PAPER NAME	AUTHOR
191310132134_gtu_report_final-5.pdf	abc abc

WORD COUNT	CHARACTER COUNT
10654 Words	66558 Characters
PAGE COUNT	FILE SIZE
74 Pages	3.3MB
SUBMISSION DATE	REPORT DATE
May 12, 2023 1:13 PM GMT+5:30	May 12, 2023 1:14 PM GMT+5:30

● 44% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 42% Internet database
- 16% Publications database
- Crossref database
- Crossref Posted Content database
- 38% Submitted Works database

● Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 8 words)

MALWARE DETECTION SYSTEM

A PROJECT REPORT

Prepared at

ISO 9001:2008
ISO 27001:2013
CMMI LEVEL-5

Bhaskaracharya National Institute for Space Applications & Geo-informatics
Ministry of Electronics and Information Technology, Govt. of India.
Gandhinagar

Submitted by

SHASHANK G. SHARMA

191310132134

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

Information and Communication Technology

Adani Institute of Infrastructure Engineering, Ahmedabad

Gujarat Technological University, Ahmedabad

MAY, 2022-23

Adani Institute of Infrastructure Engineering

Adani Institute of Infrastructure Engineering, Shantigram Near Vaishnodevi temple,
S.G. Highway, Ahmedabad 382421

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Malware Detection System** has been carried out by **Shashank G. Sharma** under my guidance in partial fulfillment of the degree of Bachelor of Engineering in Information and Communication Technology, 8th Semester of Gujarat Technological University, Ahmedabad during the academic year 2022-23.

Dr. Ritika Ladha	Dr. Ajay Kumar Vyas
Internal Guide	Head of the Department

ISO 9001:2008
ISO 27001:2013
CMMI LEVEL-5

Bhaskaracharya National Institute for Space Applications and Geo-informatics

MeitY, Government of India

Phone: 079 - 23213081   Fax: 079 - 23213091

E-mail: info@bisag.gujarat.gov.in, website: https://bisag-n.in/

CERTIFICATE

This is to certify that the project report compiled by ***Mr. Shashank G. Sharma, Mr. Shubham Patel, Mr. Pratham Patel, and Mr. Yash Soni*** *students of 8th Semester* ***BE - ICT from Adani Institue of Infrastructure Engineering, Gujarat Technological University, Ahmedabad*** *have completed their final Semester internship project satisfactorily. To the best of our knowledge, this is an original and bonafide work done by them. They have worked on a Machine Learning-based web application for "****Malware Detection System,****" starting from January 23rd, 2023 to May 10th, 2023.*

During their tenure at this Institute, they were found to be sincere and meticulous in their work. We appreciate their enthusiasm & dedication towards the work assigned to them.

We wish them every success.

Sidhdharth Patel
External Co-Guide
BISAG- N, Gandhinagar

Harsh Kiratsata
CISO,
BISAG- N, Gandhinagar

Adani Institute of Infrastructure Engineering

Adani Institute of Infrastructure Engineering, Shantigram Near Vaishnodevi temple,
S.G. Highway, Ahmedabad 382421

DECLARATION

We hereby declare that the Internship report submitted along with the Internship entitled **Malware Detection System** submitted in partial fulfillment for the degree of Bachelor of Engineering in **Information and Communication Technology** to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me at BISAG under the supervision of **Dr. Ritika Ladha** and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of the Student	Sign of Student
Shashank G. Sharma	

ACKNOWLEDGEMENT

I would like to express my appreciation to **Bhaskaracharya National Institute for Space Applications and Geo-informatics (BISAG - N),** who gave me the propitious opportunity to do this wonderful internship. Moreover, I would also like to be grateful to the **Adani Institute of Infrastructure Engineering** and **Career Development Cell**, who gave me the wonderful opportunity to do this astonishing internship at Bhaskaracharya National Institute for Space Applications and Geo-informatics.

I am very much obliged to my external guide at my internship **Mr. Siddharth Patel** and my mentor **Mr. Harsh Kiratsata**. We are highly indebted to the Head of Department of Information and Communication Technology, **Dr Ajay Kumar Vyas** for his integral role in providing us with perennial sources of knowledge and allowing us to work at Bhaskaracharya National Institute for Space Applications and Geo-informatics. I would like to present heartfelt appreciation to my internal guide **Dr. Ritika Ladha** at Adani Institute of Infrastructure Engineering for persistently monitoring the project and suggesting desirable changes.

Thank You.

Shashank G. Sharma

ABSTRACT

The emergence of novel and more sophisticated malware has made it challenging to protect computer systems and networks. A robust malware detection system is, therefore, necessary to prevent cyberattacks and safeguard sensitive information. In this project, we developed a machine learning-based malware detection system using a dataset of malware log files.

The dataset comprises API calls as features and hash numbers of files as indices, with the entries indicating the number of API calls made by each file. We explored both traditional machine learning algorithms and deep learning models, including **XGBoost**, **KNN**, **Logistic Regression**, **Random Forest**, and **Sequential MLP** model.

Our experiments showed that XGBoost achieved the highest accuracy of **99.01**% in an imbalanced dataset and **98.65**% in a balanced dataset; compared to other models. To improve our model's performance, we implemented a GridSearchCV pipeline for hyperparameter optimization, used SMOTE for dataset balancing, and employed statistical analysis techniques like the Wilcoxon signed-rank test and Mann-Whitney U test for demonstrating the significance of performance evaluation.

We deployed the trained XGBoost model using **FastAPI** and hosted it on an **AWS-EC2** engine running on an Ubuntu VM. We also provided the pickle file of the trained model for easy integration into other systems. Our model provides an effective way to detect malware with high accuracy and can be deployed in production environments to enhance cybersecurity measures.

Overall, our approach demonstrates the effectiveness of machine learning techniques in detecting malware and highlights the importance of dataset balancing and performance evaluation techniques in improving the model's accuracy. The trained model is readily available for use in real-world applications, and further improvements are also discussed to achieve robustness and effectiveness for malware detection.

LIST OF FIGURES

Fig 3.1 Key logger program 28
Fig 3.2 Malware Plan 28
Fig 3.3 Gantt chart 31
Fig 4.1 Signature based method 33
Fig 4.2 Malware types 36
Fig 4.3 Malware distribution 39
Fig 4.4 ML lifecycle 41
Fig 4.5 Excel file 42
Fig 4.6 High-level ML cycle 42
Fig 5.1 Weka dashboard 44
Fig 5.2 Algorithm accuracies in weka 46
Fig 5.3 Weka Visualization 46
Fig 5.4 Malware detection workflow 47
Fig 5.5 ML in malware 48
Fig 5.6 File graph 48
Fig 5.7 Random Forest workflow 50
Fig 5.8 Sigmoid curve 52
Fig 5.9 KNN model 53
Fig 5.10 XGboost system 54
Fig 5.11 DL NLP 55
Fig 5.12 Web dashboard 56
Fig 6.1 Training and validation accuracy 60
Fig 6.2 Training and validation accuracy 61
Fig 6.3 Box plot of performances 62
Fig 6.4 Heat map of performances 63
Fig 6.5 Heat map of CV metrics 64
Fig 6.6 Heat map of test metrics 65
Fig 6.7 Algorithms comparison 66

LIST OF SYMBOLS, ABBREVIATIONS, AND NOMENCLATURE

Symbol	**Abbreviation**
ML	Machine Learning
DL	Deep Learning
API	Application Program Interface
OS	Operating System
KNN	K- Nearest Neighbor
CV	Cross-Validation
VM	Virtual Machine
EDA	Exploratory Data Analysis
SMOTE	Synthetic Minority Oversampling Technique

LIST OF TABLES

Table 7.1 Actual Data Results 56
Table 7.2 Predicted Data Results 56

TABLE OF CONTENTS

Candidate Declaration i

Acknowledgment ii

Abstract iii

List of Figures iv

List of Abbreviations v

List of Tables vi

Table of Contents .... vii-viii

Chapter 1 Overview of the company 1

1.1 History 1

1.2 Different Product 1

Chapter 2 Overview of different plant 2

Chapter 3 Introduction of project 3-6

3.1 Internship Summary 3

3.2 Purpose 3

3.3 Objective 3

3.4 Tools and Technology 5

3.5 Internship Planning 6

Chapter 4 System Analysis 7-18

4.1 Current System 7

4.2 Weakness of the Current System 8

4.3 Requirement of the New System 10

4.3.1 Malware Types 11

4.3.2 Malware Distribution 13

4.4 System Feasibility 14

4.4.1 System Integration 15

4.5 Activity in Proposed System 15

4.6 Features of the New System 16

4.7 Techniques of the New System 16

4.8 Selection of Software 16

Chapter 5 System Design **19-32**
5.1 System Design and Methodology 19
5.2 Structure Design 22
5.2.1 ML Implementation 22
5.2.2 DL Implementation 30
5.3 Output and Interface Design 31
5.3.1 Samples of Interface 31
Chapter 6 Implementation **33-51**
6.1 Implementation Platform 33
6.2 Module Specifications 33
6.3 Program Code Snapshots 34
6.3.1 Dataset Exploration 34
6.3.2 Traditional ML Model Building 35
6.3.3 DL Model Building 38
6.3.4 Improvisations and Optimizations 40
6.4 Results and Outcomes 44
6.5 Result Analysis 47
Chapter 7 Testing **52-56**
7.1 Testing Plan 52
7.2 Testing Result and Analysis 54
7.3 Testing Results from the Webpage 56
Chapter 8 Limitations and Future Enhancements **57-59**
8.1 Overall Analysis of the Internship 57
8.2 Problems Encountered 58
8.3 Limitations and Future Enhancements 59
Chapter 9 Conclusion & Reference **60-63**
9.1 Conclusion 60
9.2 Reference 61

CHAPTER 1. OVERVIEW OF COMPANY

1.1 History:

In June 1997, realizing the need to have satellite-based communication for training at the state level the "Remote Sensing and Communication Centre" RESECO was established under the Science and Technology Cell, of the Education Department of Gujarat Government. RESECO was renamed to Bhaskaracharya Institute for Space Applications and Geo-Informatics after the great Indian Mathematician of the 12th century, Bhaskaracharya in December 2003.

1.2 Different products:

College to career program: The SATCOM facility comprises an uplink earth station, a control room, a TV studio, and a network of receiving classrooms. This network is used to air practical training for .net and Java teaching sessions conducted by Microsoft and TCS respectively.

Forestry: RESECO implemented India's first geographic information system (GIS) based computer system for the Forests & Environment Department of Gujarat. It is currently used as Coastal Zone Information System.

CHAPTER 2. OVERVIEW OF DIFFERENT PLANT

5 Bhaskaracharya Institute for Space Applications and Geoinformatics (BISAG) is a national agency by the Government of Gujarat to facilitate providing services and solutions in implementing map-based Geospatial Information Systems. BISAG's SATCOM network is a satellite communication network service to provide distant interaction state-wide. Currently, BISAG is working to implement geospatial technologies for the planning and developmental activities of agriculture, land and water resource management, wasteland/watershed development, forestry, disaster management, infrastructure, and education.

- **Satellite** 12 **Communication**: To promote and facilitate the use of Satellite broadcasting networks for distant interactive training, education, and extensions.
- **Remote Sensing Applications**: For inventory mapping, developmental planning, and monitoring of natural and man-made resources.
- **Geo-informatics System**: To conceptualize, create and organize a multi-purpose common digital database for the sector and thematic applications for various users.
- **Photogrammetry:** For the creation of a Digital Elevation Model, Terrain characteristics, Resource planning, etc.
- **Global Navigation Satellite System and Land Survey**: For Location-based services, Geo-referencing, Engineering Applications, and Research.
- **Disaster Management:** To Prepare geospatial information to provide necessary inputs to the Government to assess and mitigate damage in the event of a disaster.
- **Software Development:** To provide low-cost Decision Support Systems, Geoinformatics applications (desktop as well as web-based) to users for wider usage.
- **Technology Transfer:** To transfer technology to many 19 end users.
- **Value Added Services**: To provide tools that can be customized as per the needs of the users. Education, Research, and Training: To provide education, research, and training facilities to promote several end users through Academy for Geoinformatics.

CHAPTER 3. INTRODUCTION TO PROJECT

3.1 INTERNSHIP SUMMARY:

The problem statement introduced to our team during the internship is Malware detection using ML. We have used ML and DL to detect malware files. These files are API calls generated from a computer using Cuckoo sandbox software. Further, we implemented statistical analysis for performance validation. We have performed EDA to visualize and analyze data for malware detection.

3.2 PURPOSE:

In a world where data generation is at its peak, almost 560 thousand new malware species are introduced every day (Jovanovic, 2023). Moreover, a whopping 10 billion attacks caused by malware-related threats were recorded in the entire globe (Jovanovic, 2023). They added that more than 3 quarter of entire electronic devices is infected by some or other type of malware (Jovanovic, 2023). Kaspersky Labs (2017) define malware as “a type of computer program designed to infect a legitimate user's computer and inflict harm on it in multiple ways.”

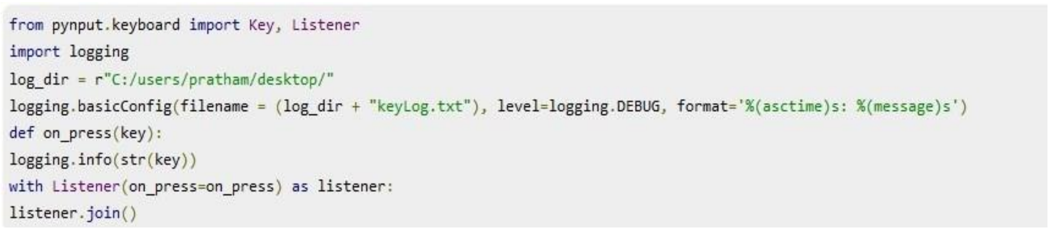
With this exponential growth of malicious software, it becomes very essential to create a safe and proven well-integrated environment to function from information stealing and espionage. While the heterogeneity of malware is at its peak it becomes almost unattainable for the archaic antivirus to fortify our security needs, finding more automated and self-driving security techniques. Keeping in mind the mentioned problems ML based concepts are highly welcomed as they are well advanced than primitive antiviruses and fully automatic.

3.3 OBJECTIVE:

The primary aim of this project is to utilize machine-learning techniques to detect malware frequencies in a system's API-based log files. By automating the detection process, our system can quickly identify any malware in a user's system during an attack, which typically takes cybersecurity personnel a considerable amount of time to detect manually. This capability can significantly improve the response time in the event of a malware attack.

There are mostly two ways the attacker creates malware which is:

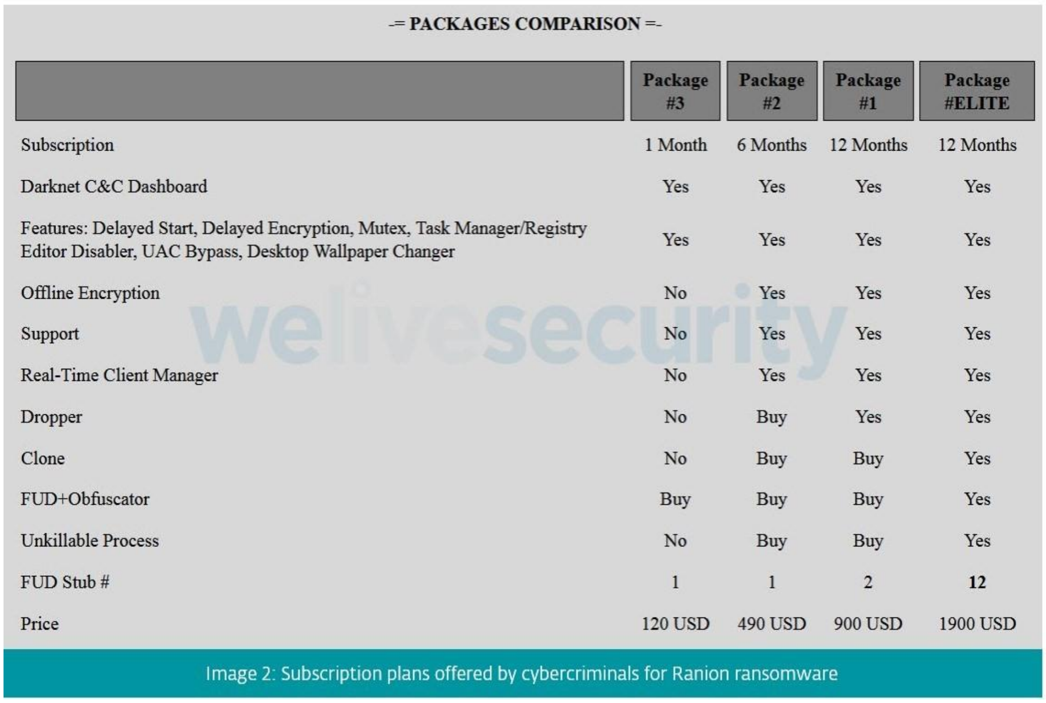
- **Program the malware:** Block codes of malware and virus are readily available on the internet which makes it possible to attack some devices with almost zero skill sets. For example, code for the key logger which tracks every key stock of the user can be programmed with a few lines of Python code.



```
from pynput.keyboard import Key, Listener
import logging
log_dir = r"C:/users/pratham/desktop/"
logging.basicConfig(filename = (log_dir + "keyLog.txt"), level=logging.DEBUG, format='%(asctime)s: %(message)s')
def on_press(key):
logging.info(str(key))
with Listener(on_press=on_press) as listener:
listener.join()
```

Fig 3.1 Key logger program

- **Buy from Black Market:** Purchasing malware from the black market needs almost zero programming and is available for as cheap as 120$.



-= PACKAGES COMPARISON =-

	Package #3	Package #2	Package #1	Package #ELITE
Subscription	1 Month	6 Months	12 Months	12 Months
Darknet C&C Dashboard	Yes	Yes	Yes	Yes
Features: Delayed Start, Delayed Encryption, Mutex, Task Manager/Registry Editor Disabler, UAC Bypass, Desktop Wallpaper Changer	Yes	Yes	Yes	Yes
Offline Encryption	No	Yes	Yes	Yes
Support	No	Yes	Yes	Yes
Real-Time Client Manager	No	Yes	Yes	Yes
Dropper	No	Buy	Yes	Yes
Clone	No	Buy	Buy	Yes
FUD+Obfuscator	Buy	Buy	Buy	Yes
Unkillable Process	No	Buy	Buy	Yes
FUD Stub #	1	1	2	**12**
Price	120 USD	490 USD	900 USD	1900 USD

Image 2: Subscription plans offered by cybercriminals for Ranion ransomware

Fig 3.2 Malware Plans in the black market

As the ease of creating the malware is effortless, it becomes essential to make our system secure using cutting-edge technologies such as ML.

3.4 TOOLS AND TECHNOLOGIES:

23 **Jupyter Notebook:** JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality. We used to program out ML, and DL models as well as to pre-process the data. Moreover, we visualized our data using various inbuilt modules of Jupyter notebook such as matplotlib, and plotly. This integrated environment helped to create PKL files at easy.

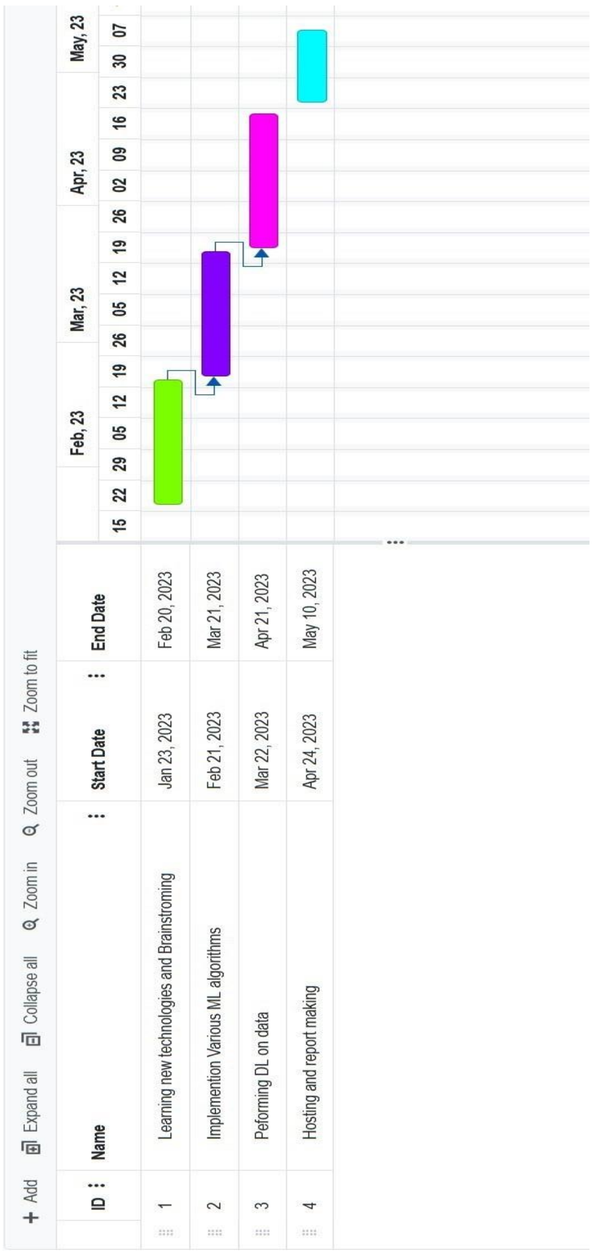
Vs Code: 5 Visual Studio Code is a source code editor that can be used with a variety of programming languages, including C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, and Rust. It is based on the Electron framework, which is used to develop Node.js web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services. As it supports many programming languages, we used Vs code to edit minor Python codes as well as we used to write CSS and HTML scripts for web hosting. We also used this platform for writing JS code for front-end and back-end connections.

35 **Cuckoo Sandbox:** Cuckoo Sandbox is an advanced, extremely modular, and 100% open-source automated malware analysis system with infinite application opportunities. Further, we 22 can analyze many different malicious files (executables, office documents, pdf files, emails, etc) as well as malicious websites under Windows, Linux, macOS, and Android virtualized environments. Due to Cuckoo's open-source nature and extensive modular design, one may customize any aspect of the analysis environment, analysis results processing, and reporting stage. As this platform gives a well-closed and secure environment to run malware files, we used it to create CSV files which have details of files that we will discuss later in upcoming chapters.

18 **Amazon Web Services:** Amazon Web Services offers cloud web hosting solutions that provide businesses, non-profits, and governmental organizations with low-cost ways to deliver their websites and web applications. Whether you are looking for a marketing, rich-media, or e-commerce website, AWS offers a wide range of website hosting options, and we will help you select the one that is right for you. We hosted our website using AWS, which provided a flawless and free way for making the website live. It stored the uploaded file temporarily, then with the server program, finds the output and gives the appropriate response.

Weka: Weka contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions. The original non-Java version of Weka was a Tcl/Tk front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data pre-processing utilities in C, and a make file-based system for running machine learning experiments.

3.5 INTERNSHIP PLANNING:



ID	Name	Start Date	End Date
1	Learning new technologies and Brainstroming	Jan 23, 2023	Feb 20, 2023
2	Implemention Various ML algorithms	Feb 21, 2023	Mar 21, 2023
3	Peforming DL on data	Mar 22, 2023	Apr 21, 2023
4	Hosting and report making	Apr 24, 2023	May 10, 2023

Fig 3.3 Gantt chart

CHAPTER 4. SYSTEM ANALYSIS

4.1 CURRENT SYSTEM

There are mainly two approaches for malware detection, which are signature-based(static) and behaviour-based(dynamic) method. Static detection analyses the code or software without executing it, scanning for known patterns or signatures of previous attacks. Dynamic detection, on the other hand, involves actively running the software in a controlled environment to monitor its behaviour and identify any suspicious actions or changes.

Static Analysis is viewed as just reading the source code of the file and creating the behaviour pattern of properties of the file. It has following techniques:

- **File Format Inspection:** It uses metadata, for illustration windows portable executable files have information on compile time, functions such as import and export.
- **String Extraction:** It generally uses software output of information. It typically includes status and error messages which is used for inferring information on malware operation.
- **Fingerprinting:** This has cryptographic computation, environmental artefacts like hardcoded username, registry.
- **AV Scanning:** If the malware used is well known malware, then it guarantees that anti-virus software will detect the intruder. However, it is irreverent, and mostly used by AV vendors or sandboxes as a confirmation.
- **Disassembly:** This refers to reversing the machine code to assembly language and inferring the software logic and intentions. This is the most common and reliable method of static analysis.

Another analysis type is dynamic analysis. Unlike static analysis, here the behaviour of the file is monitored while it is executing, and the properties and intentions of the file are inferred from that information. Usually, the file is run in the virtual environment, for example in the sandbox. During this kind of analysis, it is possible to find all behavioural attributes, such as opened files, created mutexes, etc. Moreover, it is much faster than static analysis. On the other hand, the static analysis only shows the behavioural scenario relevant to the current system properties. For example, if our virtual machine has Windows 7 installed, the results might be different from the malware running under Windows 8.1.

