were compared. In the first place, We adopted Mohammad[5] 's UCI phishing collection to conduct our research that can be found on Mendeley [13]. There are 32 qualities and 11,056 occurrences in all. No values are missing from it.

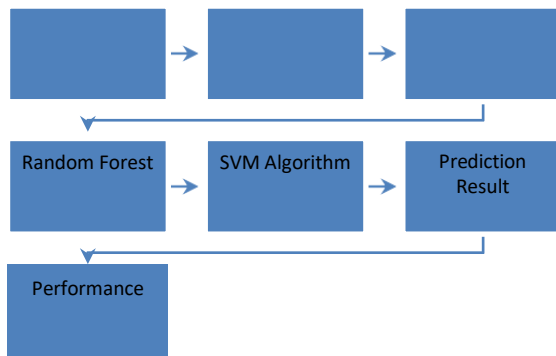
The phishing/legitimate status of a website is shown as a binary value in the categorization property. Only 6158 of

the 11,056 cases are authentic, while 4898 are phishing attempts.

The phishing/legitimate status of a website is shown as a binary value in the categorization property. Out of a total of 14,093 occurrences, we can safely say that 7044 are fraudulent phishing attempts and 7049 are genuine. While it has the same characteristics as the original dataset, this one is missing five features from the original.

Features from the lexicon, the domain, and the content are all present in both datasets.

Our Architecture-



B.Preprocessing

Under-Sampling: For example, in the very first dataset, there were only 6152 occurrences of the genuine classification and 4890 examples of something like the fraudulent subclass, leading to a class-imbalance issue. In response, a technique known as "randomised under-sampling" was used, in which reducing the amount of dominant classifiers until they were about comparable to the amount of minority classifiers examples inside information. Even after accounting for undersampling, fraudulent category still had 4890 occurrences, whereas the legitimate category still contained 4890 occurrences. After under-sampling was applied to the second dataset, it too included 7044 phishing class instances and 7044 legal class instances. A visual representation of the random undersampling method applied to the initial dataset is shown in Figure 2.

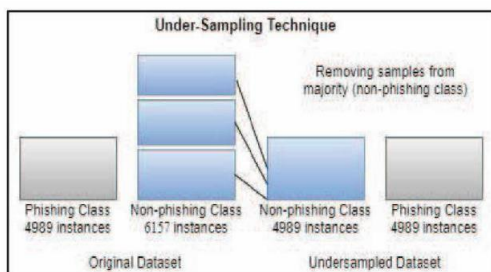


Fig. 2: Under-sampling technique

B.Feature Selection

Features selection is a crucial part of developing a high-performance model, as it helps to identify the traits that are

most important and will lead to improved accuracy. That's why we used a method of features selection known as correlation analysis. To determine the relationship between all of the independent variables and the dependent (target) attribute, statisticians use a technique known as correlation analysis. In our study, the correlation was applied to both datasets. Due to this, we analyzed a combined total of 19 characteristics from both sets to develop our own set of intelligent models and assess how well they performed.

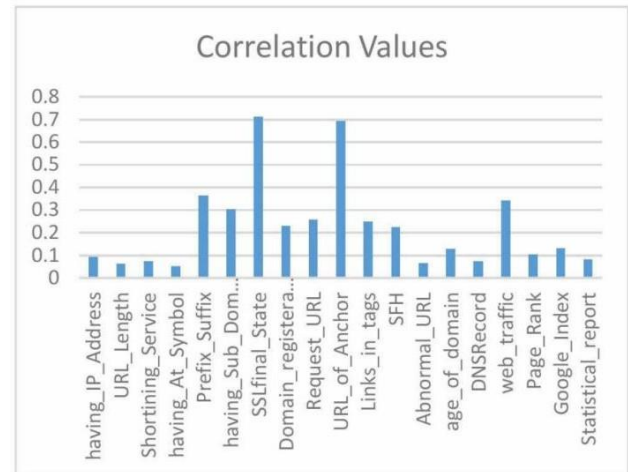


Fig. 3 Correlation values of the first dataset

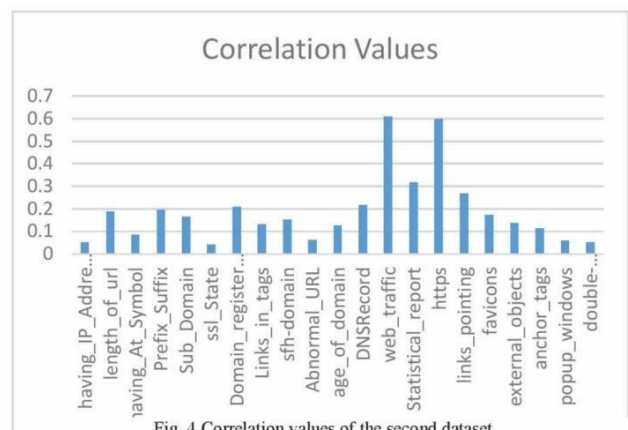


Fig. 4 Correlation values of the second dataset

Table I Description of the features selected in first dataset

#	Attributes	Description	Feature Type
1.	Having_IP_Addr ess	If the domain part has an IP address	Lexical
2.	URL_length	Number of characters in the URL	Lexical
3.	Shortening Service	Is the URL shortened	Lexical
4.	Having_At_Sym bol	Is "@" sign present	Lexical
5.	Prefix_Suffix	Does the domain part include "-" "symbol (separates suffix	Lexical