Now having a deeper insight about malware analysis, we can understand working of Signature based analysis.

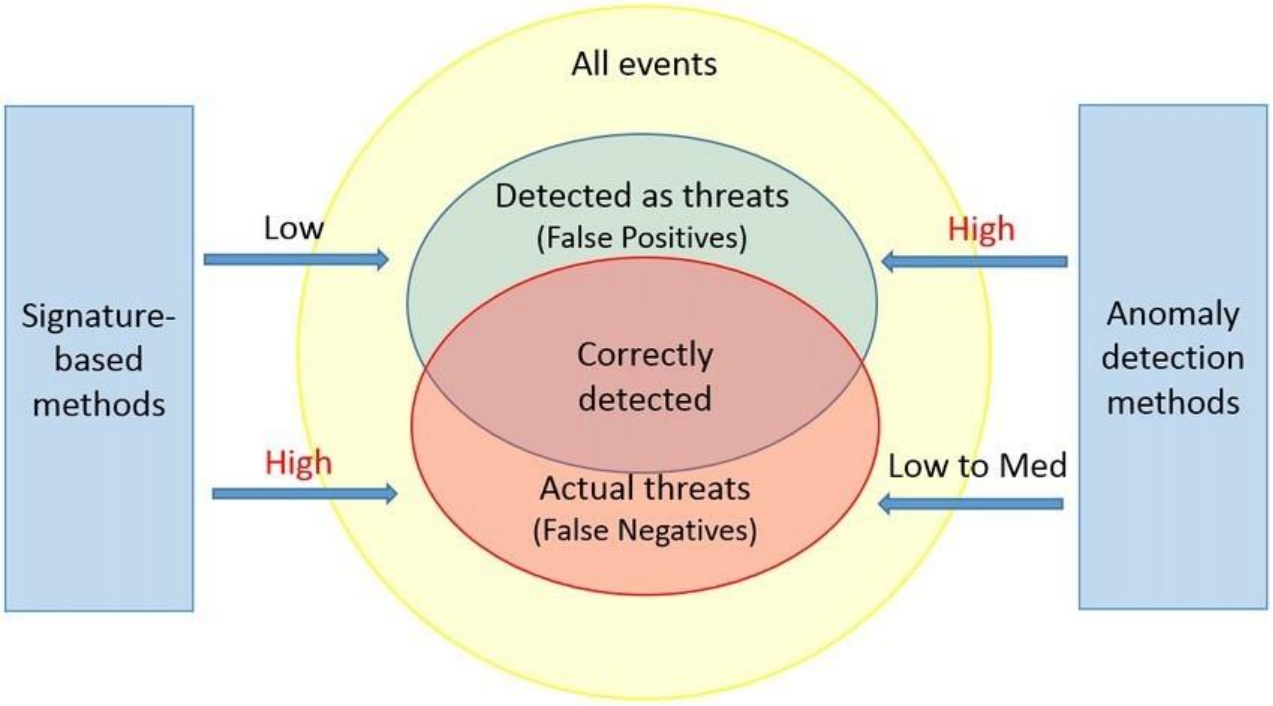


Fig 4.1 Signature based method

15 Signature-based methods usually have low false positive rate (they do not detect legitimate behaviour as an attack), they inevitably have high false negative rate (they do not detect many actual attacks), due to not having attack signatures available for new attacks. Anomaly detection methods, on the other hand, have lower rate of false negatives, as they can detect even new attacks, but their false positives rate is largely increased, because any previously unseen behaviour can be marked as an attack, even if it is legitimate.

4.2 Weakness of Current system:

The following are the drawbacks of current antivirus which can be overcome by using ML.

System Slowdown – Using an antivirus program means tons of resources from the memory and therefore the disk drive is getting used. As a result, it can drastically slow down the overall speed of the pc. Moreover, the method of scanning also can cause lags within the network.

No Complete Protection – If you are employing a free antivirus program, there is no guarantee that it will provide you the entire protection. Moreover, they can identify only certain sorts of threats. So as for acquiring a complete level of protection, you have got to use a firewall also.

Security Holes – When security holes are present inside the OS or the networking software, it will provide an opportunity for the virus to bypass the antivirus software. Unless the user takes action to stay updated, the antivirus software will not be effective.

Limited Detection Techniques – For identifying a possible threat, there is always quite one method available. However, within the case of antivirus programs, it mostly executes the tactic of virus scanning. Sometimes the antivirus programs can offer you false alarms if the scanning matches with the traditional file.

Frequent Advertisements – Apart from premium versions of antivirus programs, through some means, the free antivirus software must generate an income. Advertising is one of the ways to realize them. Many sometimes these advertisements degrade the user experience.

No Customer Support – Unless you buy the premium version, there will not be any customer support given to you. Within the event of any problem, the sole thanks to overcoming are through forums and knowledge bases.

4.3 REQUIRMENT OF NEW SYSTEM:

26 Malware detectors that are based on signatures perform well on previously known malware, which was however was discovered by some AV vendor. Moreover, it fails to detect polymorphic malware which can change its signature and new malwares for which signature ID is new. 4 In turn, the accuracy of heuristics-based detectors is not always sufficient for adequate detection, resulting in a lot of false positives and false-negatives.

Need for the new detection methods is dictated by the high spreading rate of polymorphic viruses. One of the solutions to this problem is reliance on the 11 heuristics-based analysis in combination with machine learning methods that offer a higher efficiency during detection. When relying on heuristics-based approach, there must be a certain threshold for malware triggers, defining the number of heuristics needs for the software to be called malicious. For example, we can define a set of suspicious features, such as “registry key changed, connection established, permission changed.” Then we can state that any software, that triggers at least five features from that set can be called malicious. Although this approach provides some level of effectiveness, it is not always accurate, since some features can have more “weight” than others, for example, “permission changed” usually results in more severe impact to the system than “registry key changed.” In addition to that, some feature combinations might be more suspicious than features by themselves. 1 (Rieck, et al. 2011). To take these correlations into account and provide more accurate detection, machine learning methods can be used. Malware can be very dissimilar in term of their method to attack any system. We will discuss about most used malware types and few details related to their working. Moreover, once we are familiar with the malware types and its mechanism of working, we will throw light in why ML becomes indispensable way to tackle with security threats.

4.3.1 Malware Types:

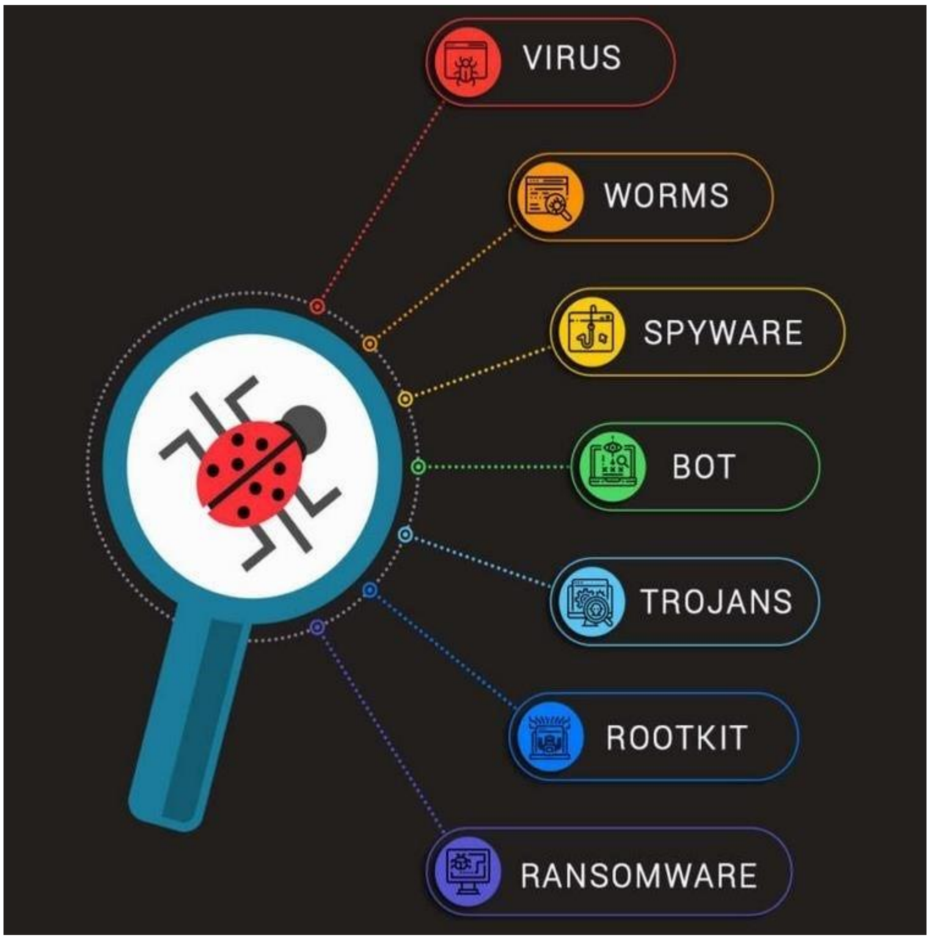


Fig 4.2 Malware types

To have a better comprehensible view in order to undertake concept of logic behind the working of a malware we need to understand how actually different types of malwares effect the electronic device. Mostly used malware are listed below:

- **Virus:** The virus is most simple type of malware. It is just simple few blocks of code which infects the user's system just like the biological one infects the body. It is installed in guileless users further this virus replicate itself causing system failure and extravagant memory usage.

- **Worm:** Basic operation of worm is identical as of virus. The main and central difference between both is that worm can clone itself in multiple devices and networks, resulting in making it more brutal.
- **Trojan:** It is more vicious then above two malwares. Trojan basically works by disguising itself in a needy and legitimate program, as a result the computer system is unable to recognize it as malware. Once injecting itself in the system it will execute the task which is programmed to do such as spying, stealing.
- **Adware:** Adware is harmless malware yet very annoying to handle. Sole purpose of adware is to pop the advertisement on the users devise. These advertisements can be very irksome and can lead to superfluous usage of memory.
- **Spyware:** As the name suggests, this malware preforms espionage the user's computer. Central goal of spyware is to slipstream users' action, mainly used spyware is keyloggers as we discussed in early section. Spyware records and maintains log files of users' action online and offline. It pilfers important passwords and sensitive data.
- **Rootkit:** It is a basic malware code that works by maliciously providing root level privilege to attacker (administrative). It can give trespasser the higher permission. They are generally very hard to trace and remains unnoticeable to many anti viruses, making it almost impossible to remove once infected.
- **Backdoor:** The backdoor is a type of malware that provides an additional secret "entrance" to the system for attackers. By itself, it does not cause any harm but provides attackers with broader attack surface. Because of this, backdoors are never used independently. Usually, they are preceding malware attacks of other types.
- **Keylogger:** key logger or popularly known as keystroke logger is a tool that notes each key stroke a user makes. It is generally used to track password a user makes in a foreign instrument for

example in ATM, if a user enters pin, then key logger can track 4-digit passcode and can make fallacious financial transactions.

- **Ransomware:** This type of malware aims to encrypt all the data on the machine and ask a victim to transfer some money to get the decryption key. Usually, a machine infected by ransomware is “frozen” as the user cannot open any file, and the desktop picture is used to provide information on attacker’s demands. (Savage, Coogan and Lau 2015).
- **Remote Administration Tools (RAT):** This malware type allows an attacker to gain access to the system and make possible modifications as if it was accessed physically. Intuitively, it can be described in the example of the TeamViewer, but with malicious intentions.

4.3.2 Malware Distribution:

The below graph shows how the usage of malware is distributed according to the usage. It is a clear take over that most of the used malware in the industry is trojans. Now a very notable thing is that this is trojan in general, it may include other malwares which are used in form of trojan. Moreover, we have viruses as 2nd widely used malware which contributes to 15 percent. Other than this we have worms and backdoors which have a total of 13 percent combined.

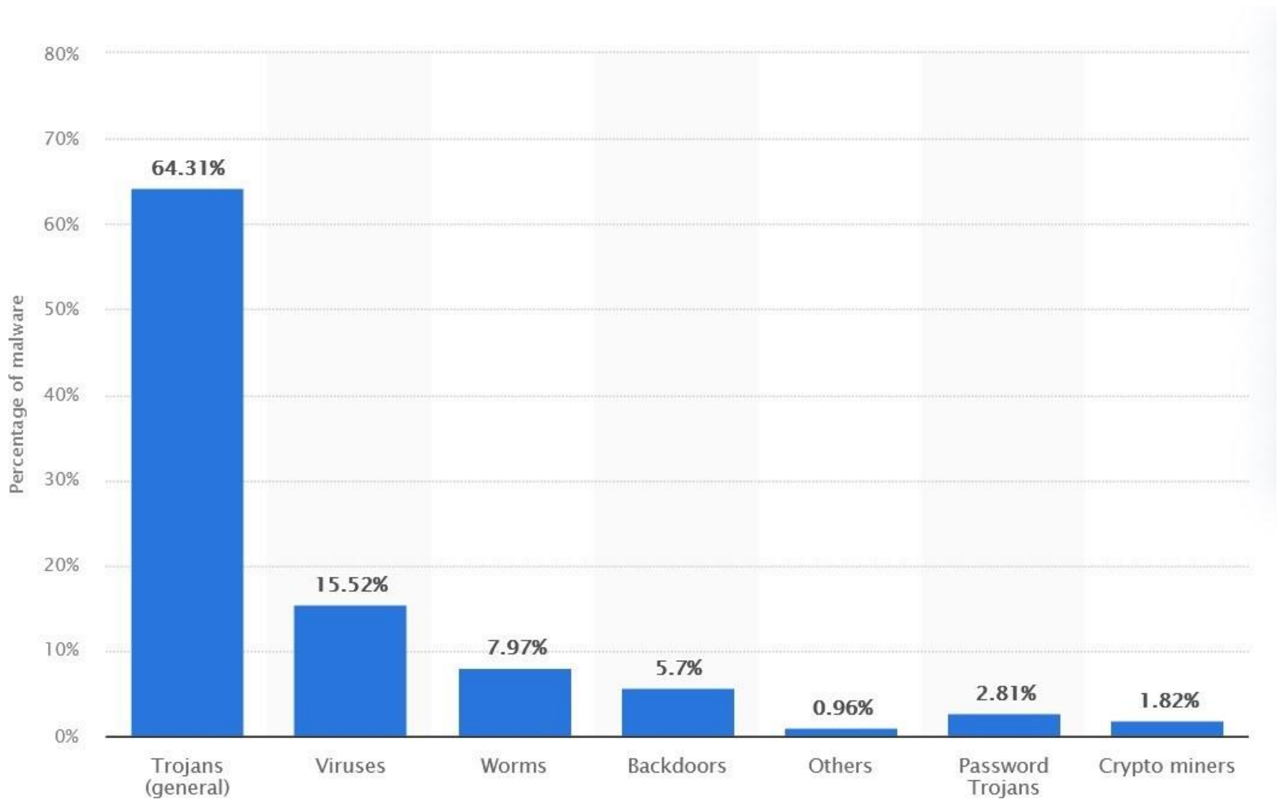


Fig 4.3 Malware Distribution

4.4 SYSTEM FEASIBLITY:

ML in whole is a mathematical as well as probabilistic model, as a result it requires tons of computation. As a human we may feel it is very easy and trivial to do such things when compared to 1000s lines of code of python. There are basically four steps involved in any ML program which

are:

- Processing input data
- Training the deep learning model
- Storing trained deep learning model
- Development of model

Compared to all three training the model takes the most of memory requirement. It should be noted that time complexity plays important role in any code. And accuracy of any model can be accomplished simply by performing all the operations at the same time, instead of taking them one after the other. This is where the GPU comes into the picture, with several thousand cores designed to compute with almost 100% efficiency. Turns out these processors are suited to perform the computation of neural networks as well.

We have used following specs for implementing our project:

- **Device name: LAPTOP-L8OI1678**
- **Processor: Intel(R) Core (TM) i5-8265U CPU @ 1.60GHz 1.80 GHz**
- **Installed RAM: 8.00 GB (7.89 GB usable)**
- **OS used: Linux and Windows**

With these features, it took almost 7 hours to run the pipelining code which we will discuss in later sections of our report.

4.4.1 System Integration:

This project can be easily implemented using different system provided they meet hardware requirement. Moreover, we can effortlessly use log files generated by computer.

4.5 Activity in proposed System:

A general workflow of a ML project is given below.

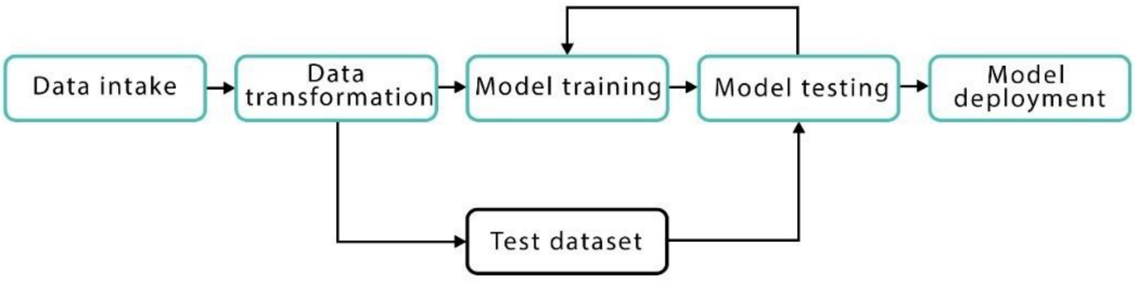


Fig 4.4 ML lifecycle

Data Intake: Loading the data in using Pandas and creating data frames.

Data Transformation: At this point, the data that was loaded at step 1 is transformed, cleared, and normalized to be suitable for the algorithm. Data is converted so that it lies in the same range, has the same format, etc. At this point feature extraction and selection, which are discussed further, are performed as well. In addition to that, the data is separated into sets – 'training set' and 'test set'.

Data from the training set is used to build the model, which is later evaluated using the test set.

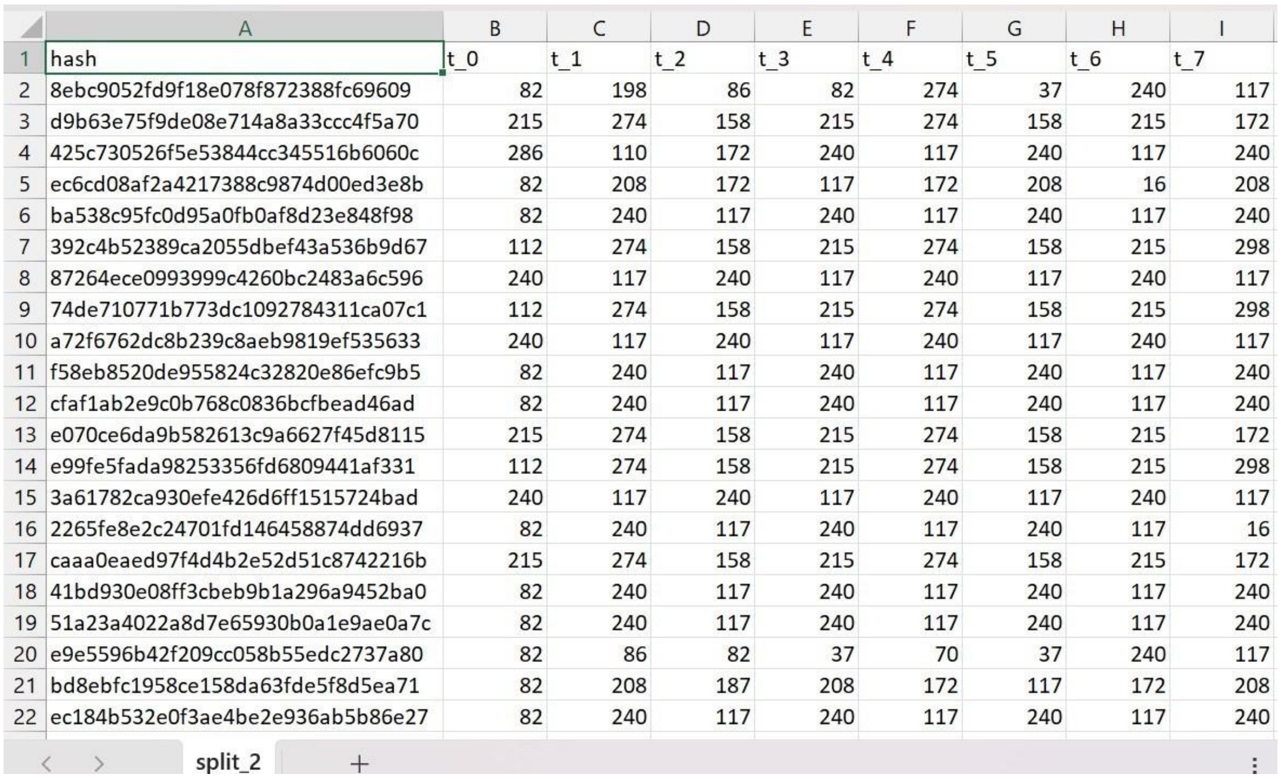
Model Training: Here the model is ready to be implemented.

Model Testing: The model that was built or trained during step 3 is tested using the test data set, and the produced result is used for building a new model, that would consider previous models, i.e.
"learn" from them.

Model Deployment: At this stage, the best model is selected (either after the defined number of iteration or as soon as the needed result is achieved).

We have used API call sequence which has 42797 malware and 1079 good ware API call sequence each. Main problem while implementing the project was lack of public domain PE dynamic malware analysis dataset for training and evaluating model. In order to collect API call sequences, we used cuckoo sandbox, which is open-source malware analysis tool. Its main advantage is it implement malware files in a closed environment.

The file generated is presented below:



	A	B	C	D	E	F	G	H	I
1	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
2	8ebc9052fd9f18e078f872388fc69609	82	198	86	82	274	37	240	117
3	d9b63e75f9de08e714a8a33ccc4f5a70	215	274	158	215	274	158	215	172
4	425c730526f5e53844cc345516b6060c	286	110	172	240	117	240	117	240
5	ec6cd08af2a4217388c9874d00ed3e8b	82	208	172	117	172	208	16	208
6	ba538c95fc0d95a0fb0af8d23e848f98	82	240	117	240	117	240	117	240
7	392c4b52389ca2055dbef43a536b9d67	112	274	158	215	274	158	215	298
8	87264ece0993999c4260bc2483a6c596	240	117	240	117	240	117	240	117
9	74de710771b773dc1092784311ca07c1	112	274	158	215	274	158	215	298
10	a72f6762dc8b239c8aeb9819ef535633	240	117	240	117	240	117	240	117
11	f58eb8520de955824c32820e86efc9b5	82	240	117	240	117	240	117	240
12	cfaf1ab2e9c0b768c0836bcfbead46ad	82	240	117	240	117	240	117	240
13	e070ce6da9b582613c9a6627f45d8115	215	274	158	215	274	158	215	172
14	e99fe5fada98253356fd6809441af331	112	274	158	215	274	158	215	298
15	3a61782ca930efe426d6ff1515724bad	240	117	240	117	240	117	240	117
16	2265fe8e2c24701fd146458874dd6937	82	240	117	240	117	240	117	16
17	caaa0eaed97f4d4b2e52d51c8742216b	215	274	158	215	274	158	215	172
18	41bd930e08ff3cbeb9b1a296a9452ba0	82	240	117	240	117	240	117	240
19	51a23a4022a8d7e65930b0a1e9ae0a7c	82	240	117	240	117	240	117	240
20	e9e5596b42f209cc058b55edc2737a80	82	86	82	37	70	37	240	117
21	bd8ebfc1958ce158da63fde5f8d5ea71	82	208	187	208	172	117	172	208
22	ec184b532e0f3ae4be2e936ab5b86e27	82	240	117	240	117	240	117	240

split_2

Fig 4.5 Excel file

It has hash numbers for each API sequence and t_1 to t_99 is values.

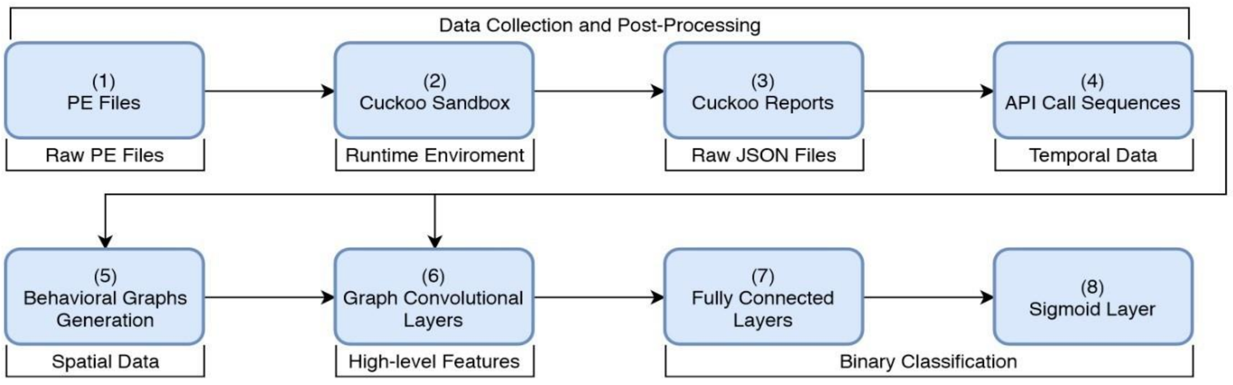


Figure 1: High-level flow of the proposed method.