		prefix)	
6.	Having_Sub_Do main	Is the number of sub- domains high(Greater than 2)	Lexical
7.	SSLfinal_State	Has valid SSL certificate (age)	Domain
8.	Domain_registrat ion_length	Is the domain age of the website short (shorter than a year)	Domain
9.	Request_URL	Are the external objects(images,video s) loaded from another domain	Content
10.	URL_of_Anchor	Is the URL anchor linking to a website	Content
11.	Links_in_tags	Presence of <meta>, <script>,<link> tags	Content
12.	SFH	Does the server from handler(SFH) contain empty strings	Domain
13.	Abnormal_URL	Is the hostname included is WHOIS database	Lexical
14.	Age_of_domain	Is the age of the website short or long	Domain
15.	DNSRecord	If the website is recognized by WHOIS database	Domain
16.	Web_traffic	If the webpage is frequently visited	Domain
17.	Page_Rank	How important the webpage is on the internet	Domain
18.	Google_Index	If the website is in Google's index or not	Domain
19.	Statistical_report	If the host belongs to top phishing IPs or domains(PhishTank)	Domain

After looking at their correlation coefficients, we can see that the first dataset has four key properties that the second dataset does not. In the second dataset, we chose to include all of the missing features from the first dataset as well as the remaining features. In Table III, we see the characteristics that are shared between the two datasets, those that are unique to the first dataset, and those that are unique to the second dataset.

Table II Description of the additional features selected in the second dataset

#	Attributes	Description	Feature Type
1.	external_objects	Any suspicious behaviors related	Content

		to website elements	
2.	anchor_tags	Any suspicious behaviors related to the anchor tag	Content

Table III Common features selected in both the datasets

#	Attributes	Presence of these features in the datasets
1.	Having_IP_Address	Features selected in both datasets
2.	URL_Length	
3.	Shortining_Service	
4.	Having_At_Symbol	
5.	Prefix_Suffix	
6.	Having_Sub_Domain	
7.	SSLfinal_state	
8.	Domain_registration_length	
9.	Links_in_tags	
10.	SFH	
11.	Abnormal_URL	
12.	Age_of_domain	
13.	DNSRecord	
14.	Web_traffic	
15.	Statistical_report	
16.	Request_URL	Features selected in the first dataset but not in the second
17.	URL_of_Anchor	
18.	Page_Rank	
19.	Google_Index	Features selected in the second dataset but not in the first
20.	External_objects	
21.	anchor_tags	

B. Algorithms Used

Random Forest: A machine learning classification technique that creates a number of interdependent decision trees. A forecast is made by each tree in the RF, and the one with the highest votes is used.

Logistic Regression : Classification model in machine learning that uses the connection between independent variables to make predictions. Its primary use is in solving issues of categorical choice.

SVM : A scatter plot with n-dimensional pieces of data using a supervised ML algorithm, where n is the number of characteristics. The next step is to construct a hyperplane that accurately separates the two groups.

C. Evaluation Metrics

The intelligent models' projected and actual classifications were laid out in a four-way table called a confusion matrix, which was used as the basis for the assessment metrics. There are four parts to the distribution: correct results (TP), incorrect results (FP), correct results (TN), and incorrect results (FN) (FN). Accurately identifying phishing websites as such (TP) and mistakenly identifying genuine websites as such (FP) are the two extremes of the phishing prediction spectrum.

IV. EXPERIMENTAL SETUP AND RESULTS

Once the datasets were preprocessed, the intelligent models could be constructed. A computer with just an Intel Core CPU and 4 GB of RAM was used for the tests. Python and Jupyter Notebook were used to create the DL and ML models (Keras and Scikit Learn). Each model was developed using a 70:30 test train split of two sets. At last, models are scored on a variety of metrics, including accuracy, precision, and recall. In Table IV, we detail the various parameter values for each category.