Fig 4.6 High level ML cycle

4.6 Features of New system:

With the help of proposed model, we can automatically detect malware with the help of ml algorithm. The problem with the previous system was that it was unable to find the malware with the new ID.

With implementing ml on malware detection, we can easily find malicious file without any problem moreover we have implemented deep learning models in order to increase the efficiency.

4.7 Techniques of New System:

We have used 4 machine learning algorithms to implement our data. We used Random Forest, KNN, logistic regression, and XGBoost. Further we compare the accuracies of all those algorithms we have statistically represented those accuracies with the help of data visualization techniques in Python we have used modules such as matplotlib and pyplot to plot graphs. We have used pipelining to generate the pkl files which we used to deploy model on web.

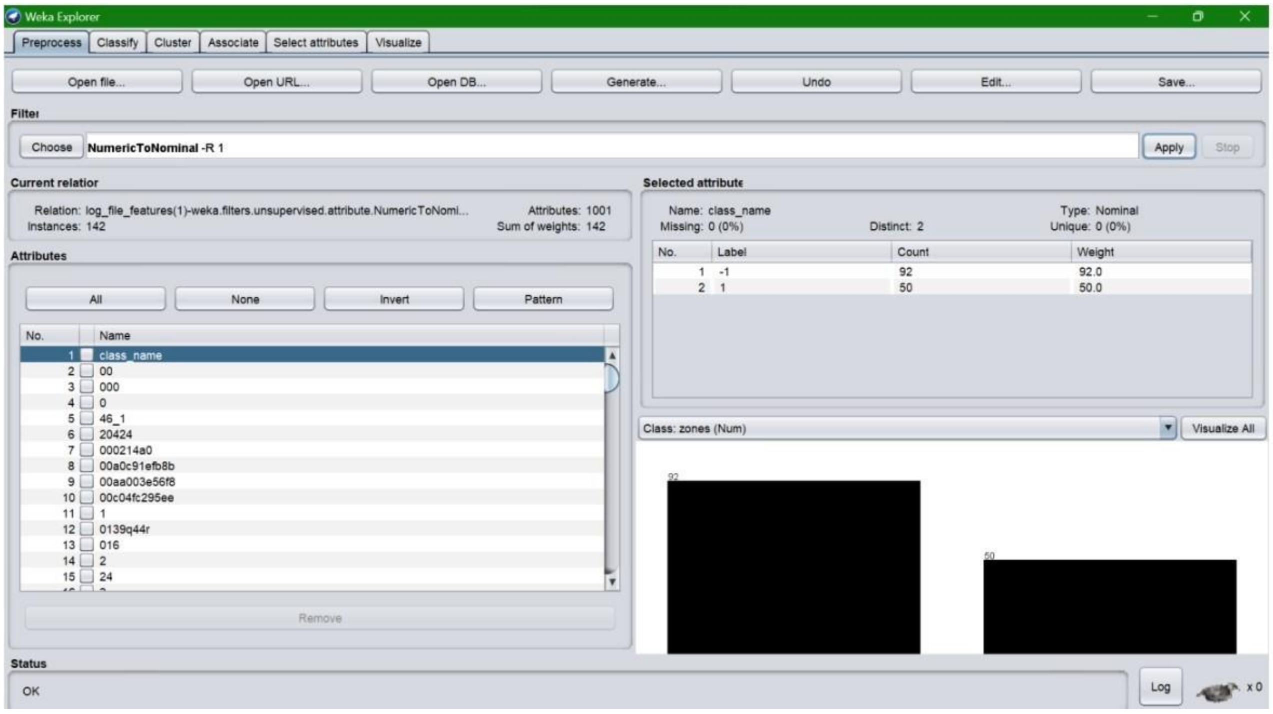
4.8 Selection of software:

Software used: Jupyter notebook, Vs Code, Cuckoo Sandbox, Amazon Web Services, Weka.

CHAPTER 5. SYSTEM DESIGN

5.1 System Design and methodology:

Basic process of project started by doing brainstorming related to the word statement. We implemented our project with two basic method which are using weka and programming. In weka it is very straight forward process, we just have to upload the file in weka and have to pre-process (change the basic data types) further we just need to select the desired algorithm and can get the result. Below is the photo from weka dashboard.



Filter: NumericToNominal -R 1

Current relation

Relation: log_file_features(1)-weka.filters.unsupervised.attribute.NumericToNomi... Attributes: 1001
Instances: 142 Sum of weights: 142

Attributes

No.	Name
1	class_name
2	00
3	000
4	0
5	46_1
6	20424
7	000214a0
8	00a0c91efb8b
9	00aa003e56f8
10	00c04fc295ee
11	1
12	0139q44r
13	016
14	2
15	24

Selected attribute

Name: class_name Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count	Weight
1	-1	92	92.0
2	1	50	50.0

Class: zones (Num)

Status

OK

Fig 5.1 Weka Dashboard

Moreover, we implemented navie bays algorithm in weka let us get more details on this algorithm. Naive Bayes is the classification machine learning algorithm that relies on the Bayes Theorem. It can be used for both binary and multi-class classification problems. The main point relies on the idea of treating each feature independently. Naive Bayes method evaluates the probability of each feature independently, regardless of any correlations, and makes the prediction based on the Bayes Theorem.

That is why this method is called “naive” – in real-world problems features often have some level of correlation between each other. We have two probabilities which are:

- 1 **Class Probability:** It is a probability of a class in the dataset. In other words, if we select a random item from the dataset, this is the probability of it belonging to a certain class.
- **Conditional Probability:** is the probability of the feature value given the class.

Formulas for each probability can be calculated simply

For class probability it is

$$P(C) = \frac{count(instances\ in\ C)}{count(instances\ in\ Ntotal)}$$

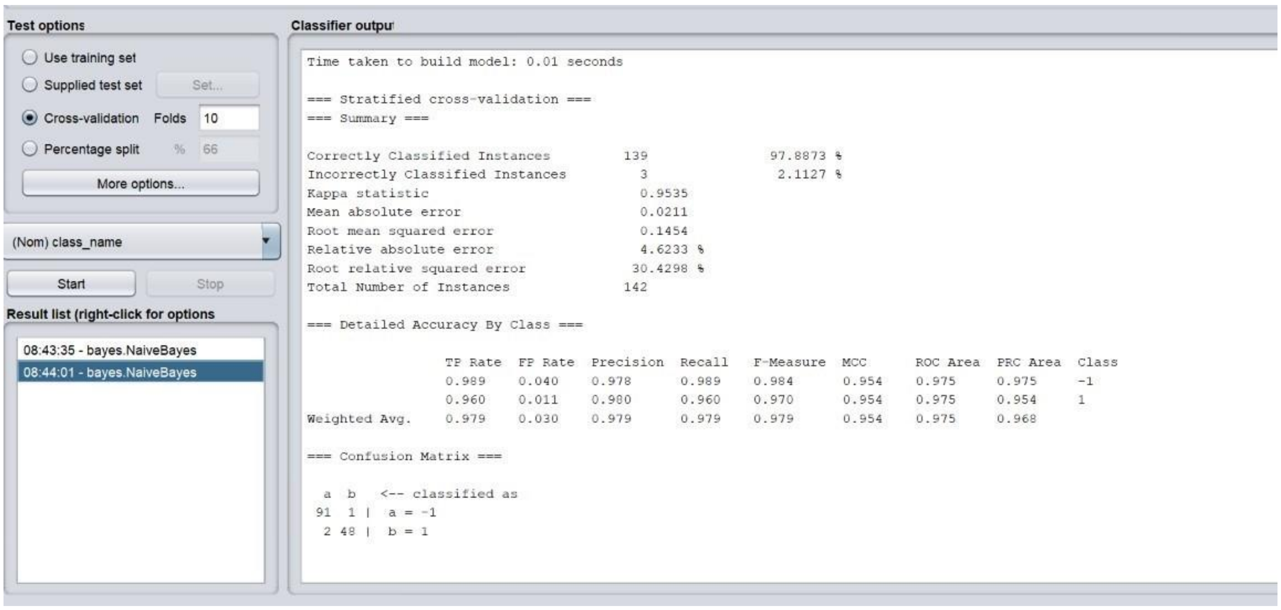
For conditional probability it is

$$P(V|C) = \frac{count(instances\ with\ V\ and\ C)}{count(instances\ with\ V)}$$

31 Given the probabilities, we can calculate the probability of the instance belonging to a class and therefore make decisions using the Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

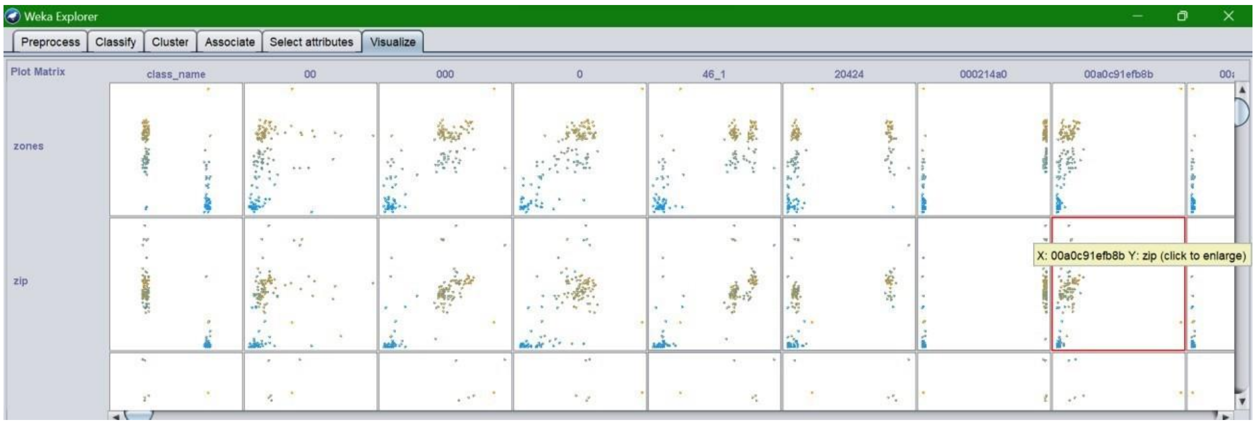
Following is our output for naive baye algorithm. We have used 10-fold cross validation with 66 split percent. Correctly classified instance has a very good accuracy of 97.88%. the model has a mean absolute error of minor 0.021%



```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         139               97.8873 %
Incorrectly Classified Instances         3                2.1127 %
Kappa statistic                          0.9535
Mean absolute error                      0.0211
Root mean squared error                  0.1454
Relative absolute error                  4.6233 %
Root relative squared error             30.4298 %
Total Number of Instances              142

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.989    0.040    0.978      0.989    0.984      0.954    0.975     0.975     -1
                 0.960    0.011    0.980      0.960    0.970      0.954    0.975     0.954     1
Weighted Avg.    0.979    0.030    0.979      0.979    0.979      0.954    0.975     0.968

=== Confusion Matrix ===

  a  b   <-- classified as
 91  1 |  a = -1
  2 48 |  b = 1
```

Fig 5.2 Algorithm accuracies in weka

The below image shows the visualization of weka which has scatter plot.



`

Fig 5.3 Weka visualization

5.2 Structure Design:

5.2.1 ML Implementation:

17 The machine learning-based malware detection process mainly includes two stages: training and detection, as shown in Fig. In the former stage, the analysts usually extract features from the samples set and then employ the features to train the automatic classifier; in the latter stage, the features will first be extracted from the samples to be detected, and then input into the trained classifier to obtain a decision result.

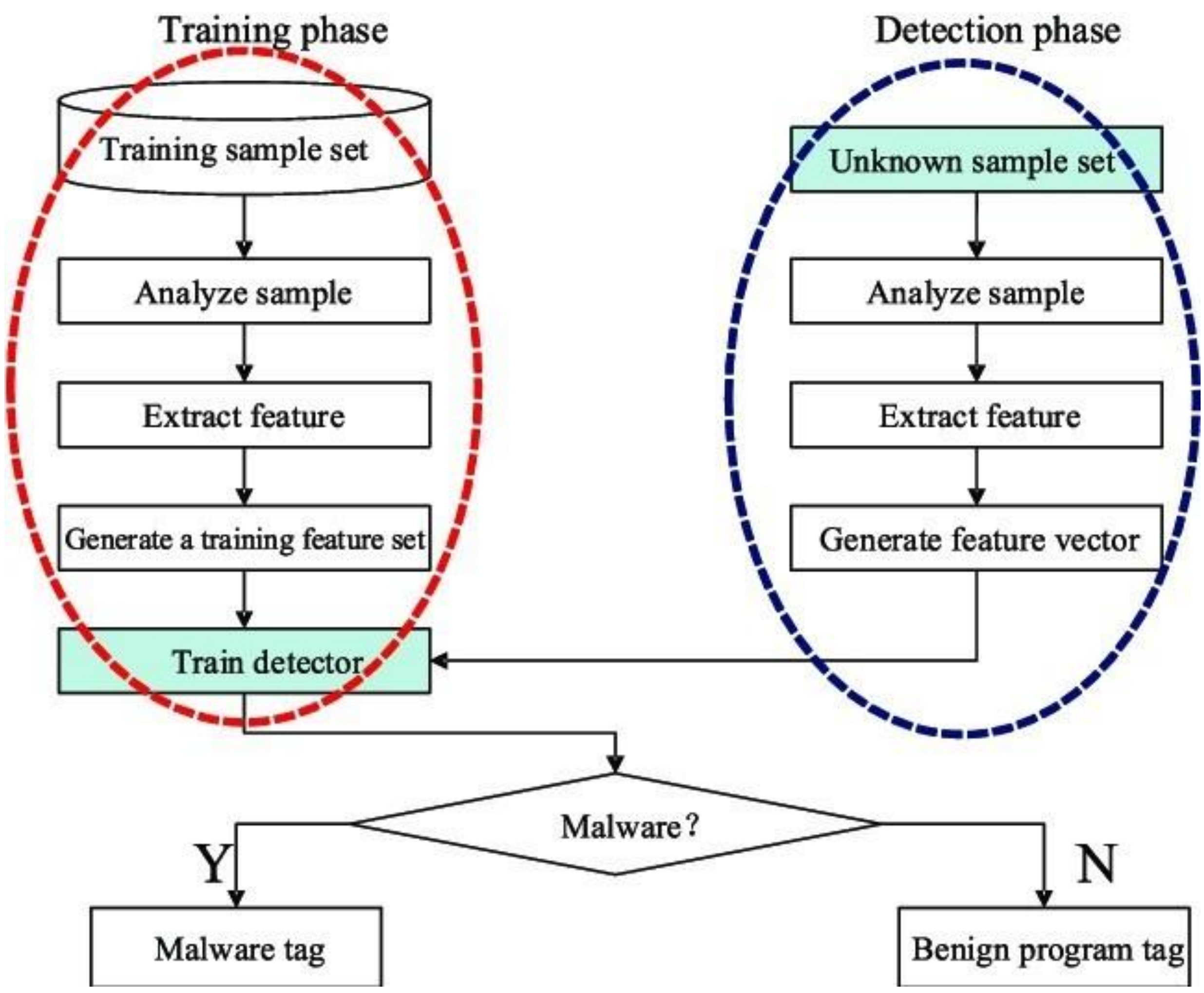


Fig 5.4 Malware detection workflow

In the training phase we select the model which we will implement, either neural network or decision tree. Generally, it depends on parameters like accuracies for which model to select. After that we trained our model and verified its quality, the next phase is applying the model to new objects. Here the parameters do not change we only predict. In malware detection this is also known as protection phase. Often Vendors deliver a trained model to consumers where the product makes decision based on prediction autonomously.

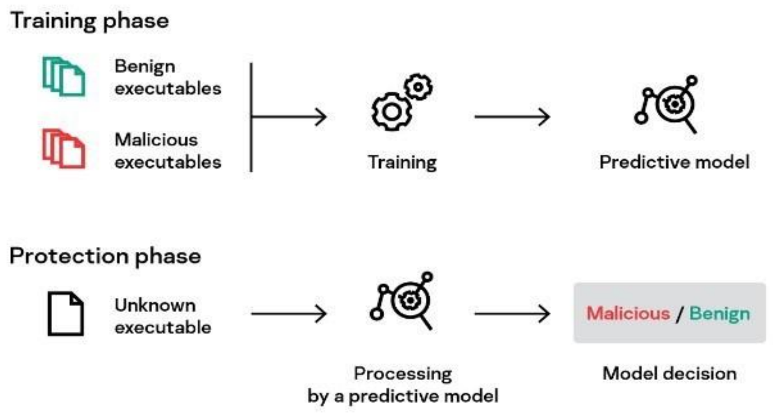


Fig 5.5 ML in malware

After data pre-processing and EDA the initial data visualization results we got was as follows:

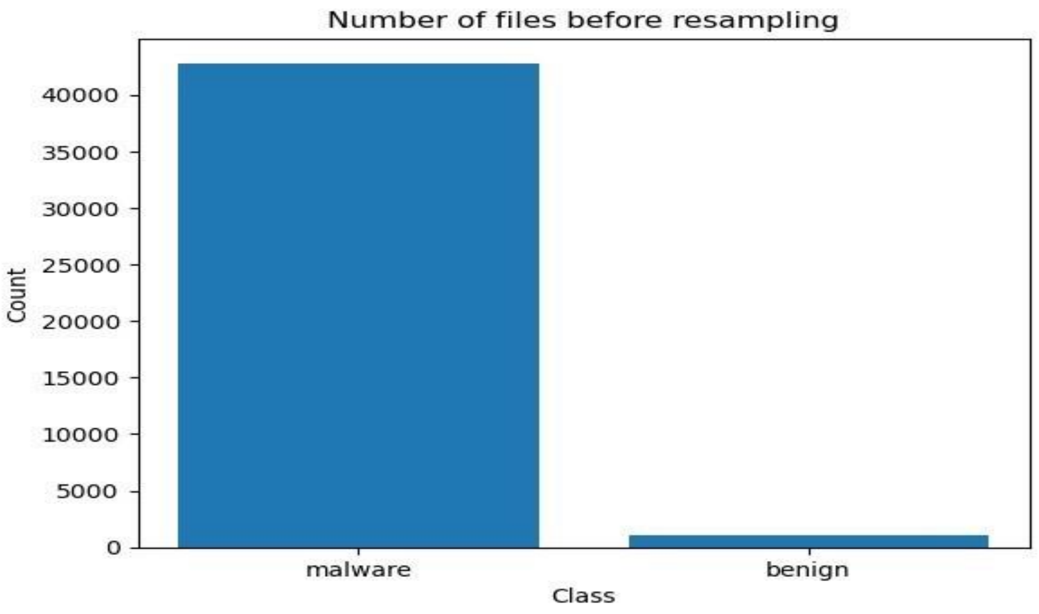


Fig 5.6 File graph

Here, 42797 files were malware and on the flip side, 1079 files were benign. We have used following ML algorithms for implementing our model.

- Random forest
- Logistic Regression
- KNN
- XGboost

Let us discuss each model in detail and then we will see the accuracies of each model along with performance matrix.

4 **Random Forest:** Random Forest is one of the most popular machine learning algorithms. It requires almost no data preparation and modelling but usually results in accurate results. Random Forests are based on the decision trees described in the previous section. More specifically, Random Forests are the collections of decision trees, producing a better prediction accuracy. That is why it is called a 'forest' – it is basically a set of decision trees.

The basic idea is to grow multiple decision trees based on the independent subsets of the dataset. At each node, n variables out of the feature set are selected randomly, and the best split on these variables is found. In simple words, the algorithm can be described as follows.

Firstly, multiple trees are roughly built on 62 percent of training data. Many 25 predictor variables are randomly selected out of all predictor variables. Further best split from this variable is used to split. By default, the amount of the selected variables is the square root of the total number of all predictors for classification, and it is constant for all trees.

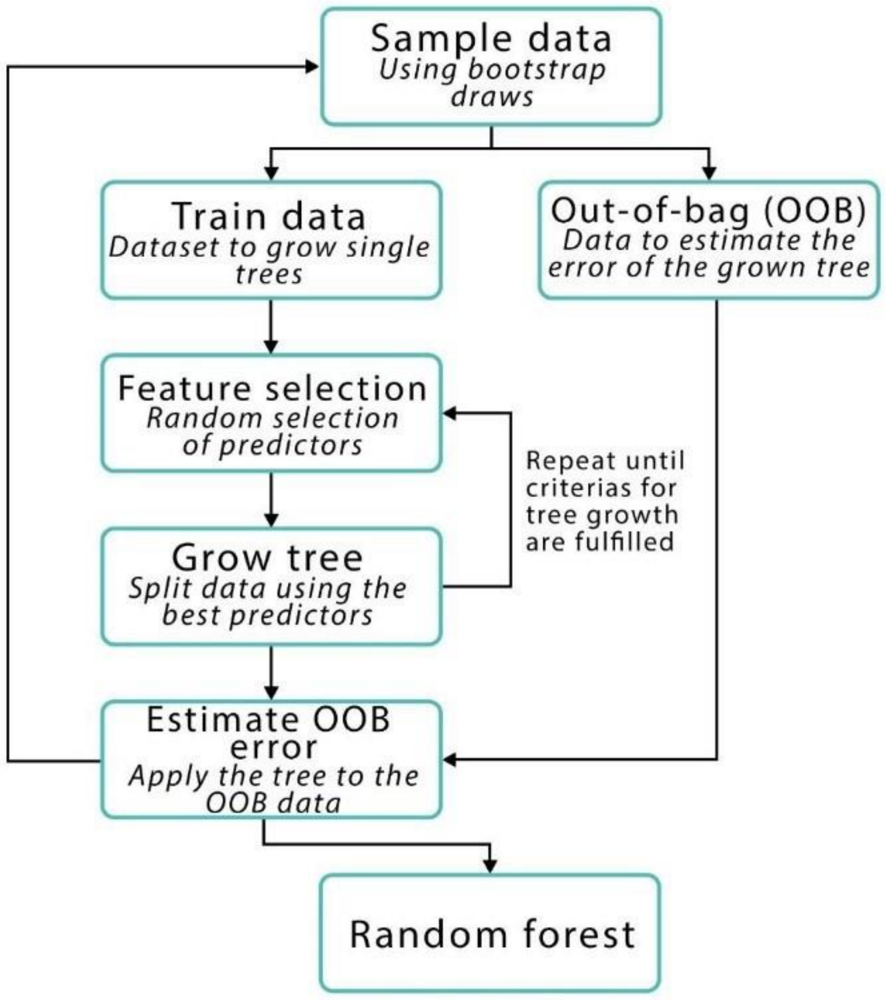

Sample data
Using bootstrap draws
Train data
Dataset to grow single trees
Out-of-bag (OOB)
Data to estimate the error of the grown tree
Feature selection
Random selection of predictors
Repeat until criterias for tree growth are fulfilled
Grow tree
Split data using the best predictors
Estimate OOB error
Apply the tree to the OOB data
Random forest

Fig 5.7 Random Forest workflow

Random forests inherit many of the advantages of the decision trees algorithms. They are applicable to both regression and classification problems; they are easy to compute and quick to fit. They also usually result in the better accuracy. However, unlike decision trees, it is not very easy to interpret the results. In decision trees, by examining the resulting tree, we can gain valuable information about which variables are important and how they affect the result. This is not possible with random forests. It can also be described as a more stable algorithm than the decision trees – if we modify the data a little bit, decision trees will change, most likely reducing the accuracy. This will not happen in the random forest algorithms – since it is the combination of many decision trees, the random forest will remain stable.

6 **Logistic Regression:** Logistic regression is a machine learning algorithm used for classification problems. That is, it can be used to predict whether an instance belongs to one class or the other. For example, it could be used to predict whether a person is male or female, based on their height, weight, and other features. It is a supervised learning algorithm that can be used to predict the probability of occurrence of an event. Logistic regression model learns the relationship between the features and the classes. The logistic regression algorithm is used to map the input data to a probability, unlike linear regression which is used to map the input data to continuous output values. Logistic regression models are used to predict the probability of an event occurring, such as whether a customer will purchase a product. The output of the logistic regression model is a probability value between 0 and 1. The output represents the probability that the class of the input data is 1.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

The input data is mapped to a probability using the sigmoid function. The sigmoid function, also called as logistic function, is a mathematical function that maps values (sum of weighted input) from -infinity to +infinity to values between 0 and 1. The sigmoid function that represents the hypothesis is defined as shown above.

$$z = \theta^T x$$

Further, 6 the value of z in sigmoid function represents the weighted sum of input values and can be written as above.

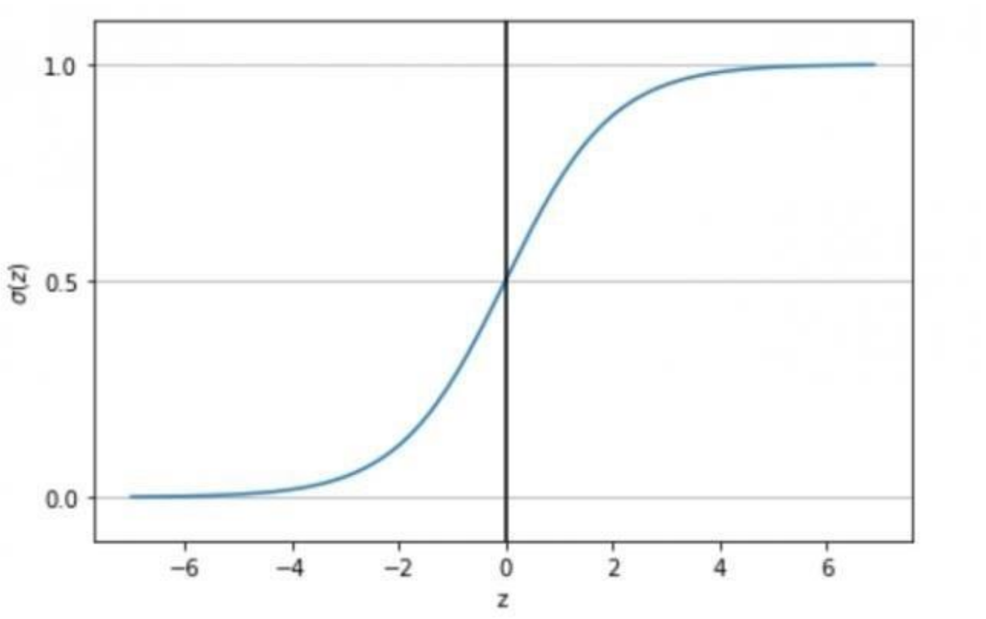


Fig 5.8 Sigmoid curve

The following plot is created when the sigmoid function, σ(z) is plotted against the net input function output, z. Note that the value of sigmoid function ranges between 0 and 1.

In the above plot, the σ(z) approaches 1 when z approaches infinity. Similarly, σ(z) approaches 0 when z approaches negative of infinity. Thus, it can be concluded that the value of σ(z) ranges from 0 to 1. At z = 0, σ(z) takes the value of 0.5.

KNN: K-Nearest Neighbour (KNN) is one of the simplest, though, accurate machine learning algorithms. KNN is a non-parametric algorithm, meaning that it does not make any assumptions about the data structure. In real world problems, data rarely obeys the general theoretical assumptions, making non-parametric algorithms a good solution for such problems. KNN model representation is as simple as the dataset – there is no learning required, the entire training set is stored.

KNN can be used for both classification and regression problems. In both problems, the prediction is based on the k training instances that are closest to the input instance. In the KNN classification problem, the output would be a class, to which the input instance belongs, predicted by the majority vote of the k closest neighbour. In the regression problem, the output would be the property value, which is generally a mean value of the k nearest neighbour. The schematic example is outlined in Figure.

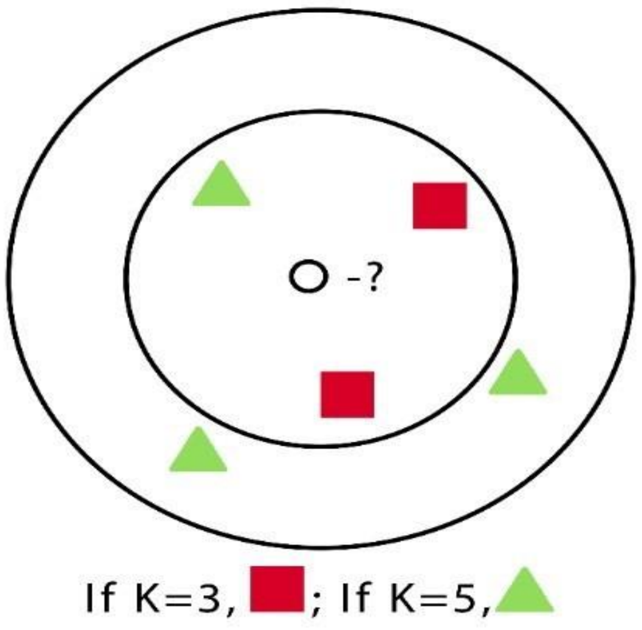


Fig 5.9 KNN model

The following are the lists of distances which are considered in a KNN algorithm.

$$Hamming\ Distance:\ d_{ij} = \sum_{k=1}^{p} |x_{ik} - x_{jk}|$$

$$Manhattan\ Distance:\ d_1(p,q) = ||p-q||_1 = \sum_{i=1}^{n} |p_i - q_i|$$

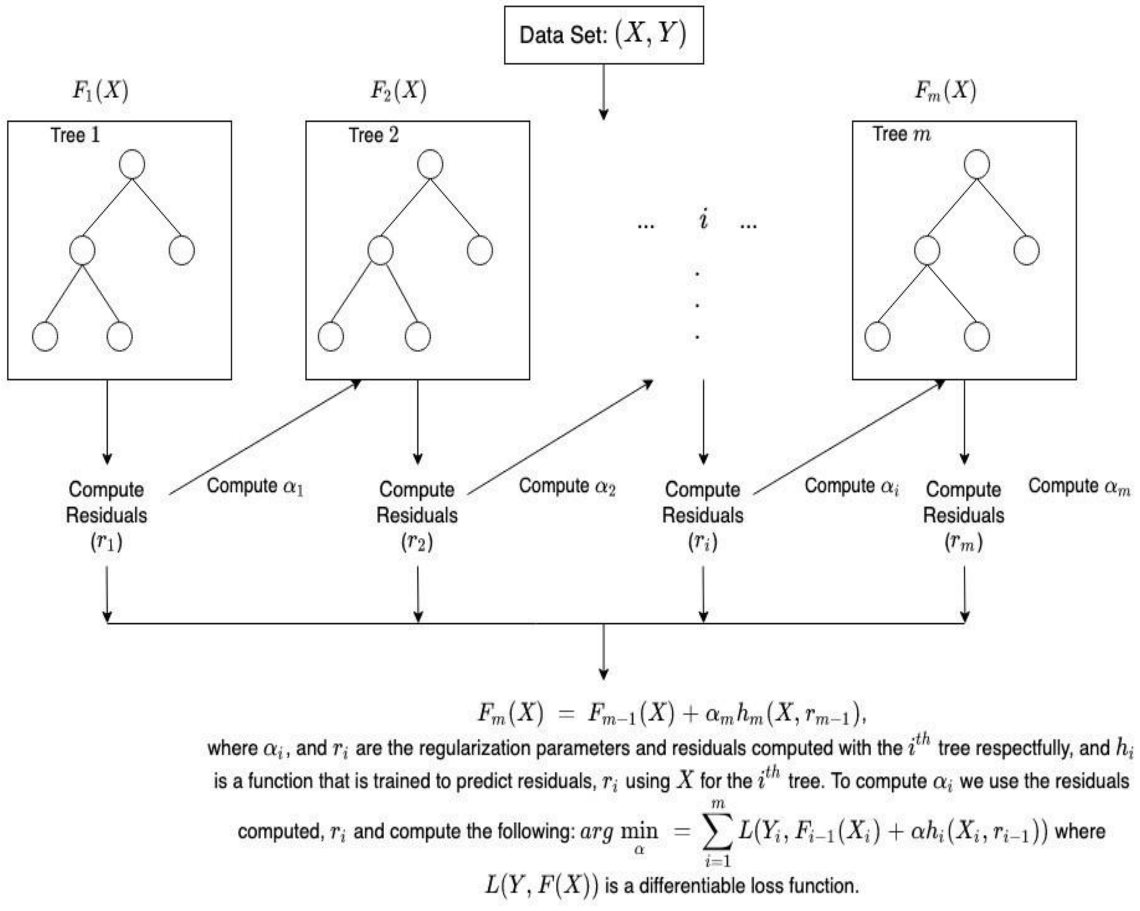
$$Minkowski\ Distance = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

$$EuclidianDistance = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2 \ ;\ p\ and\ q\ are\ the\ points\ in\ n - space}$$

Euclidian distance is good for the problems, where the features are of the same type. For the features of different types, it is advised to use, for example, Manhattan Distance.

XGboost: XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Fig 5.10 XGboost system

5.2.2 DL Implementation:

Multi-Layer Perceptron is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network, we combine neurons together so that the outputs of some neurons are inputs of other neurons.

n the multi-layer perceptron diagram above, we can see that there are three inputs and thus three input nodes and the hidden layer has three nodes. The output layer gives two outputs, therefore there are two output nodes. The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passes it to the output layer. Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.

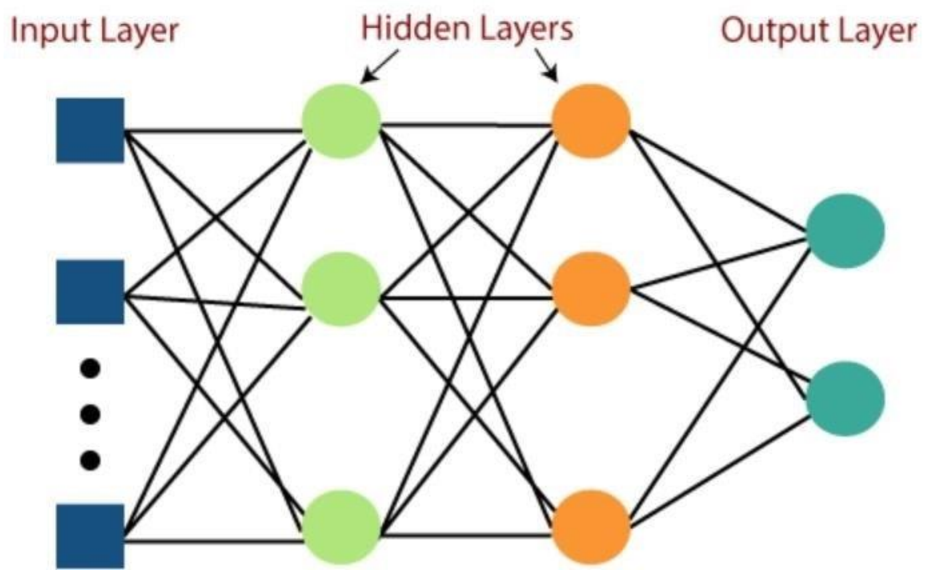


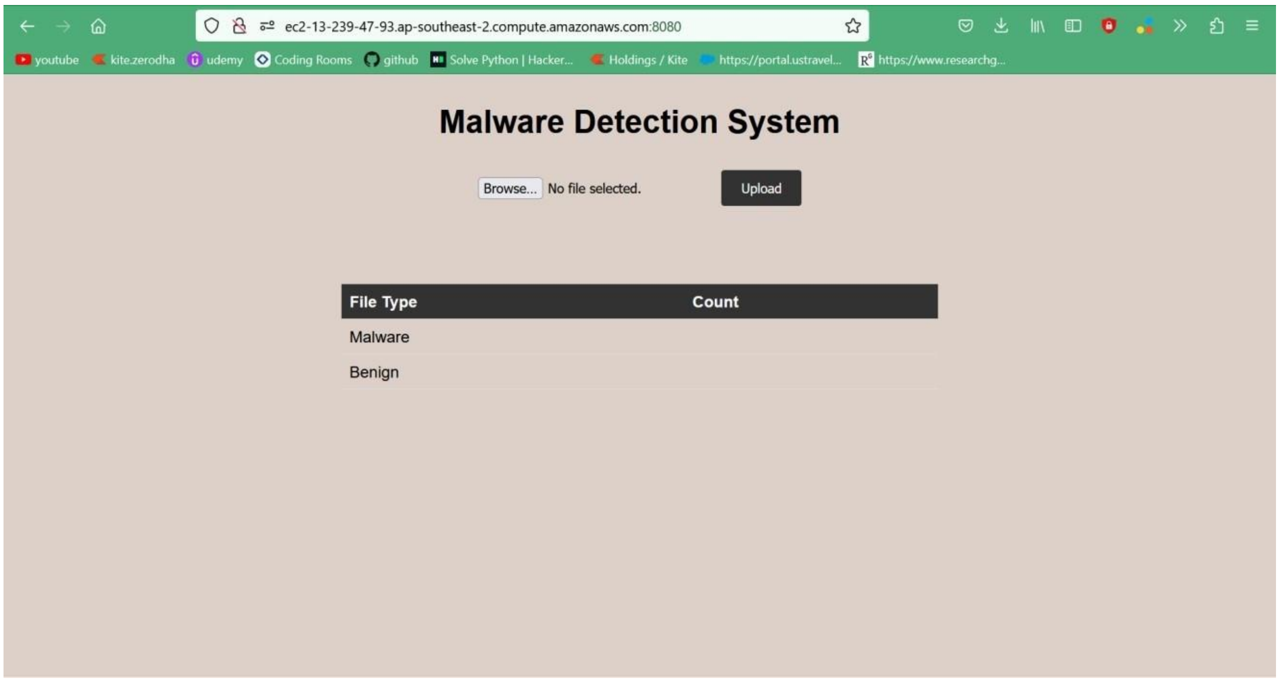
Fig 5.11 DL NLP

5.3 Output and Interface Design:

We have deployed out ML model on web-based API. This project is hosted on Amazon web servers which a free hosting service. We have use fast api and on server side we used PKL file format. On the client side, the user gets the result as total number of malware files with the help of Ajax. We used HTML and CSS to make this site. The site is kept simple and user friendly. Among all ML algorithms, we have used Xgboost as it has maximum accuracies (will be discussed in later part).

5.3.1 Samples of Interface:

Below is the photo of web API.

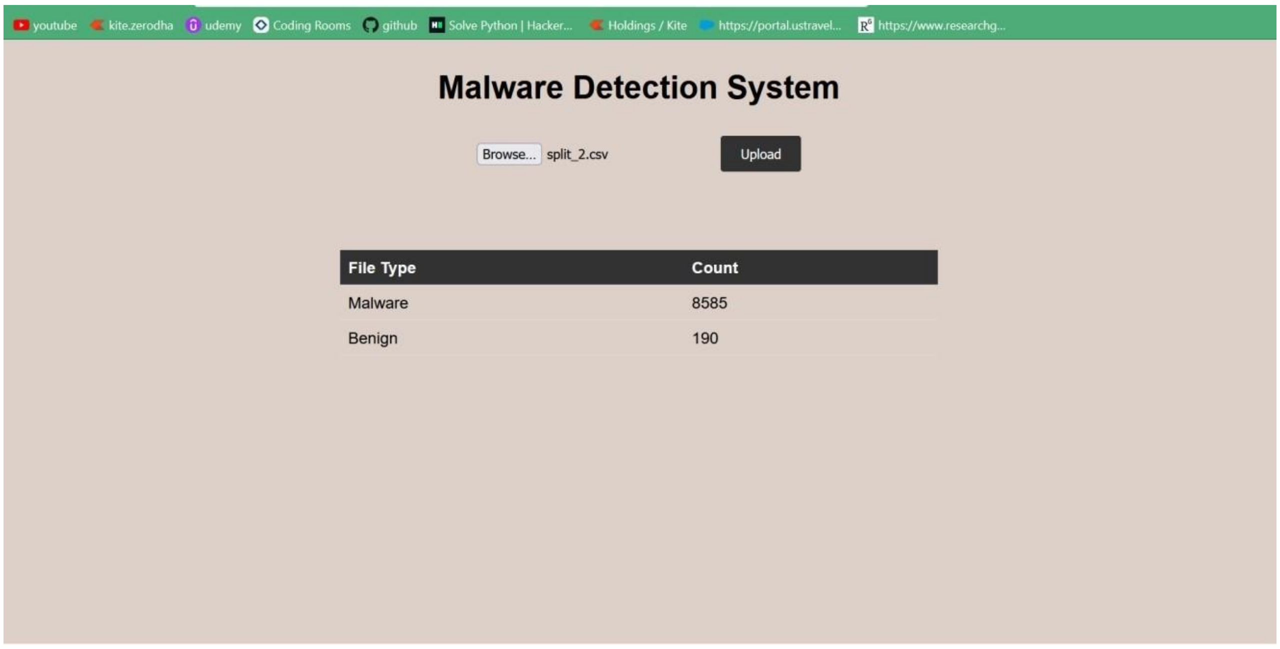


Malware Detection System

File Type	Count
Malware	
Benign	

Fig 5.12 Web dashboard

On clicking browse user is prompted to select file option



Malware Detection System

File Type	Count
Malware	8585
Benign	190

The output is in the form of malware and benign file number which is generated automatically by clicking upload button.

Web link: http://ec2-13-239-47-93.ap-southeast-2.compute.amazonaws.com:8080/

CHAPTER 6. IMPLEMENTATION

6.1 Implementation Platform:

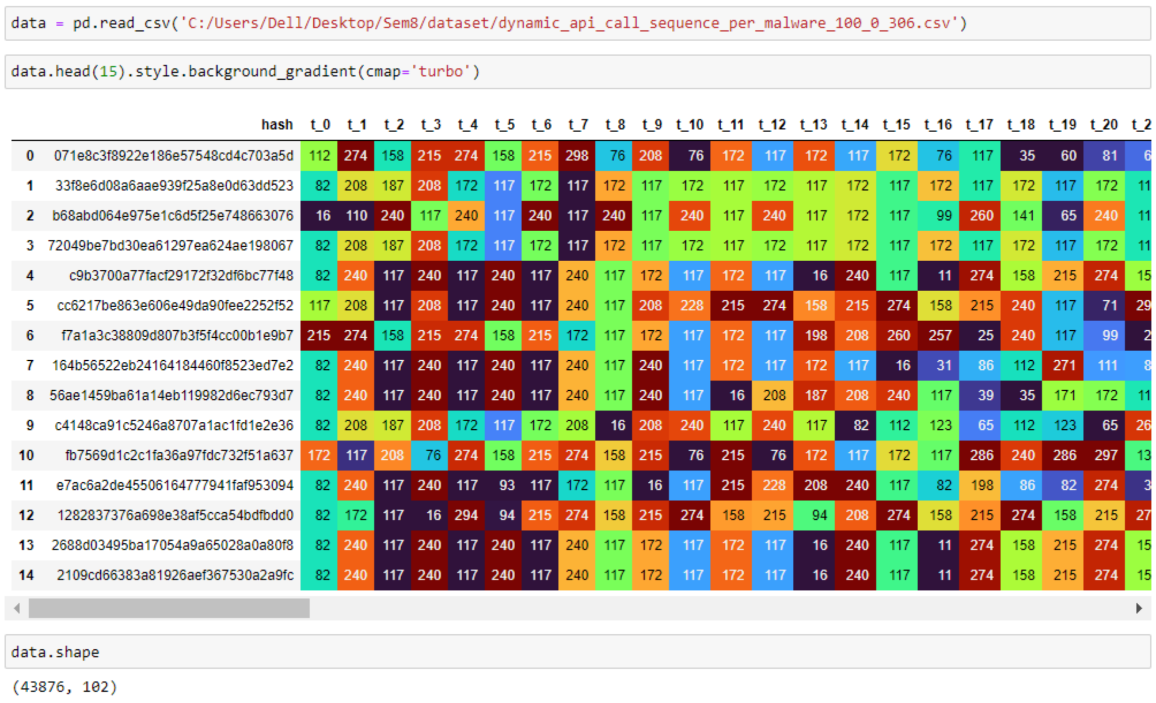
Implementation platform for the project was Windows as well as Linux. We window to run python code and used jupyter notebook as IDE, further we used Linux to implement cuckoo sandbox and to generate log files, Ram size of the system is 8 GB SSD is 256 GB, HHD is 512 GB. Further Laptop used was core I5 8th gen. As the project has no high-end requirement for graphics, we have used inbuilt NVidia graphic card. The web application is started by using Python interpreter and was tested on Firefox browser on the Linux and Windows OS along with chrome browser present on the Android device.

6.2 Module Specifications:

1. uvicorn~=0.21.1
2. pandas~=1.5.3
3. fastapi~=0.95.0
4. asyncio
5. nest_asyncio
6. shutils
7. scikit-learn~=0.24.2
8. xgboost~=1.4.2
9. seaborn~=0.11.1
10. numpy~=1.19.5
11. matplotlib~=3.4.2
12. scikit-plot~=0.3.7
13. plotly~=5.1.0
14. tensorflow~=2.5.0
15. Jinja2~=3.0.1
16. python-multipart~=0.0.56.3

6.3 Program Code Snapshots:

6.3.1 Dataset exploration



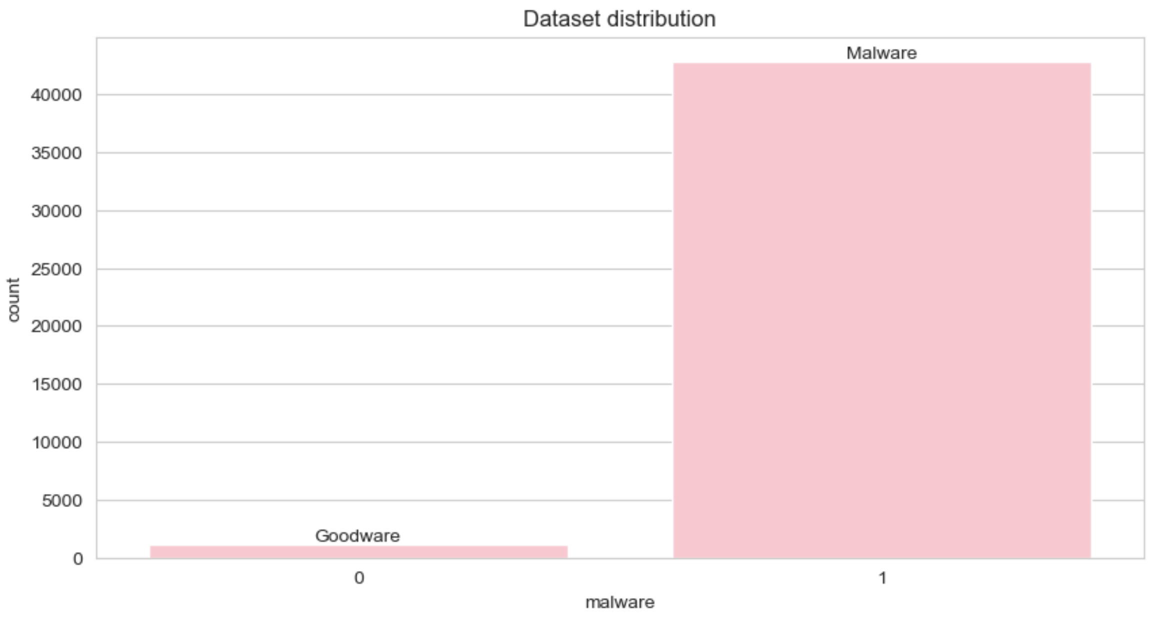
```
data = pd.read_csv('C:/Users/Dell/Desktop/Sem8/dataset/dynamic_api_call_sequence_per_malware_100_0_306.csv')
```

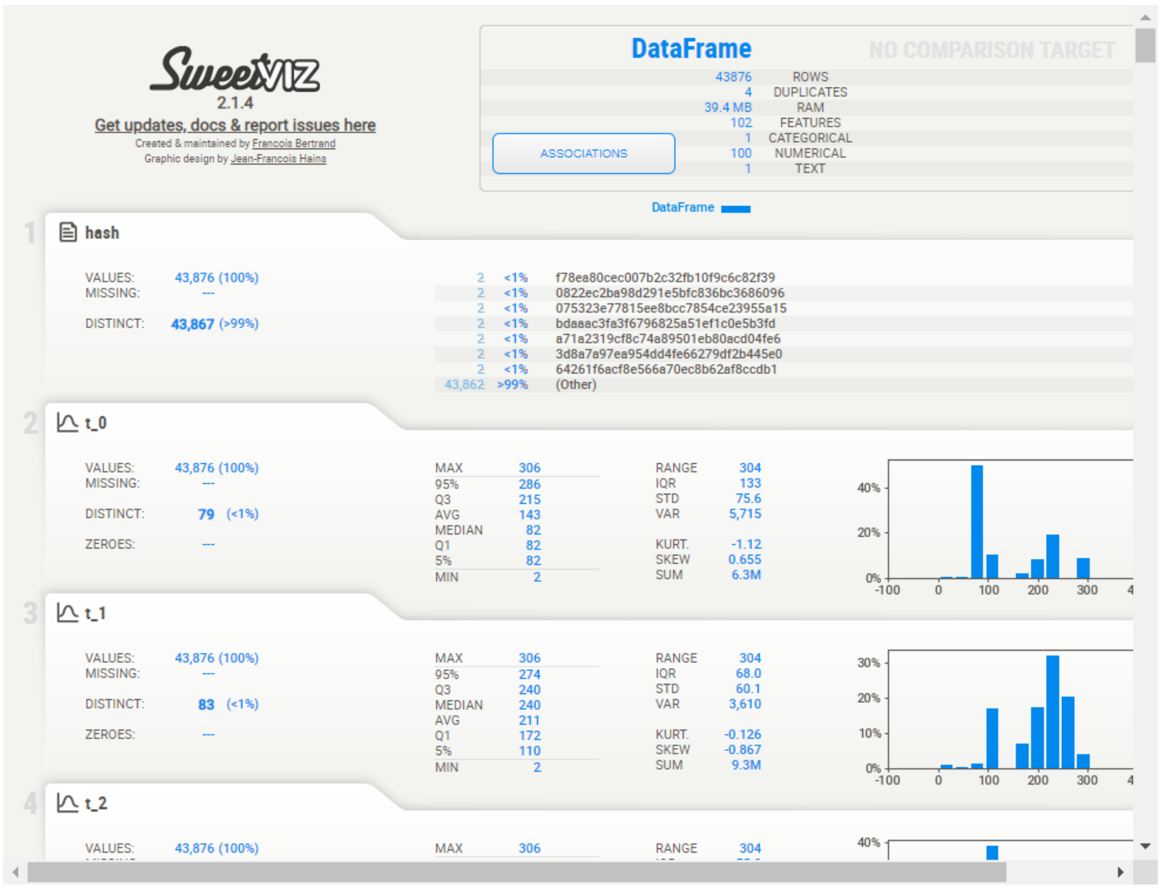
```
data.head(15).style.background_gradient(cmap='turbo')
```