Table IV Parameter settings for each classifier

Model	Parameter	Value
RF	Number of trees	100
SVM	Kernel	linear
LR	Maximum number of iterations	100
CNN	Number of layers	4
	Number of neurons in hidden layers	32,64,63
	Activation function in hidden layers	ReLU
	Number of neurons in output layer	1
	Batch size	32
ANN	Number of hidden layers	2
	Number of neurons in hidden layers	12
	Activation function in hidden layers	ReLU
	Number of neurons in output layer	4
	Activation function in output layer	SoftMax
	Batch size	32

Table V below displays the characteristics chosen as the best by the features selection method, along with the findings acquired by utilising all the features in the original data (without any features methodologies performed) (correlation analysis).

Table V Results when using all features and best features for first dataset

Using all features				
Algorithm	Accuracy	Precision	Recall	F-score
RF	96.53	97.39	95.85	96.61
LR	91.98	92.97	91.41	92.18
SVM	91.98	93.23	91.11	92.16
ANN	93.336	91.92	93.38	93.48
CNN	94.23	95.71	94.47	94.52
Using Best features				
RF	100	100	100	100
LR	99.08	98.91	99.30	99.11
SVM	100	100	100	100
ANN	99.21	99.72	98.71	99.21
CNN	98.19	96.58	99.91	98.23

Table V shows that for the first dataset, RF and SVM created from the best-selected features had a 100% accuracy, which is the greatest of any classifier. After that, the ANN model got to 99.21% accuracy, then LR got to 99.08% accuracy. Classifiers' output - precision, recall, and F-score may also be seen. To summarise, both RF and SVM were able to produce classifiers with 100% accuracy, recall, and f-scores when employing the best-selected features.

Despite this, it's worth noting that RF fared better than CNN, which is surprising given that CNNs are often thought to be better at quickly capturing the relevant information and producing more accurate results. Traditional CNNs, however, need enormous data to be sufficiently successful. The amount of layers upon which they are constructed is also a factor in how well they function. One probable explanation for its poor performance in our trials is the very modest size of the datasets employed. One possible reason for this is that the CNN model used in our experiment was constructed with fewer neurons/layers than would normally be used.

The results of the best features acquired from the attribute selection approach are shown in Table VI below, along with the data achieved by utilising all the characteristics in the discrete time series (with no aspects selection techniques performed) (correlation analysis).

Table VI Results when using all features and best features for the second dataset

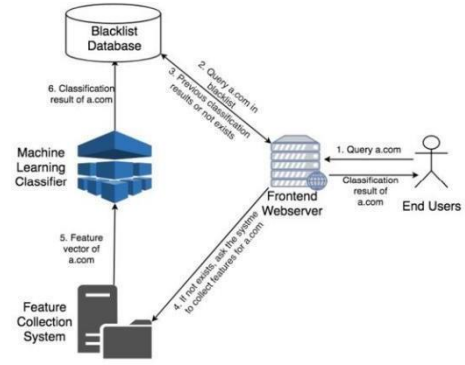
Using all features				
Algorith m	Accurac y	Precisio n	Recall	F-score
RF	92.05	93.10	90.81	91.94
LR	83.81	84.39	82.95	83.66
SVM	86.71	90.23	82.29	86.08
ANN	88.88	89.69	87.40	88.53
CNN	90.26	93.15	88.21	90.61
Using Best features				
RF	92.83	93.71	90.67	92.65
LR	84.29	84.60	83.66	84.18
SVM	86.84	90.26	82.57	86.25
ANN	89.98	92.25	87.38	89.98
CNN	90.46	90.46	88.30	90.24

Using the best-selected attributes, RF obtained a 92.83% accuracy, with CNN coming in as a distant second at a 90% accuracy. In addition, RF achieved the greatest recall and fscore, and had the best accuracy (93.71). However, unlike the first dataset, the models' performance on this one did not improve much when features were selected. Again, we see that CNN's performance is inferior than RF's while being considerably closer to it in all sets of findings. Since the same CNN architecture was employed for both experimental sets, the results might be attributed to either the size of the dataset or the design.

Moreover, we can see from the aforementioned statistics that the features selection strategy resulted in an increase in classification accuracy across the board for both datasets. This means that feature engineering methods are essential for enhancing classifier efficacy.

Since most reputable websites are approved, they show up in the Google index, and they've been there for a while, these may be used as indicators of whether or not the site is a phishing attempt. URL of Anchor and Request URL are two content-based features that look for telltale signs of malicious links by analysing HTML meta tags and inline text, respectively. This in return is critical in determining whether or not the site is a phishing attempt. Also, in the features selection step, these 4 characteristics produced a high significance level for the first dataset, indicating that they indeed play a role in identifying phishing websites.

Management Plan -



Finally, the results of an experiment where ML algorithms are implemented to the other discrete time series using only attributes present in both datasets are shown in the table below (III). However, when compared to the results of the past versions in table VI, which can be seen by looking at the numbers with table VII, there was no substantial improvement. Perhaps this is due to the fact that contextual factors & link ids, two features that were removed from the 2nd sample due to poor correlation values, are responsible for this. Therefore, including them in the models would have been overkill.