	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_10	t_11	t_12	t_13	t_14	t_15	t_16	t_17	t_18	t_19	t_20	t_2
0	071e8c3f8922e186e57548cd4c703a5d	112	274	158	215	274	158	215	298	76	208	76	172	117	172	117	172	76	117	35	60	81	6
1	33f8e6d08a6aae939f25a8e0d63dd523	82	208	187	208	172	117	172	117	172	117	172	117	172	117	172	117	172	117	172	117	172	11
2	b68abd064e975e1c6d5f25e748663076	16	110	240	117	240	117	240	117	240	117	240	117	240	117	172	117	99	260	141	65	240	11
3	72049be7bd30ea61297ea624ae198067	82	208	187	208	172	117	172	117	172	117	172	117	172	117	172	117	172	117	172	117	172	11
4	c9b3700a77facf29172f32df6bc77f48	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274	158	215	274	15
5	cc6217be863e606e49da90fee2252f52	117	208	117	208	117	240	117	240	117	208	228	215	274	158	215	274	158	215	240	117	71	29
6	f7a1a3c38809d807b3f5f4cc00b1e9b7	215	274	158	215	274	158	215	172	117	172	117	172	117	198	208	260	257	25	240	117	99	2
7	164b56522eb24164184460f8523ed7e2	82	240	117	240	117	240	117	240	117	240	117	172	117	172	117	16	31	86	112	271	111	8
8	56ae1459ba61a14eb119982d6ec793d7	82	240	117	240	117	240	117	240	117	240	117	16	208	187	208	240	117	39	35	171	172	11
9	c4148ca91c5246a8707a1ac1fd1e2e36	82	208	187	208	172	117	172	208	16	208	240	117	240	117	82	112	123	65	112	123	65	26
10	fb7569d1c2c1fa36a97fdc732f51a637	172	117	208	76	274	158	215	274	158	215	76	215	76	172	117	172	117	286	240	286	297	13
11	e7ac6a2de45506164777941faf953094	82	240	117	240	117	93	117	172	117	16	117	215	228	208	240	117	82	198	86	82	274	3
12	1282837376a698e38af5cca54bdfbdd0	82	172	117	16	294	94	215	274	158	215	274	158	215	94	208	274	158	215	274	158	215	27
13	2688d03495ba17054a9a65028a0a80f8	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274	158	215	274	15
14	2109cd66383a81926aef367530a2a9fc	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274	158	215	274	15

```
data.shape
```

```
(43876, 102)
```




Sweetviz
2.1.4
Get updates, docs & report issues here
Created & maintained by Francois Bertrand
Graphic design by Jean-Francois Hains
ASSOCIATIONS
NO COMPARISON TARGET

DataFrame

43876	ROWS
4	DUPLICATES
39.4 MB	RAM
102	FEATURES
1	CATEGORICAL
100	NUMERICAL
1	TEXT

DataFrame

1 hash

VALUES:	43,876 (100%)
MISSING:	---
DISTINCT:	43,867 (>99%)

2	<1%	f78ea80cec007b2c32fb10f9c6c82f39
2	<1%	0822ec2ba98d291e5bfc836bc3686096
2	<1%	075323e77815ee8bcc7854ce23955a15
2	<1%	bdaaac3fa3f6796825a51ef1c0e5b3fd
2	<1%	a71a2319cf8c74a89501eb80acd04fe6
2	<1%	3d8a7a97ea954dd4fe66279df2b445e0
2	<1%	64261f6acf8e566a70ec8b62af8ccdb1
43,862	>99%	(Other)

2 t_0

VALUES:	43,876 (100%)	MAX	306	RANGE	304
MISSING:	---	95%	286	IQR	133
DISTINCT:	79 (<1%)	Q3	215	STD	75.6
ZEROES:	---	AVG	143	VAR	5,715
		MEDIAN	82	KURT.	-1.12
		Q1	82	SKEW	0.655
		5%	82	SUM	6.3M
		MIN	2		

3 t_1

VALUES:	43,876 (100%)	MAX	306	RANGE	304
MISSING:	---	95%	274	IQR	68.0
DISTINCT:	83 (<1%)	Q3	240	STD	60.1
ZEROES:	---	MEDIAN	240	VAR	3,610
		AVG	211	KURT.	-0.126
		Q1	172	SKEW	-0.867
		5%	110	SUM	9.3M
		MIN	2		

4 t_2

VALUES:	43,876 (100%)	MAX	306	RANGE	304

6.3.2 Traditional ML Model Building

<u>Train, Test, K-fold</u>:

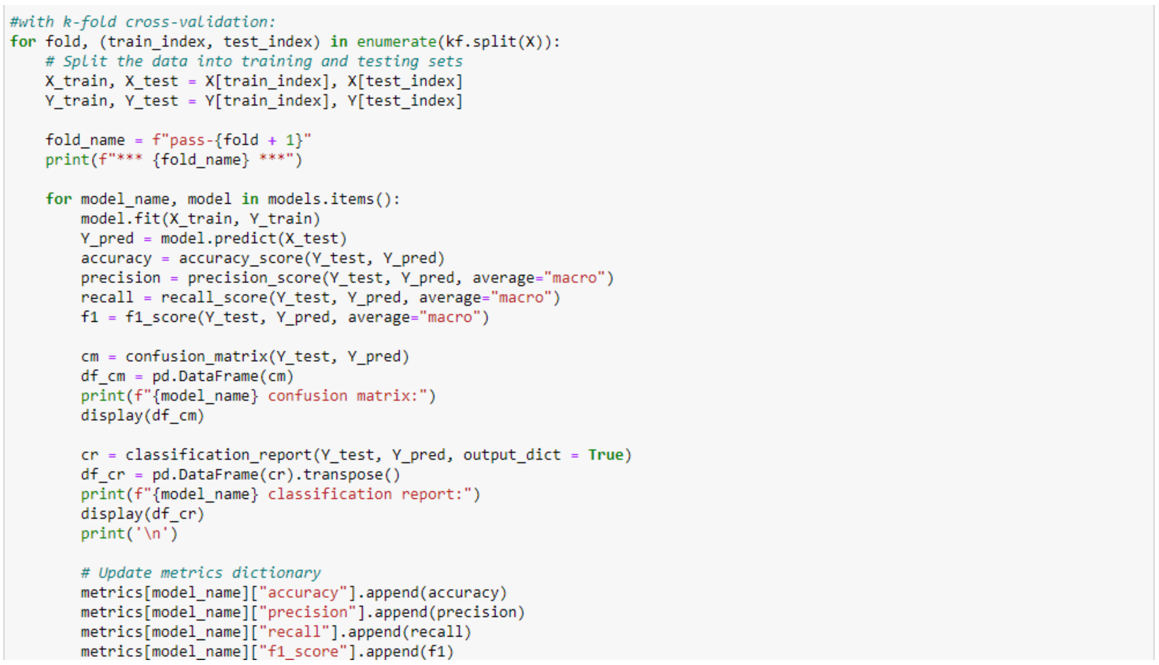
```
#dropping unwanted features
data1 = data.drop('hash', axis = 1)
```

```
#splitting data into training and testing set

X = data1.drop('malware', axis = 1).values
Y = data1['malware'].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 56)
```

```
#normalizing the dataset
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

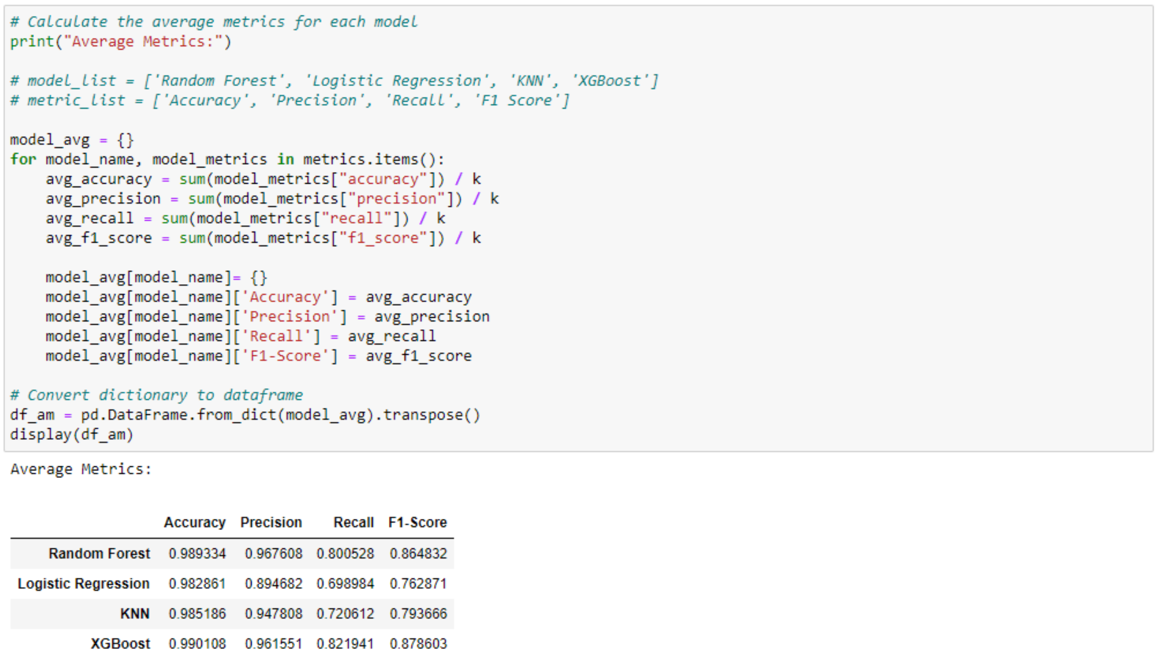
```
#Buid models and evaluate accuracy, confusion matrix
models = {'Random Forest': RandomForestClassifier(),
          'Logistic Regression': LogisticRegression(),
          'KNN': KNeighborsClassifier(n_neighbors=5),
          'XGBoost': xgb.XGBClassifier()
         }

#define number of folds
k = 5

#Create the cross-validation object
kf = KFold(n_splits = k)

metrics = {model_name: {"accuracy": [], "precision": [], "recall": [], "f1_score": []} for model_name in models}
```

Model fitting:



```
#with k-fold cross-validation:
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    # Split the data into training and testing sets
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    fold_name = f"pass-{fold + 1}"
    print(f"*** {fold_name} ***")

    for model_name, model in models.items():
        model.fit(X_train, Y_train)
        Y_pred = model.predict(X_test)
        accuracy = accuracy_score(Y_test, Y_pred)
        precision = precision_score(Y_test, Y_pred, average="macro")
        recall = recall_score(Y_test, Y_pred, average="macro")
        f1 = f1_score(Y_test, Y_pred, average="macro")

        cm = confusion_matrix(Y_test, Y_pred)
        df_cm = pd.DataFrame(cm)
        print(f"{model_name} confusion matrix:")
        display(df_cm)

        cr = classification_report(Y_test, Y_pred, output_dict = True)
        df_cr = pd.DataFrame(cr).transpose()
        print(f"{model_name} classification report:")
        display(df_cr)
        print('\n')

        # Update metrics dictionary
        metrics[model_name]["accuracy"].append(accuracy)
        metrics[model_name]["precision"].append(precision)
        metrics[model_name]["recall"].append(recall)
        metrics[model_name]["f1_score"].append(f1)
```

Model Evaluation:

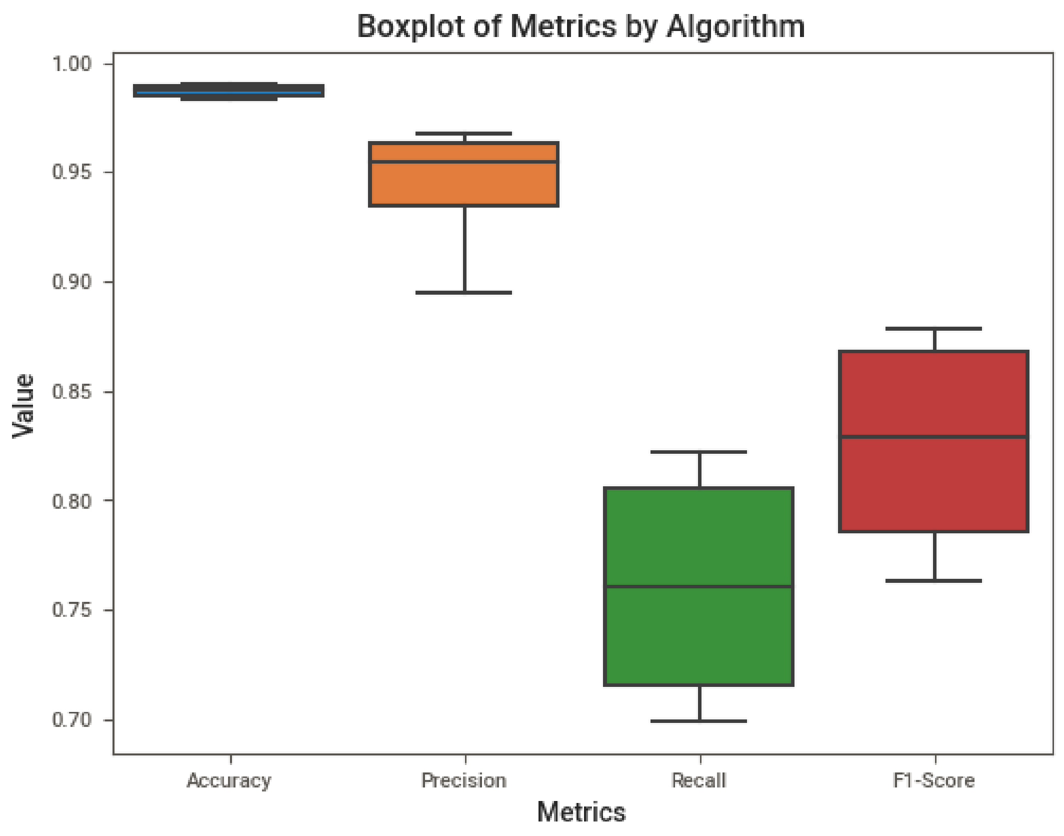


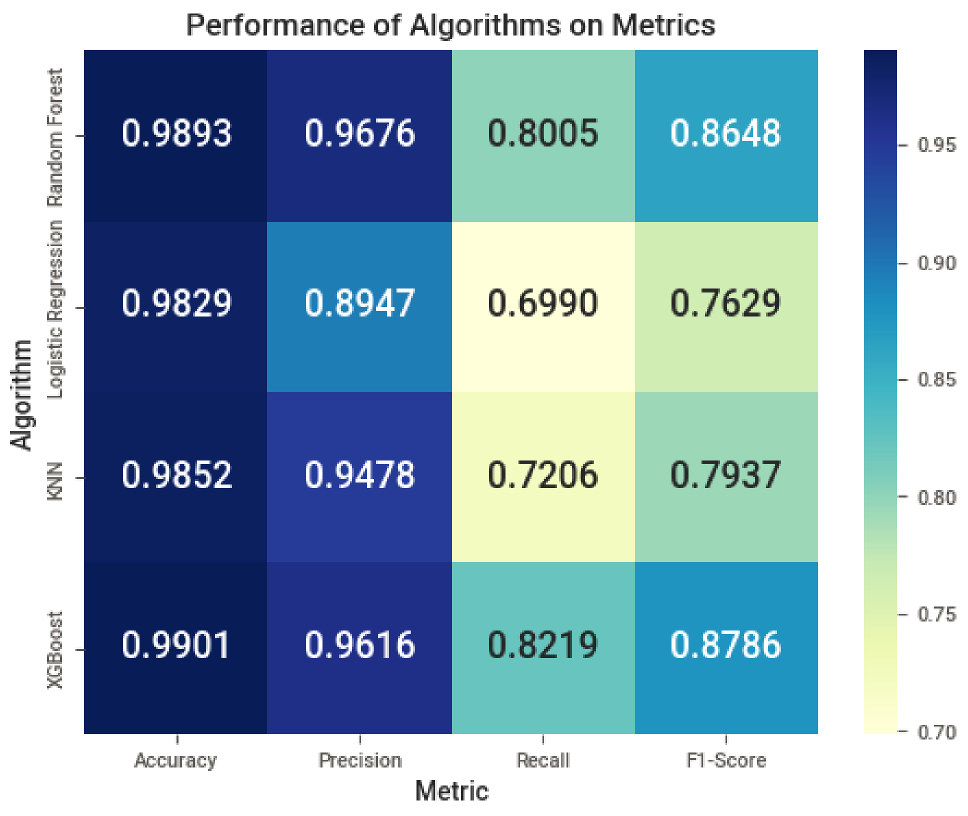
```
# Calculate the average metrics for each model
print("Average Metrics:")

# model_list = ['Random Forest', 'Logistic Regression', 'KNN', 'XGBoost']
# metric_list = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

model_avg = {}
for model_name, model_metrics in metrics.items():
    avg_accuracy = sum(model_metrics["accuracy"]) / k
    avg_precision = sum(model_metrics["precision"]) / k
    avg_recall = sum(model_metrics["recall"]) / k
    avg_f1_score = sum(model_metrics["f1_score"]) / k

    model_avg[model_name]= {}
    model_avg[model_name]['Accuracy'] = avg_accuracy
    model_avg[model_name]['Precision'] = avg_precision
    model_avg[model_name]['Recall'] = avg_recall
    model_avg[model_name]['F1-Score'] = avg_f1_score

# Convert dictionary to dataframe
df_am = pd.DataFrame.from_dict(model_avg).transpose()
display(df_am)
```

```
Average Metrics:
```

	Accuracy	Precision	Recall	F1-Score
Random Forest	0.989334	0.967608	0.800528	0.864832
Logistic Regression	0.982861	0.894682	0.698984	0.762871
KNN	0.985186	0.947808	0.720612	0.793666
XGBoost	0.990108	0.961551	0.821941	0.878603

Model Comparisons:





<u>Saving model to pickle file for future use</u>:

```
classifier = models['XGBoost']
```

```
# Creating pickle file using serialization
import pickle
pickle_out = open("classifier.pkl","wb")
pickle.dump(classifier, pickle_out)
pickle_out.close()
```

6.3.3 DL Model Building

<u>Train, test and normalizing</u>:

```
# Read Data
data = pd.read_csv('C:/Users/Dell/Desktop/Sem8/dataset/dynamic_api_call_sequence_per_malware_100_0_306.csv')
```

```
# dropping unwanted column
data2 = data.drop('hash', axis = 1)
```

```
#splitting data into training and testing set

X = data2.drop('malware', axis = 1).values
Y = data2['malware'].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 56)
```

```
#normalizing the dataset
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

<u>Model building and development</u>:

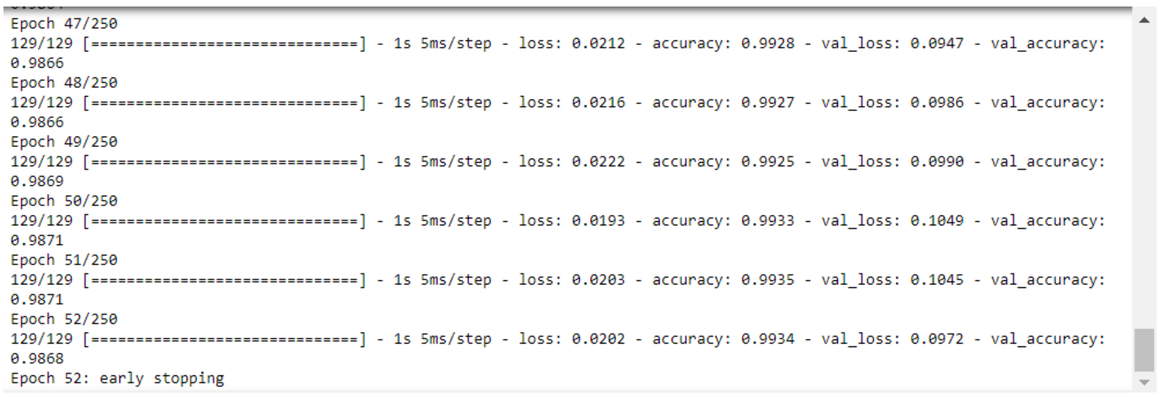
```
# build model
model = Sequential()
```

```
# Dense layer = each neuron is connected to every neuron in the previous layer
#100 units = 100 neurons
model.add(Dense(100, activation = 'relu'))
# 50% neurons dropped from previous layer to avoid dependency
model.add(Dropout(0.5))

model.add(Dense(50, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(25, activation = 'relu'))
model.add(Dropout(0.5))

# sigmoid specialized for binary classification, units = 1 means single output
model.add(Dense(units = 1, activation = 'sigmoid'))

# configure learning process
# optimize binary_crossentropy loss funtion(o/p b/w o and 1) with adam optimizer
#optimize loss fn by improving accuracy
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
# stop training the model if the validation loss doesn't improve by 0.001 after 30 epochs
# prevent overfitting
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 30)
```

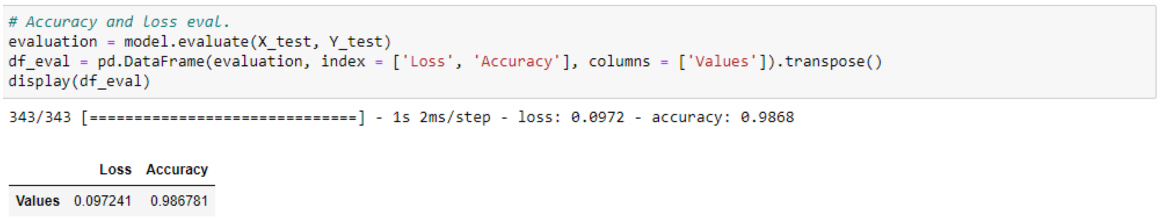
```
# x: input features, y: target variables
# epochs: training dataset iterations
# bach_size: number of samples used in each iterations
# validation_data: tuple which evaluates model on different set of data
#callbacks: set of callbacks applied during training
# history: variable that'll be used to monitor training and validation loss over time
history = model.fit(x = X_train, y = Y_train, epochs = 250, batch_size = 256, validation_data = (X_test, Y_test), callbacks = [ea
```

Early stopping:



```
Epoch 47/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0212 - accuracy: 0.9928 - val_loss: 0.0947 - val_accuracy:
0.9866
Epoch 48/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0216 - accuracy: 0.9927 - val_loss: 0.0986 - val_accuracy:
0.9866
Epoch 49/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0222 - accuracy: 0.9925 - val_loss: 0.0990 - val_accuracy:
0.9869
Epoch 50/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0193 - accuracy: 0.9933 - val_loss: 0.1049 - val_accuracy:
0.9871
Epoch 51/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0203 - accuracy: 0.9935 - val_loss: 0.1045 - val_accuracy:
0.9871
Epoch 52/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0202 - accuracy: 0.9934 - val_loss: 0.0972 - val_accuracy:
0.9868
Epoch 52: early stopping
```

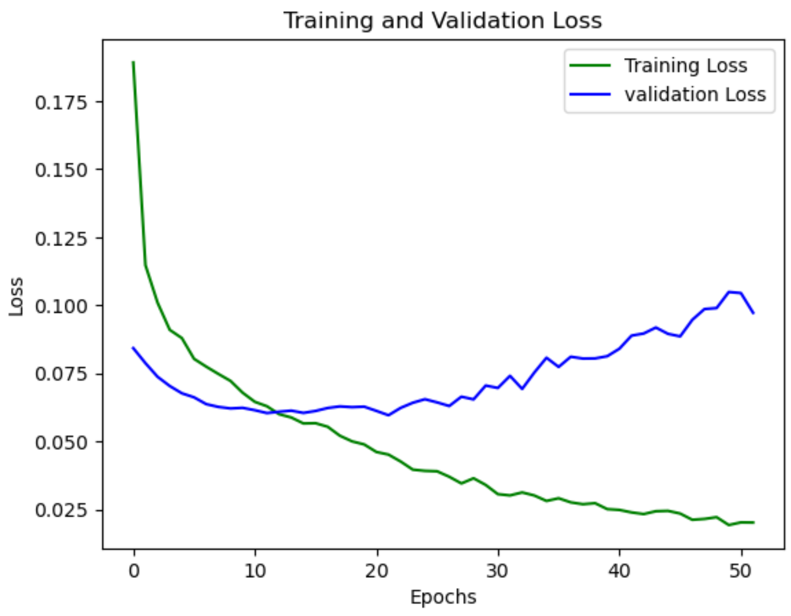
DL Model Evaluation:

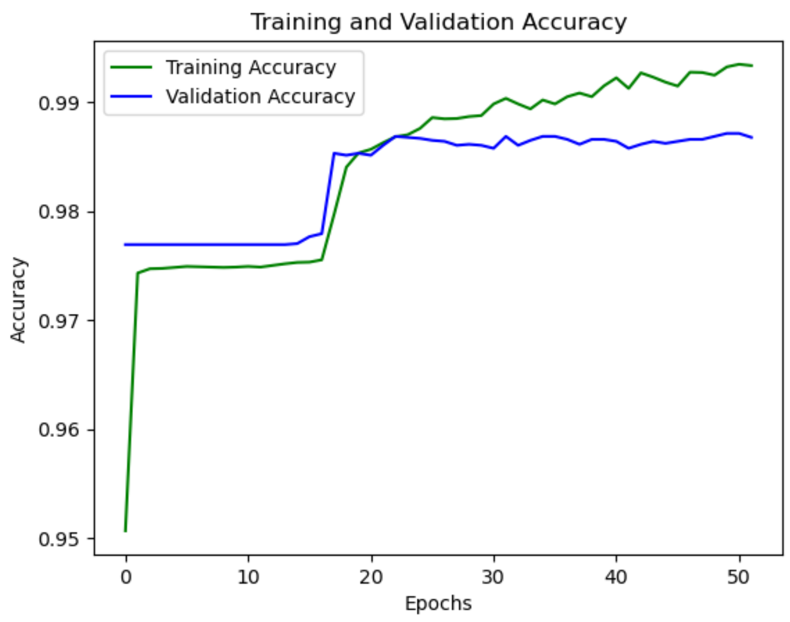


```
# Accuracy and loss eval.
evaluation = model.evaluate(X_test, Y_test)
df_eval = pd.DataFrame(evaluation, index = ['Loss', 'Accuracy'], columns = ['Values']).transpose()
display(df_eval)
```

```
343/343 [==============================] - 1s 2ms/step - loss: 0.0972 - accuracy: 0.9868
```

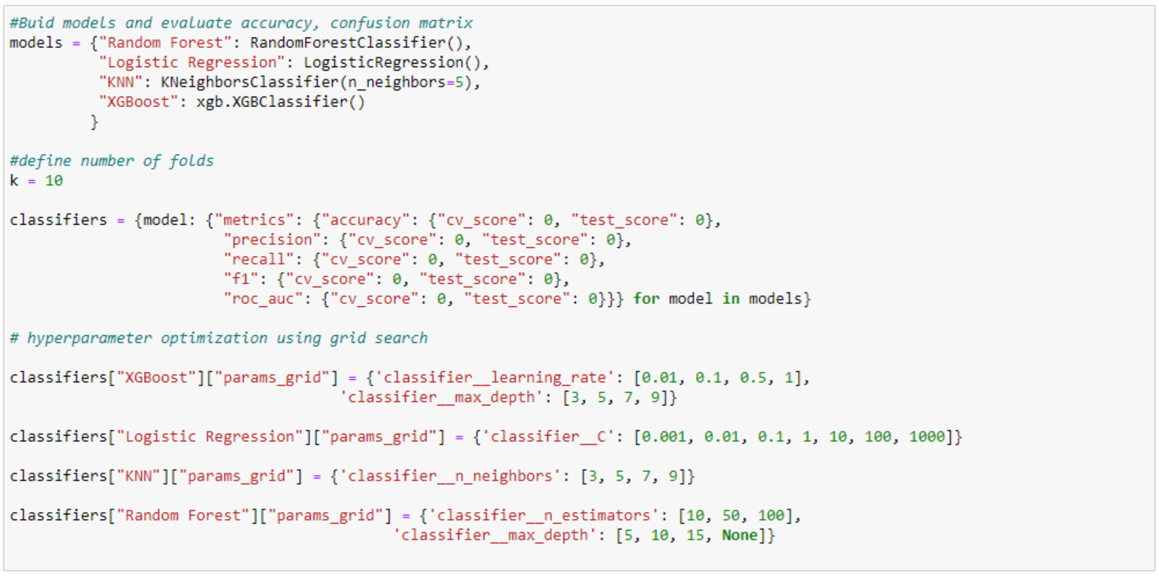
	Loss	Accuracy
Values	0.097241	0.986781





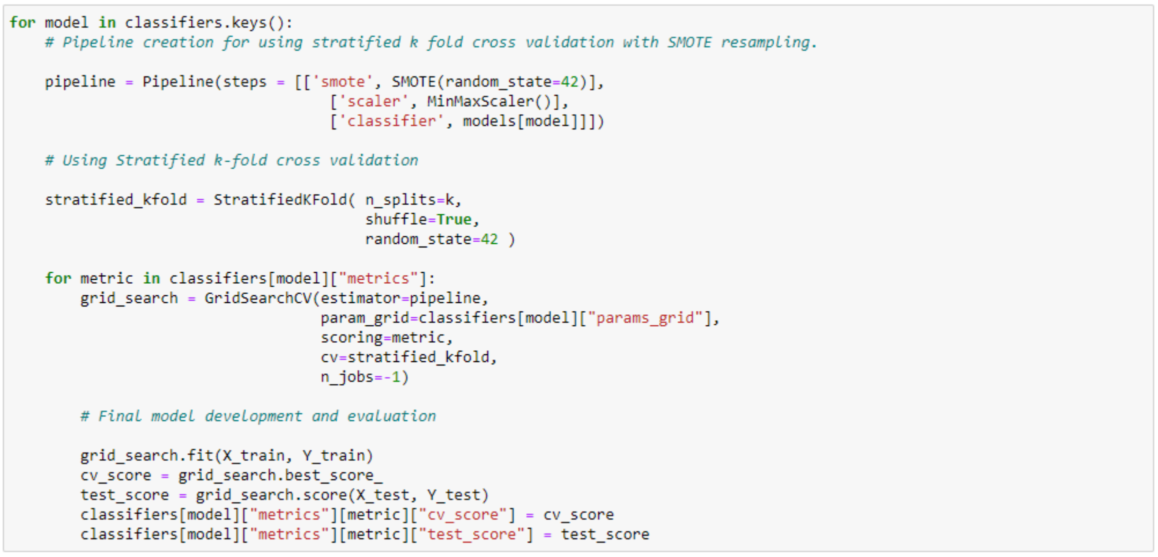
6.3.4 Improvisations and optimizations:

Hyperparameter optimisations:



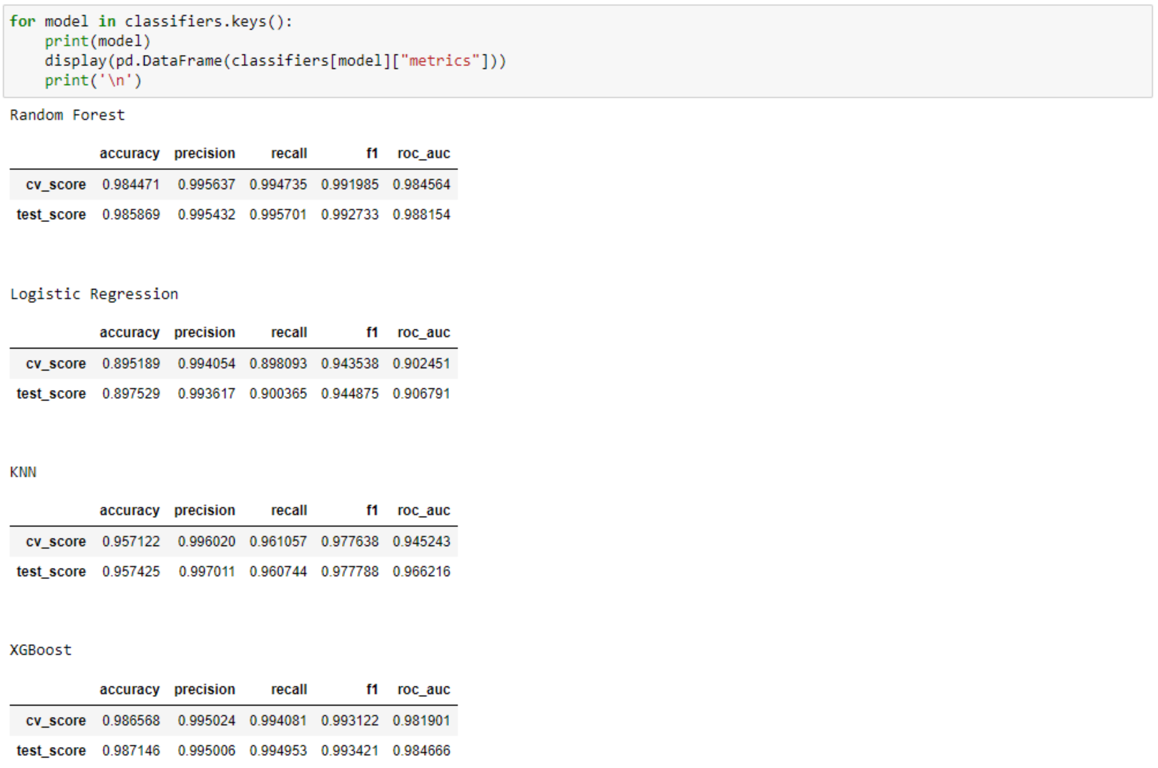
```
#Buid models and evaluate accuracy, confusion matrix
models = {"Random Forest": RandomForestClassifier(),
          "Logistic Regression": LogisticRegression(),
          "KNN": KNeighborsClassifier(n_neighbors=5),
          "XGBoost": xgb.XGBClassifier()
         }

#define number of folds
k = 10

classifiers = {model: {"metrics": {"accuracy": {"cv_score": 0, "test_score": 0},
                        "precision": {"cv_score": 0, "test_score": 0},
                        "recall": {"cv_score": 0, "test_score": 0},
                        "f1": {"cv_score": 0, "test_score": 0},
                        "roc_auc": {"cv_score": 0, "test_score": 0}}} for model in models}

# hyperparameter optimization using grid search

classifiers["XGBoost"]["params_grid"] = {'classifier__learning_rate': [0.01, 0.1, 0.5, 1],
                                      'classifier__max_depth': [3, 5, 7, 9]}

classifiers["Logistic Regression"]["params_grid"] = {'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

classifiers["KNN"]["params_grid"] = {'classifier__n_neighbors': [3, 5, 7, 9]}

classifiers["Random Forest"]["params_grid"] = {'classifier__n_estimators': [10, 50, 100],
                                            'classifier__max_depth': [5, 10, 15, None]}
```

Pipeline development for SMOTE implementation and model fitting:



```
for model in classifiers.keys():
    # Pipeline creation for using stratified k fold cross validation with SMOTE resampling.

    pipeline = Pipeline(steps = [['smote', SMOTE(random_state=42)],
                                ['scaler', MinMaxScaler()],
                                ['classifier', models[model]]])

    # Using Stratified k-fold cross validation

    stratified_kfold = StratifiedKFold( n_splits=k,
                                        shuffle=True,
                                        random_state=42 )

    for metric in classifiers[model]["metrics"]:
        grid_search = GridSearchCV(estimator=pipeline,
                                   param_grid=classifiers[model]["params_grid"],
                                   scoring=metric,
                                   cv=stratified_kfold,
                                   n_jobs=-1)

        # Final model development and evaluation

        grid_search.fit(X_train, Y_train)
        cv_score = grid_search.best_score_
        test_score = grid_search.score(X_test, Y_test)
        classifiers[model]["metrics"][metric]["cv_score"] = cv_score
        classifiers[model]["metrics"][metric]["test_score"] = test_score
```

Metrics evaluation after optimizations:



```
for model in classifiers.keys():
    print(model)
    display(pd.DataFrame(classifiers[model]["metrics"]))
    print('\n')
```

```
Random Forest
```

	accuracy	precision	recall	f1	roc_auc
cv_score	0.984471	0.995637	0.994735	0.991985	0.984564
test_score	0.985869	0.995432	0.995701	0.992733	0.988154

```
Logistic Regression
```

	accuracy	precision	recall	f1	roc_auc
cv_score	0.895189	0.994054	0.898093	0.943538	0.902451
test_score	0.897529	0.993617	0.900365	0.944875	0.906791

```
KNN
```

	accuracy	precision	recall	f1	roc_auc
cv_score	0.957122	0.996020	0.961057	0.977638	0.945243
test_score	0.957425	0.997011	0.960744	0.977788	0.966216

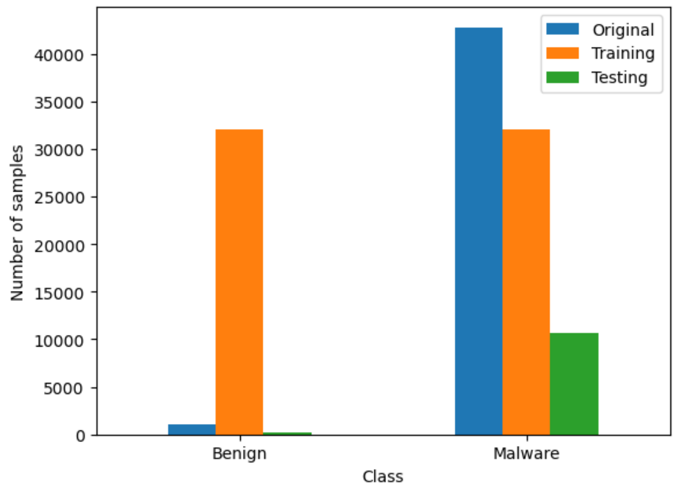
```
XGBoost
```

	accuracy	precision	recall	f1	roc_auc
cv_score	0.986568	0.995024	0.994081	0.993122	0.981901
test_score	0.987146	0.995006	0.994953	0.993421	0.984666

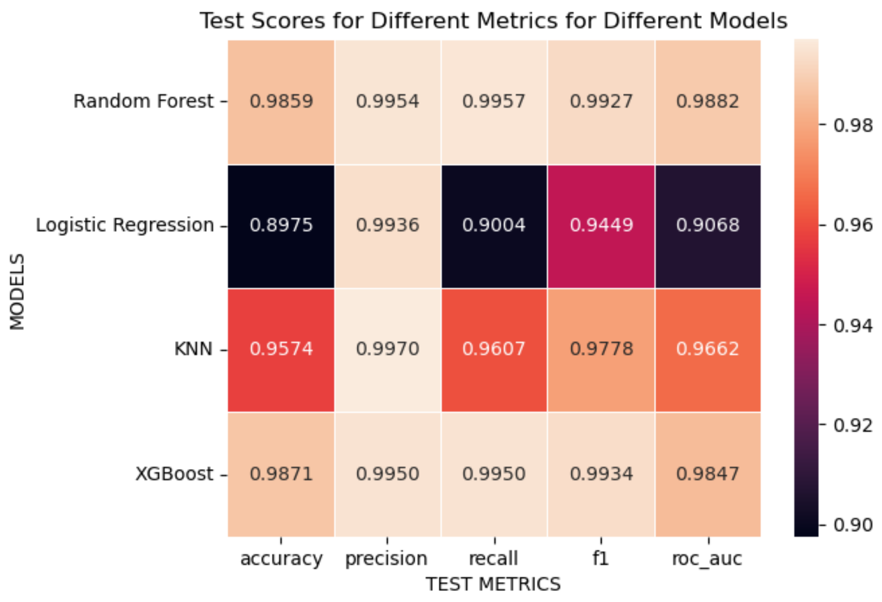
Balanced dataset after SMOTE resampling:

```
# Count the number of samples in each class in the original dataset, training dataset, and testing dataset
counts = pd.DataFrame({'Original': np.bincount(Y),
                       'Training': np.bincount(pipeline.named_steps['smote'].fit_resample(X_train, Y_train)[1]),
                       'Testing': np.bincount(Y_test)},
                      index=['Benign', 'Malware'])

# Plot the bar graph
ax = counts.plot(kind='bar', rot=0)
ax.set_xlabel('Class')
ax.set_ylabel('Number of samples')
display(counts)
plt.show()
```

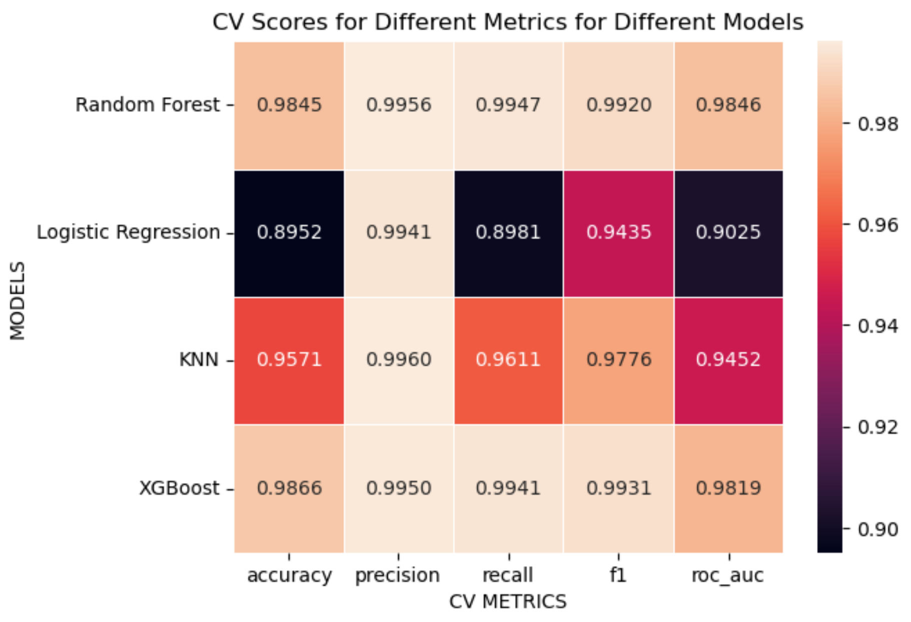
	Original	Training	Testing
Benign	1079	32098	270
Malware	42797	32098	10699



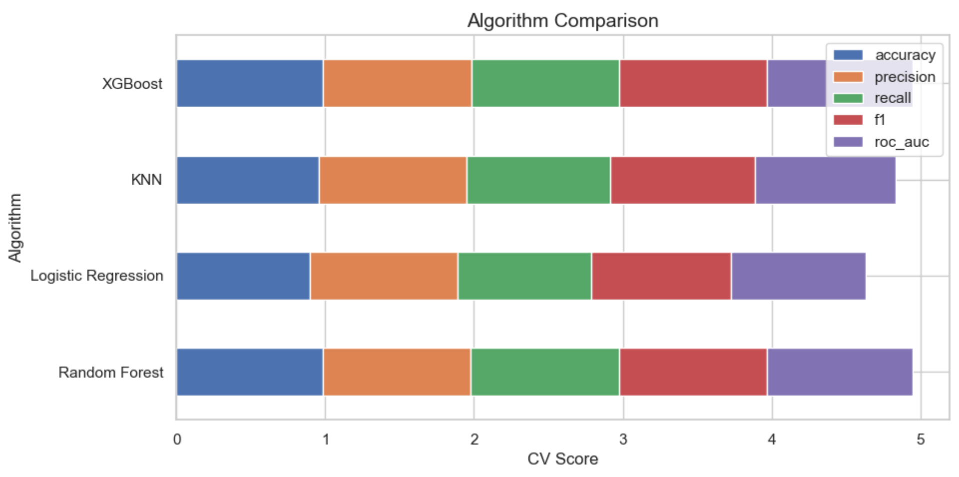
Optimized Model Evaluation:

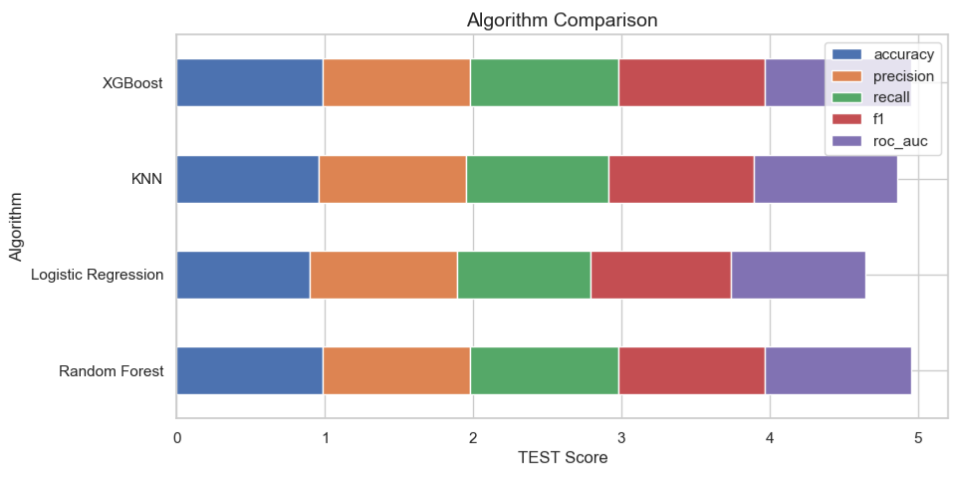


	accuracy	precision	recall	f1	roc_auc
Random Forest	0.9859	0.9954	0.9957	0.9927	0.9882
Logistic Regression	0.8975	0.9936	0.9004	0.9449	0.9068
KNN	0.9574	0.9970	0.9607	0.9778	0.9662
XGBoost	0.9871	0.9950	0.9950	0.9934	0.9847



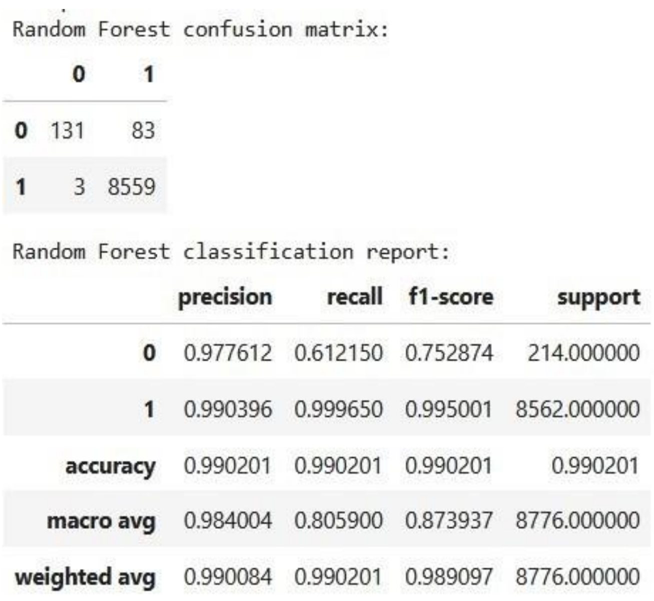
Different Algorithm Comparisons:





6.4 Results and Outcomes:

In the given images we have showed the confusion matrix and classification table for 5 algorithms, we showed the image of only one pass, however we have used total of 5 such passed in order to generate appropriate result. The we have averaged the result and presented in a table which is illustrated in next section.