TABLE VII Results on using available features of dataset1 in dataset 2

Using all Features				
Algorith m	Accurac y	Precisio n	Recall	F-score
RF	91.98	91.42	92.59	92.01
LR	84.29	82.84	86.37	84.57
SVM	87.41	89.15	85.09	87.08
ANN	89.12	88.05	90.92	89.46
CNN	90.31	90.11	92.14	90.25

V. CONCLUSIONS AND FUTURE WORK

An increasing number of phishing websites and internet users have necessitated innovative methods for detecting these malicious domains. Numerous studies have been conducted on the topic of distinguishing phishing websites from legal ones by using vast datasets and clever approaches. In this post, we've focused mostly on features development as we worked to create a number of modeling software and identify the aspects that influence these systems' ability to identify malicious urls.

We deployed RF, and SVM for feature selection more to enhance the classification accuracy of the deployed models.

Overall, RF performed the best in both circumstances. RF and SVM achieved perfect accuracy on the first dataset owing to the original characteristics study as well as the associated aspects study, but the second dataset did not fair as well. Because it was found that the initial dataset had more associated traits that were lacking in the second. For this reason, we conducted an extensive series of studies to learn what factors influence the recognition of the intelligent models while making the determination of whether or not a site is phishing.

We recommend further research into other datasets and the use of deeper DL models to improve and compare performance, as well as the identification of additional essential factors in detecting phishing sites.

REFERENCES

- [1] M. Aljabri et al., "Intelligent Techniques for Detecting Network Attacks: Review and Research Directions," *Sensors*, vol. 21, no. 21, 2021, doi: 10.3390/s21217070.
- [2] L. AR and S. Thomas, "Detecting malicious URLs using machine learning techniques: A Comparative Literature Review," *Jnt. Res. J. Eng. Technol.*, vol. 6, no. 6, 2019.
- [3] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL Detection using Machine Learning: A Survey," vol. 1, no. 1, pp. 1-37, 2017.
- [4] D. Huang, K. Xu, and J. Pei, "Malicious URL detection by dynamically mining patterns without pre-defined elements," *World Wide Web*, vol. 17, no. 6, pp. 1375-1394, 2014, doi: 10.1007/s11280-013-0250-4.
- [5] R. Mohammad, T. L. McCluskey, and F. Thabtah, "UCI Machine Learning Repository: Phishing Websites Data Set." <https://archive.ics.uci.edu/ml/datasets/phishing+websites> (accessed Nov. 20, 2021).
- [6] S. Abeysooriya, "Phishing Detection Dataset | Kaggle," 2021. <https://www.kaggle.com/sandunabeysooriya/phishing-detectiondataset> (accessed Nov. 17, 2021).
- [7] S. A. Khan, W. Khan, and A. Hussain, "Phishing Attacks and Websites Classification Using Machine Learning and Multiple Datasets (A Comparative Analysis)," in *International Conference on Intelligent Computing Methodologies*, Springer. Cham. 2020, pp. 301-313.
- [8] A. Awasthi and N. Goel, "Phishing Website Prediction: A Comparison of Machine Learning Techniques," in *Data Intelligence and Cognitive Informatics*, Springer. Singapore. 2021, pp. 637-650.
- [9] M. Korkmaz, O. K. Sahingoz, and B. Diri, "Detection of Phishing Websites by Using Machine Learning-Based URL Analysis," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020, pp. 1-7, doi: 10.1109/ICCCNT49239.2020.9225561.
- [10] S. Hossain, D. Sarma, and R. J. Chakma, "Machine learning-based phishing attack detection," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 9, pp. 378-388, 2020, doi: 10.14569/IJACSA.2020.0110945.
- [11] M. Tubyte and A. Paulauskaite-Taraseviciene, "Research on phishing email detection based on URL parameters using machine learning algorithms," *CEUR Workshop Proc.*, vol. 2915, pp. 18- 26, 2021.
- [12] R. Kiruthiga and D. Akila, "Phishing websites detection using machine learning," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2 Special Issue 11, pp. 111-114, 2019, doi: 10.35940/ijrte.B1018.0982S1119.
- [13] T. Olowookere, A. O. Adetunmbi, and J. O. Ajayi, "Phishing dataset." 2021. (accessed Nov. 17, 2021)
- [14] I. Witten, E. Frank, M. Hall, and C. Pal, *Data Mining - Practical Machine Learning Tools and Techniques*, 4th ed. Morgan Kaufmann, 2016.
- [15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning (Adaptive Computation and Machine Learning series)*. MIT Press, 2016.