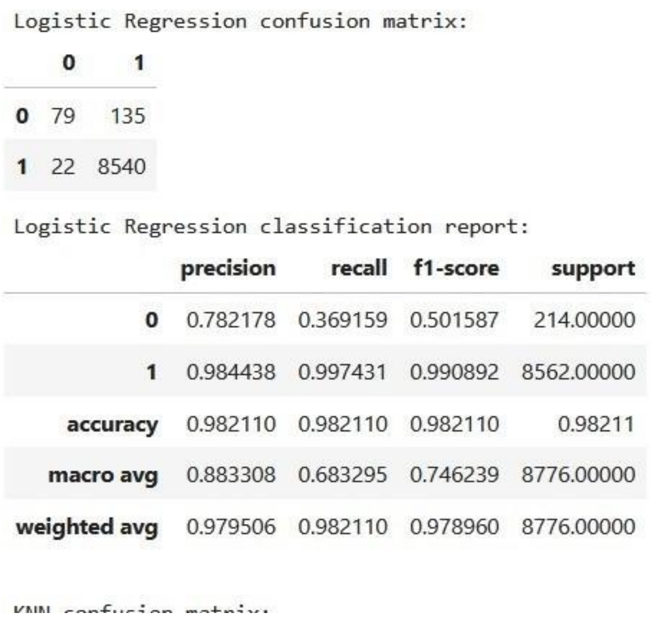


Random Forest confusion matrix:

	0	1
0	131	83
1	3	8559

Random Forest classification report:

	precision	recall	f1-score	support
0	0.977612	0.612150	0.752874	214.000000
1	0.990396	0.999650	0.995001	8562.000000
accuracy	0.990201	0.990201	0.990201	0.990201
macro avg	0.984004	0.805900	0.873937	8776.000000
weighted avg	0.990084	0.990201	0.989097	8776.000000

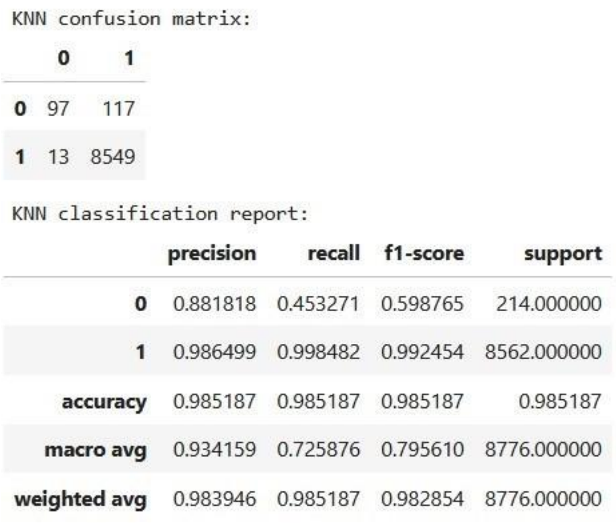


Logistic Regression confusion matrix:

	0	1
0	79	135
1	22	8540

Logistic Regression classification report:

	precision	recall	f1-score	support
0	0.782178	0.369159	0.501587	214.00000
1	0.984438	0.997431	0.990892	8562.00000
accuracy	0.982110	0.982110	0.982110	0.98211
macro avg	0.883308	0.683295	0.746239	8776.00000
weighted avg	0.979506	0.982110	0.978960	8776.00000

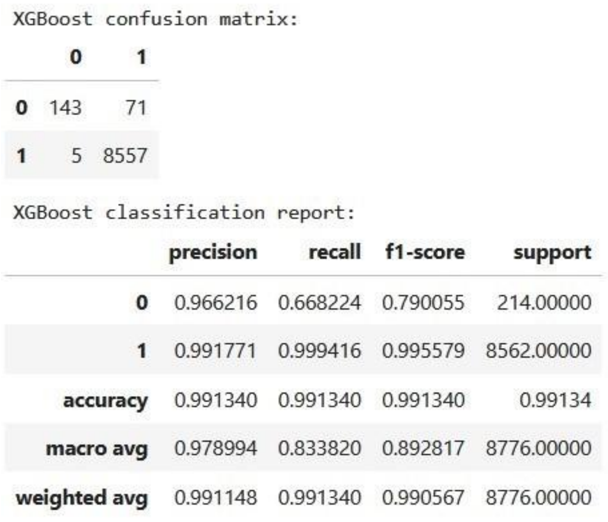


KNN confusion matrix:

	0	1
0	97	117
1	13	8549

KNN classification report:

	precision	recall	f1-score	support
0	0.881818	0.453271	0.598765	214.000000
1	0.986499	0.998482	0.992454	8562.000000
accuracy	0.985187	0.985187	0.985187	0.985187
macro avg	0.934159	0.725876	0.795610	8776.000000
weighted avg	0.983946	0.985187	0.982854	8776.000000



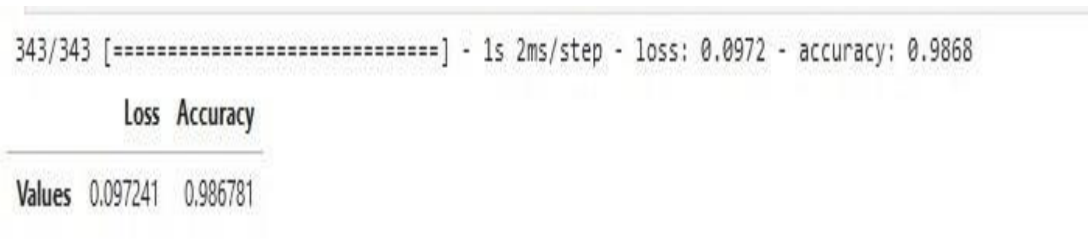
XGBoost confusion matrix:

	0	1
0	143	71
1	5	8557

XGBoost classification report:

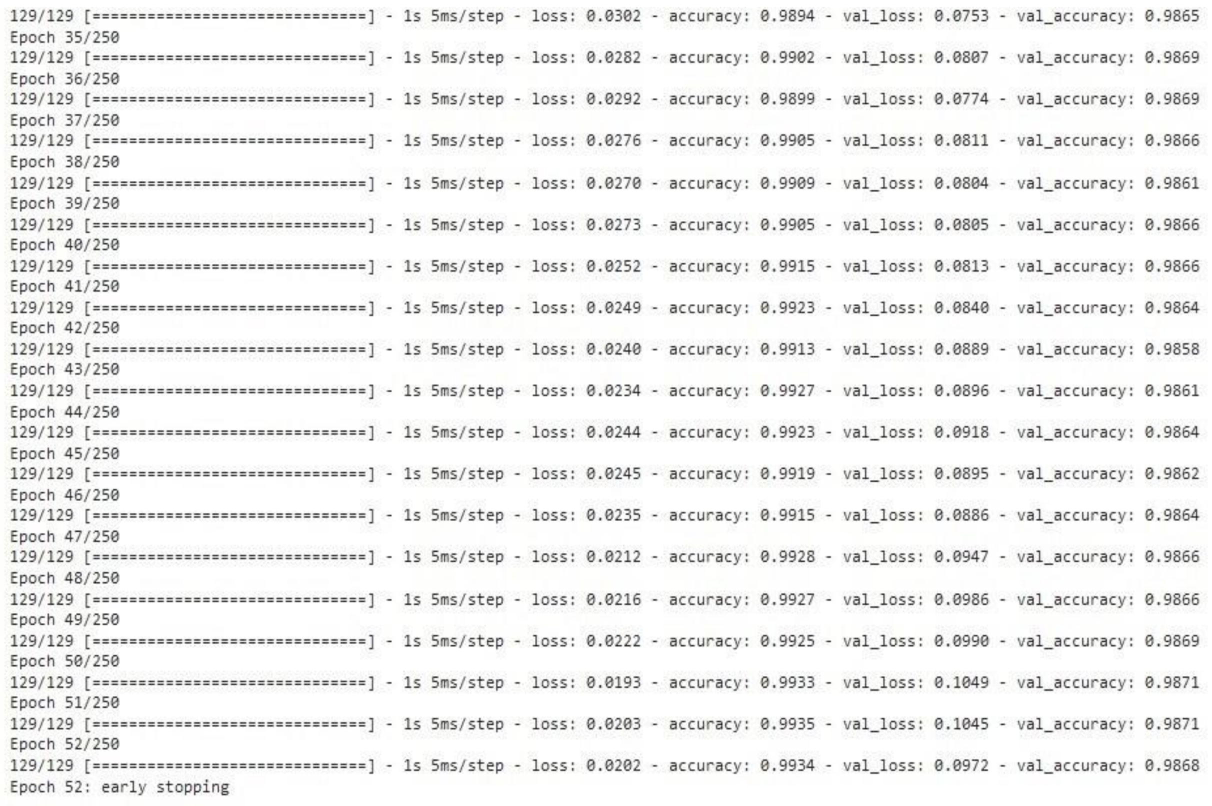
	precision	recall	f1-score	support
0	0.966216	0.668224	0.790055	214.00000
1	0.991771	0.999416	0.995579	8562.00000
accuracy	0.991340	0.991340	0.991340	0.99134
macro avg	0.978994	0.833820	0.892817	8776.00000
weighted avg	0.991148	0.991340	0.990567	8776.00000

Below is the photo of number of epochs, loss and accuracy of DL implementation.



```
343/343 [==============================] - 1s 2ms/step - loss: 0.0972 - accuracy: 0.9868
```

	Loss	Accuracy
Values	0.097241	0.986781



```
129/129 [==============================] - 1s 5ms/step - loss: 0.0302 - accuracy: 0.9894 - val_loss: 0.0753 - val_accuracy: 0.9865
Epoch 35/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0282 - accuracy: 0.9902 - val_loss: 0.0807 - val_accuracy: 0.9869
Epoch 36/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0292 - accuracy: 0.9899 - val_loss: 0.0774 - val_accuracy: 0.9869
Epoch 37/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0276 - accuracy: 0.9905 - val_loss: 0.0811 - val_accuracy: 0.9866
Epoch 38/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0270 - accuracy: 0.9909 - val_loss: 0.0804 - val_accuracy: 0.9861
Epoch 39/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0273 - accuracy: 0.9905 - val_loss: 0.0805 - val_accuracy: 0.9866
Epoch 40/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0252 - accuracy: 0.9915 - val_loss: 0.0813 - val_accuracy: 0.9866
Epoch 41/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0249 - accuracy: 0.9923 - val_loss: 0.0840 - val_accuracy: 0.9864
Epoch 42/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0240 - accuracy: 0.9913 - val_loss: 0.0889 - val_accuracy: 0.9858
Epoch 43/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0234 - accuracy: 0.9927 - val_loss: 0.0896 - val_accuracy: 0.9861
Epoch 44/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0244 - accuracy: 0.9923 - val_loss: 0.0918 - val_accuracy: 0.9864
Epoch 45/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0245 - accuracy: 0.9919 - val_loss: 0.0895 - val_accuracy: 0.9862
Epoch 46/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0235 - accuracy: 0.9915 - val_loss: 0.0886 - val_accuracy: 0.9864
Epoch 47/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0212 - accuracy: 0.9928 - val_loss: 0.0947 - val_accuracy: 0.9866
Epoch 48/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0216 - accuracy: 0.9927 - val_loss: 0.0986 - val_accuracy: 0.9866
Epoch 49/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0222 - accuracy: 0.9925 - val_loss: 0.0990 - val_accuracy: 0.9869
Epoch 50/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0193 - accuracy: 0.9933 - val_loss: 0.1049 - val_accuracy: 0.9871
Epoch 51/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0203 - accuracy: 0.9935 - val_loss: 0.1045 - val_accuracy: 0.9871
Epoch 52/250
129/129 [==============================] - 1s 5ms/step - loss: 0.0202 - accuracy: 0.9934 - val_loss: 0.0972 - val_accuracy: 0.9868
Epoch 52: early stopping
```

We used line graph to compare the training loss, validation loss as well as training accuracy and validation accuracy in case of choosing total number of epochs required which is represented below.

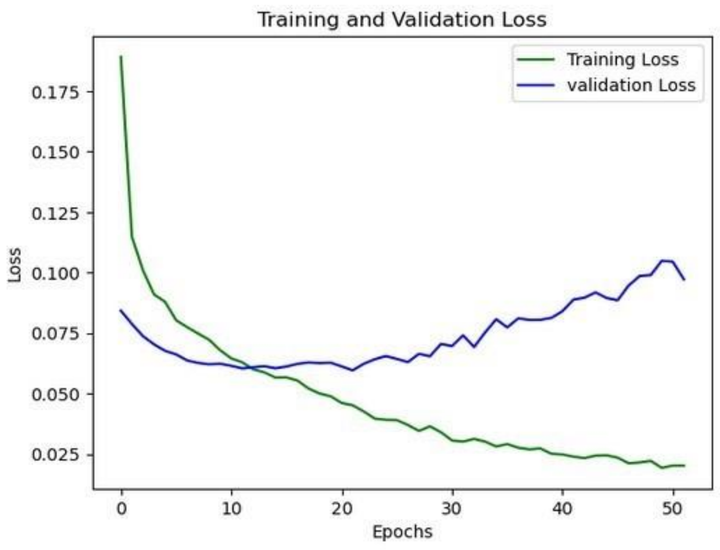


Fig 6.1 Training and validation loss

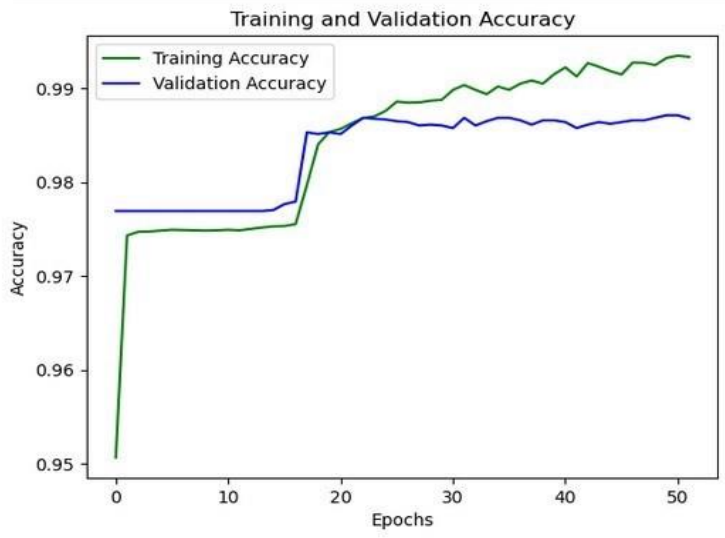
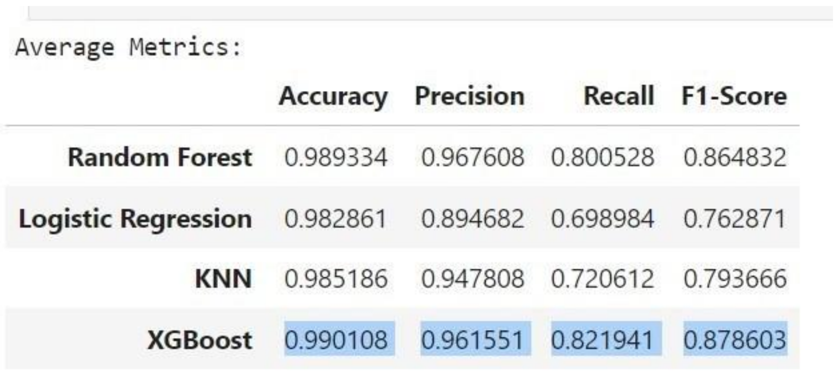


Fig 6.2 Training and validation accuracy

6.5 Result Analysis:



Average Metrics:

	Accuracy	Precision	Recall	F1-Score
Random Forest	0.989334	0.967608	0.800528	0.864832
Logistic Regression	0.982861	0.894682	0.698984	0.762871
KNN	0.985186	0.947808	0.720612	0.793666
XGBoost	0.990108	0.961551	0.821941	0.878603

We have created this table to compare the metrics of four ML algorithms, and as a result we can see that XGBoost has the highest accuracy and the highest precision among all 4 it has it has a whooping high accuracy of 0.99%, hence we have used XGBoost in our Model deployment.

The boxplot shows the closeness of the metrics used in evaluation of these four Machine Learning algorithms.

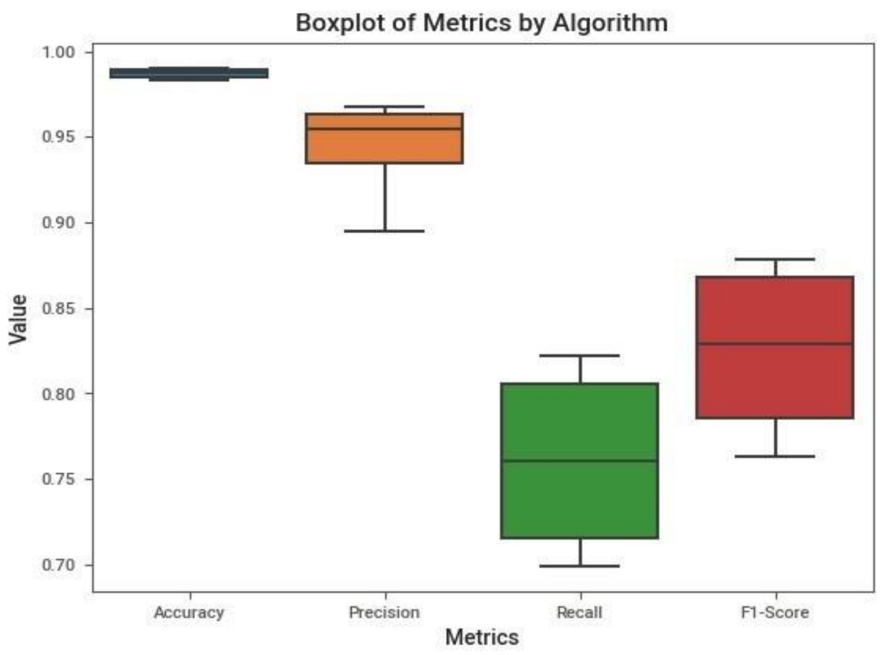


Fig 6.3 Box plot of performances

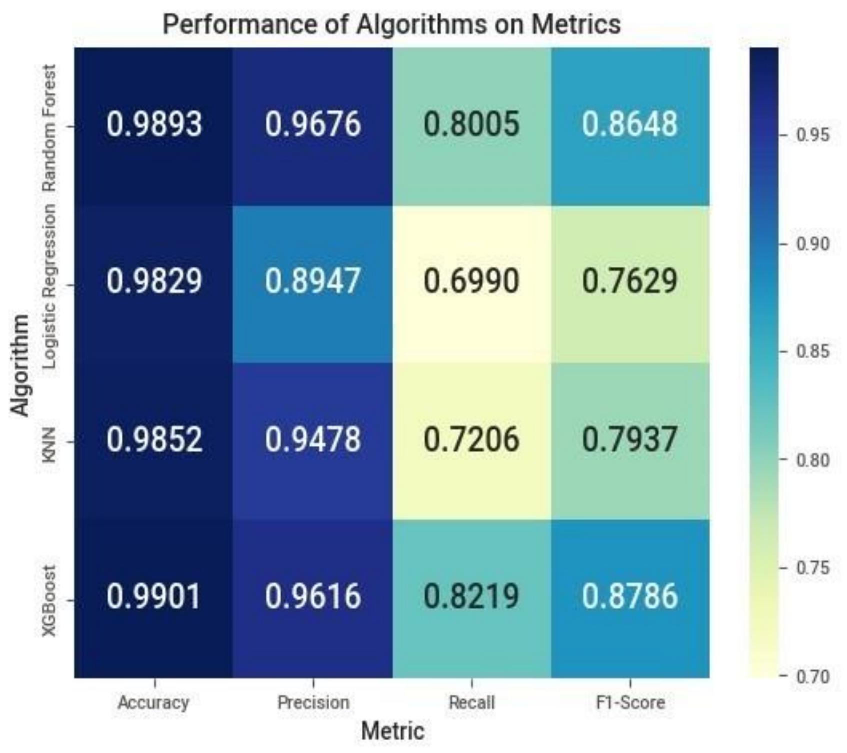


Fig 6.4 Heat map of performances

From this heatmap, we can clearly see that all the metrics of XGBoost algorithm are greater than the other three algorithms, while showing the higher metrics with darker shades and lower metrics with lighter shades.

For the final improvements in the project, we used SMOTE technique to overcome the problem of Class Imbalance, Stratified 10-fold CV and Hyperparameter Optimization using GridSearchCV to improve model performance, and Mann-Whitney U Test and Wilcoxon Signed-Rank Test to evaluate whether there is any significant difference between the performance metrics of all the ML Algorithms.

The Figure below shows the comparison of the CV scores and the Test Scores of each ML algorithm with each performance metric involved, which is done with the help of heat maps.

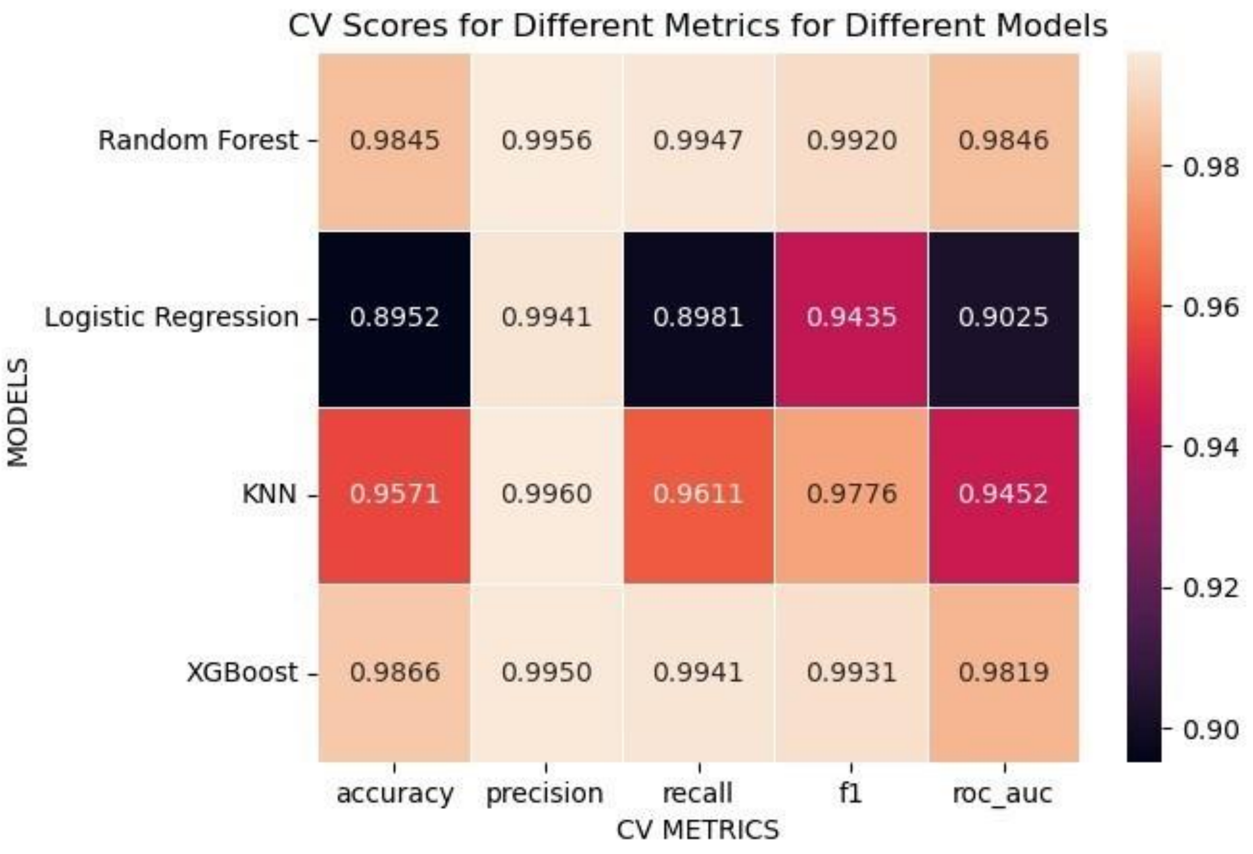


Fig 6.5 Heat map of CV metrics

The final Test Scores and CV Scores are computed by taking the minimum of the scores aggregated for each of the 4 sections. We choose the minimum because we believe all four sections are important, and so a system must consider all to raise the score.

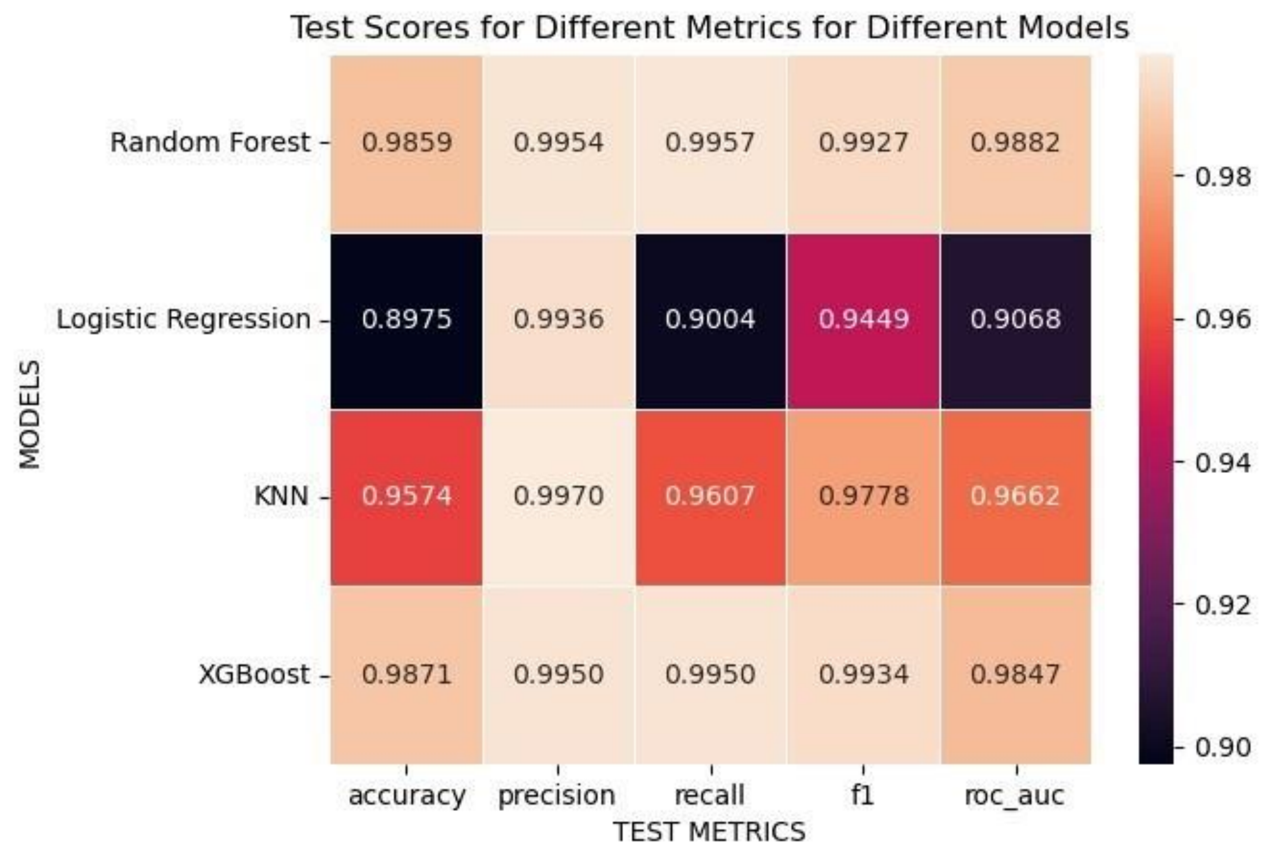
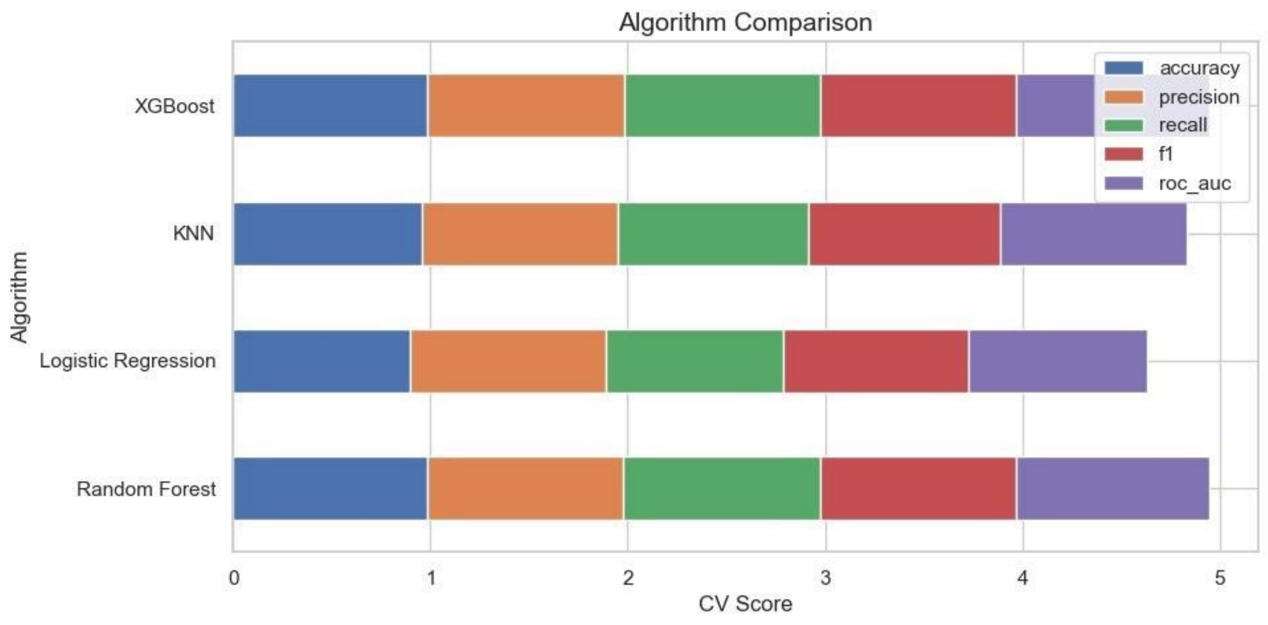


Fig 6.6 Heat map of Test metrics



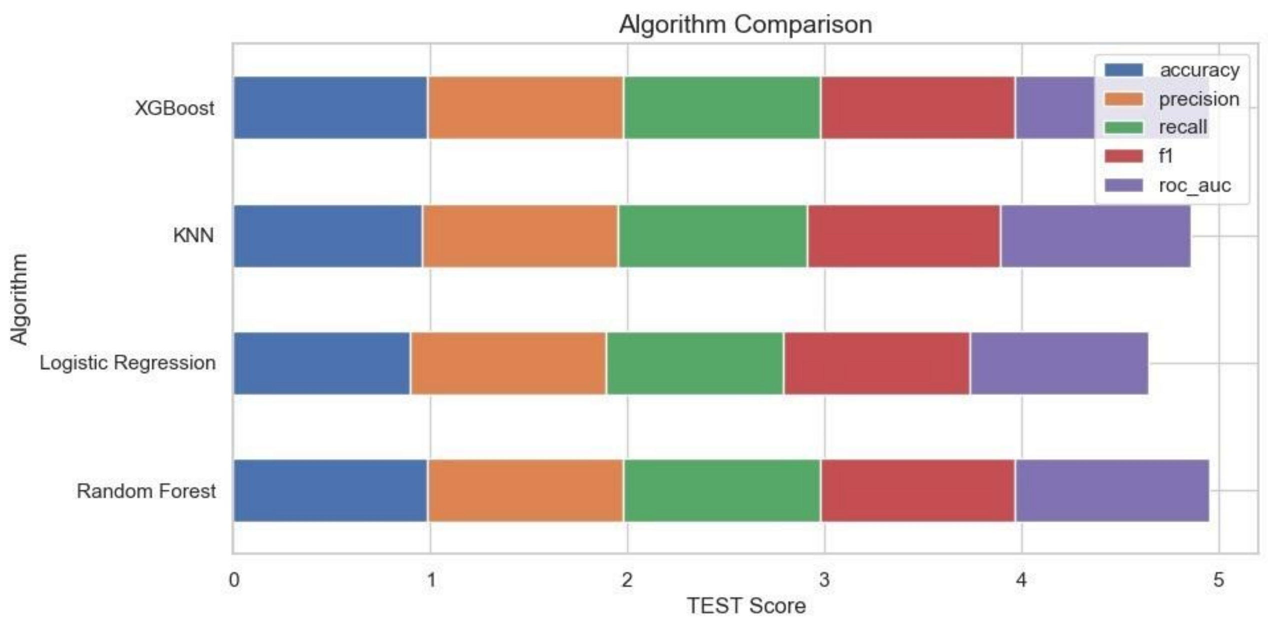


Fig 6.7 Algorithms comparison

We used SMOTE Technique to eliminate class-imbalance problem. SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

At first the total no. of oversampling observations, N is set up. Generally, it is selected such that the binary class distribution is 1:1. But that could be tuned down based on need. Then the iteration starts by first selecting a positive class instance at random. Next, the KNN's (by default 5) for that instance is obtained. At last, N of these K instances is chosen to interpolate new synthetic instances. To do that, using any distance metric the difference in distance between the feature vector and its neighbours is calculated. Now, this difference is multiplied by any random value in (0,1] and is added to the previous feature vector.

CHAPTER 7. TESTING

7.1 Testing Plan:

We have used two testing approaches for testing accuracies which are:

1. **Wilcoxon signed-rank test:**

The Wilcoxon test, which can refer to either the rank sum test or the signed rank test version, is a nonparametric statistical test that compares two paired groups. The tests essentially calculate the difference between sets of pairs and analyse these differences to establish if they are statistically significantly different from one another.

These models assume that the data comes from two matched, or dependent, populations, following the same person or stock through time or place. The data is also assumed to be continuous as opposed to discrete. Because it is a nonparametric test, it does not require a particular probability distribution of the dependent variable in the analysis.

The formula for W-statistic is given as follows:

$$W = \sum_{i=1}^{N_r} [\operatorname{sgn}(x_{2,i} - x_{1,i}) \cdot R_i]$$

W	= test statistic
N_r	= sample size, excluding pairs where x1 = x2
sgn	= sign function
$x_{1,i}$, $x_{2,i}$	= corresponding ranked pairs from two distributions
R_i	= rank i

2. Mann-Whitney U Test:

The Mann-Whitney U test is a nonparametric statistical test used to compare two independent groups of continuous data that do not follow a normal distribution. It is used to determine if there is a significant difference between the medians of the two groups.

In the Mann-Whitney U test, the data is ranked from lowest to highest, and a U statistic is calculated based on the sum of ranks of one group relative to the other. The test then compares the U statistic to a critical value to determine if the difference between the groups is statistically significant.

The Mann-Whitney U test is commonly used in scientific research and can be applied to a wide range of fields, such as psychology, biology, and economics.

The formula for Mann-Whitney U test is expressed as follows:

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2}$$

or

$$U_2 = R_2 - \frac{n_2(n_2+1)}{2}$$

where:

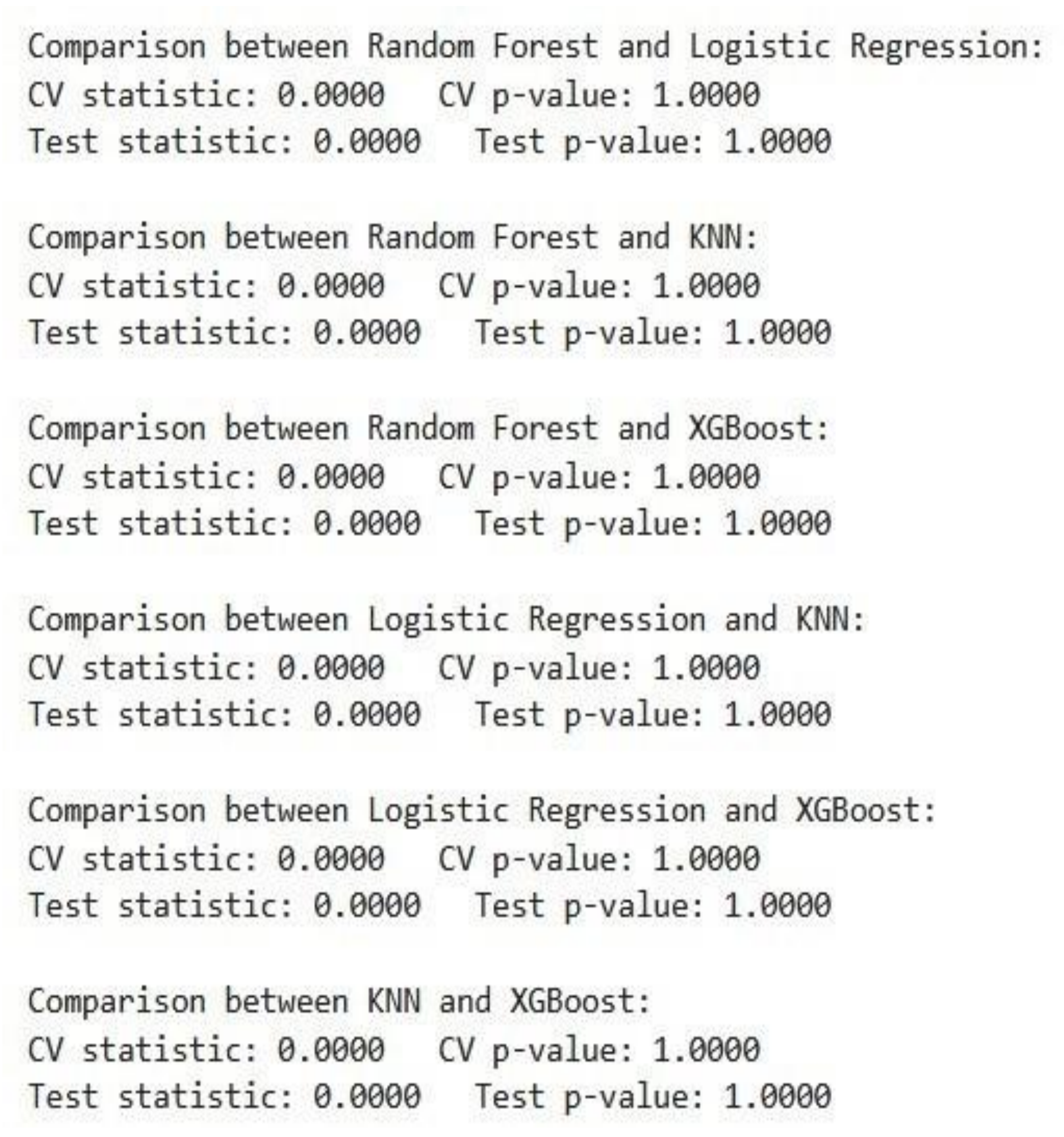
- U is the Mann-Whitney U statistic
- n1 and n2 are the sample sizes of the two groups being compared.
- R1 is the sum of the ranks of the observations in the first group.

The formula assumes that the two groups have been combined and ranked in order of increasing value.

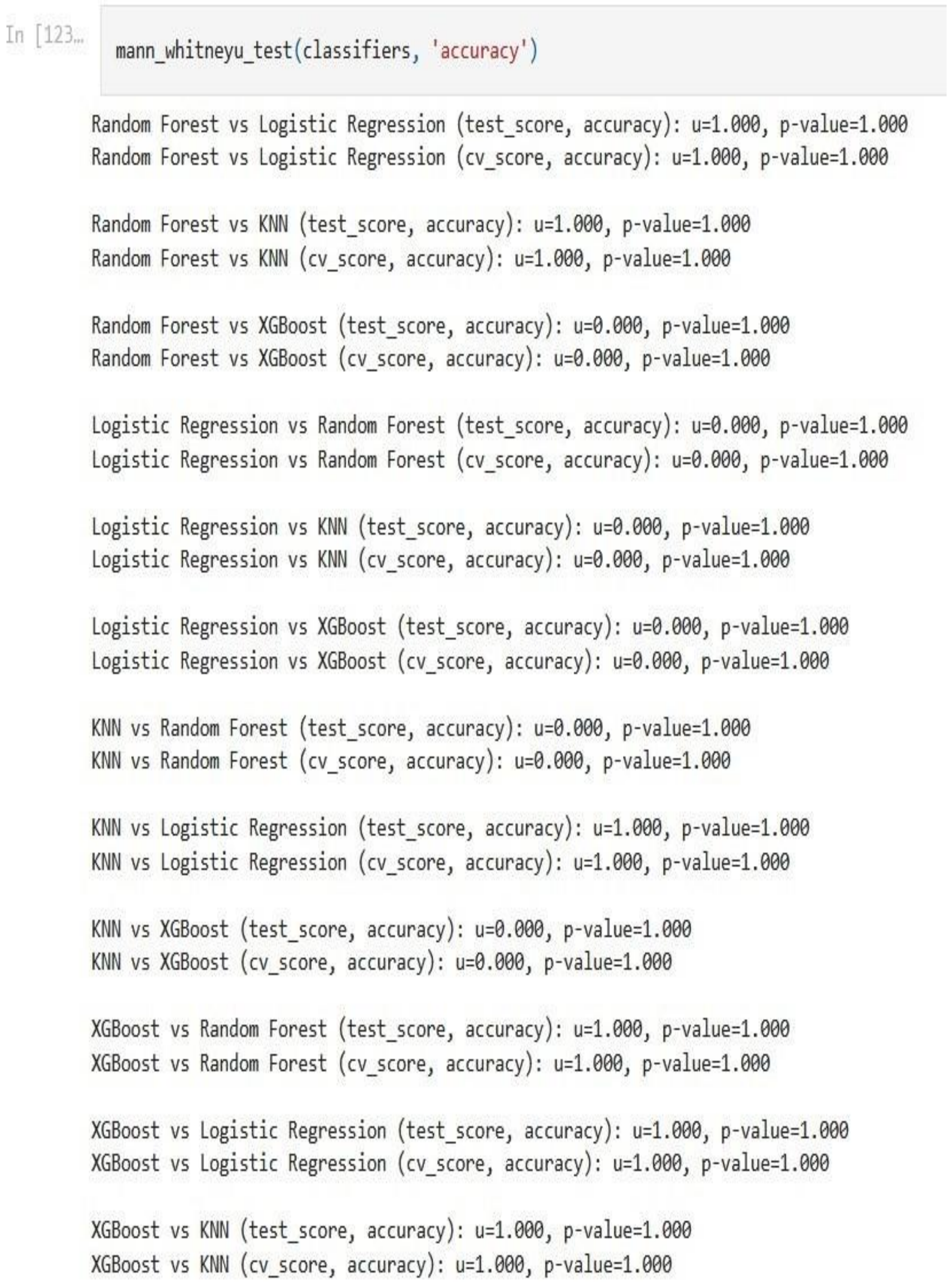
Once the U statistic has been calculated, it can be compared to a critical value in a statistical table to determine if the difference between the two groups is statistically significant. Alternatively, the p-value of the test can be calculated, which represents the probability of obtaining the observed U statistic or a more extreme value, assuming that the null hypothesis (no difference between the two groups) is true.

7.2 Test Results and Analysis:

Below is the photo of a part of result of both tests:



```
Comparison between Random Forest and Logistic Regression:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000

Comparison between Random Forest and KNN:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000

Comparison between Random Forest and XGBoost:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000

Comparison between Logistic Regression and KNN:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000

Comparison between Logistic Regression and XGBoost:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000

Comparison between KNN and XGBoost:
CV statistic: 0.0000   CV p-value: 1.0000
Test statistic: 0.0000   Test p-value: 1.0000
```



```
In [123…  mann_whitneyu_test(classifiers, 'accuracy')
```

```
Random Forest vs Logistic Regression (test_score, accuracy): u=1.000, p-value=1.000
Random Forest vs Logistic Regression (cv_score, accuracy): u=1.000, p-value=1.000

Random Forest vs KNN (test_score, accuracy): u=1.000, p-value=1.000
Random Forest vs KNN (cv_score, accuracy): u=1.000, p-value=1.000

Random Forest vs XGBoost (test_score, accuracy): u=0.000, p-value=1.000
Random Forest vs XGBoost (cv_score, accuracy): u=0.000, p-value=1.000

Logistic Regression vs Random Forest (test_score, accuracy): u=0.000, p-value=1.000
Logistic Regression vs Random Forest (cv_score, accuracy): u=0.000, p-value=1.000

Logistic Regression vs KNN (test_score, accuracy): u=0.000, p-value=1.000
Logistic Regression vs KNN (cv_score, accuracy): u=0.000, p-value=1.000

Logistic Regression vs XGBoost (test_score, accuracy): u=0.000, p-value=1.000
Logistic Regression vs XGBoost (cv_score, accuracy): u=0.000, p-value=1.000

KNN vs Random Forest (test_score, accuracy): u=0.000, p-value=1.000
KNN vs Random Forest (cv_score, accuracy): u=0.000, p-value=1.000

KNN vs Logistic Regression (test_score, accuracy): u=1.000, p-value=1.000
KNN vs Logistic Regression (cv_score, accuracy): u=1.000, p-value=1.000

KNN vs XGBoost (test_score, accuracy): u=0.000, p-value=1.000
KNN vs XGBoost (cv_score, accuracy): u=0.000, p-value=1.000

XGBoost vs Random Forest (test_score, accuracy): u=1.000, p-value=1.000
XGBoost vs Random Forest (cv_score, accuracy): u=1.000, p-value=1.000

XGBoost vs Logistic Regression (test_score, accuracy): u=1.000, p-value=1.000
XGBoost vs Logistic Regression (cv_score, accuracy): u=1.000, p-value=1.000

XGBoost vs KNN (test_score, accuracy): u=1.000, p-value=1.000
XGBoost vs KNN (cv_score, accuracy): u=1.000, p-value=1.000
```

7.3 Testing Results from the webpage

Actual data	Rows, columns	Malware count	Benign count
Split0	12724, 102	12436	288
Split1	15357, 102	14972	385
Split2	8775, 102	8554	221
Split3	7020, 102	6835	185

Table 7.1 Actual Data Results

Predicted Data	Rows, Columns	Malware Count	Benign Count	Accuracy
Split0	12724, 102	12447	277	99.91
Split1	15357, 102	14998	359	99.83
Split2	8775, 102	8585	190	99.30
Split3	7020, 102	6855	165	99.71

Table 7.2 Predicted Data Results

CHAPTER 8. Limitations and Future Enhancements

8.1 Overall Analysis of the Internship

The internship project "Malware Detection Using Machine Learning" was successfully completed with the help of the internal and external guides. The project involved the development of a machine learning model that can detect malware using a dataset of malware log files containing API calls as features and hash numbers of files as indices. Traditional and deep learning algorithms, including XGBoost, KNN, Logistic Regression, Random Forest, and Sequential MLP models, were employed in the project.

The project implemented a GridSearchCV pipeline for hyperparameter optimization and used SMOTE for dataset balancing. Statistical analysis techniques such as the Wilcoxon signed-rank test and Mann-Whitney U test were employed for performance evaluation. The trained model was deployed using FastAPI and hosted on an AWS-EC2 engine running on an Ubuntu VM, with the pickle file of the trained model made available for easy integration into other systems.

The internal guide provided insights on testing processes and the GridSearchCV pipeline process, while the external guide provided insights on the generation/sources of the dataset for model development and analysis. The project team engaged in frequent visits to the BISAG-N institute for weekly report discussions and reporting of project progress to the guides. The team also engaged in online and offline meetings with the internal guide from the college for project reviews.

Overall, the internship project demonstrated the effectiveness of machine learning techniques in detecting malware, with potential for further improvements in feature engineering techniques and ensemble models. The inputs and guidance provided by the guides were critical to the success of the project, ensuring that the project team remained on track and that the project was of high quality.

8.2 Problems Encountered

During the "Malware Detection Using Machine Learning" project, several challenges were encountered. The primary challenge was generating log files of malware for the project. Due to this, the team had to use API-call based log files of malware and benign files from Kaggle. Additionally, dataset imbalance was a big issue, which was resolved by employing the SMOTE technique.

Model fitting was also a significant challenge, as it took approximately 8 hours on an Intel i5-gen 7 256GB SSD laptop. This led to delays in the project timeline and required the team to optimize the model's parameters.

Hyperparameter selection and optimization were also time-consuming tasks that required significant effort. Implementing stratified k-fold cross-validation also took a considerable amount of time as it was necessary to develop four different models.

The model deployment using Amazon-EC2 engine also provided with some challenges, as it only allows the working of live website till the server is running on AWS. So, we optimized it by running the server in a command prompt that we duplicated on the virtual machine on the ec2 engine.

Lastly, there were some minor challenges related to project coordination and communication, such as scheduling conflicts and coordinating with the guides. However, with the help of effective communication and collaboration, these issues were resolved.

8.3 Limitation and Future Enhancements

The "Malware Detection Using Machine Learning" project successfully deployed a machine learning model for detecting malware on a basic web page using FastAPI and hosted on AWS EC2. However, the deployment was limited in terms of user-friendliness and automation capabilities. The deployed model lacks advanced features that could enhance the user experience, such as real-time notifications and user authentication. Additionally, the deployment did not include automated reporting capabilities, limiting its effectiveness for real-time monitoring.

To address these limitations, future enhancements could include the development of a more comprehensive web interface with additional features that provide users with greater control over their data and alerts. Automation capabilities could be improved by integrating continuous monitoring and automated reporting to alert users of potential malware threats in real-time.

Furthermore, the limitations of the machine learning model used in this project should also be considered. While the XGBoost model provided high accuracy, it was limited by the lack of diversity in the dataset used for model training. Additionally, the model relied solely on API call-based log files and did not take into account other potential features that could enhance the accuracy of the model.

Future enhancements could include the exploration of additional features for model development, such as file properties and network traffic analysis. Additionally, the use of cloud computing resources could be optimized to improve model fitting and deployment speed. Overall, this project provides a solid foundation for further exploration and development 28 in the field of malware detection using machine learning.

CHAPTER 9. Conclusion and References

9.1 Conclusion

In conclusion, the project of "Malware Detection Using Machine Learning" has been successfully implemented using a dataset containing log files of malware with API calls as features and hash numbers of files as indices. Two approaches were used - traditional machine learning algorithms and deep learning algorithms. After comparing the results, the XGBoost model proved to be the most accurate model with an average accuracy of 99.69% on the test data.

The project faced several challenges during the implementation, such as dataset imbalance, hyperparameter selection and optimization, and long model fitting time. The implementation of GridSearchCV pipeline for hyperparameter optimization and SMOTE technique for dataset imbalance removal was used to overcome these challenges.

The model was also deployed on a basic web-page and hosted on AWS-EC2 engine using Ubuntu VM. Future enhancements could include further automation of the model, improved web interface, and the use of more diverse datasets to increase the model's robustness. Additionally, the limitations of the project include the limited availability of the dataset and the use of only one type of log file.

Overall, this project has demonstrated the potential of machine learning in detecting malware with high accuracy and provided a foundation for future improvements and applications in the field of cybersecurity.

Code Link: https://github.com/shashankgsharma/malwaredetectionsystem/

Deployed Link: http://ec2-13-239-47-93.ap-southeast-2.compute.amazonaws.com:8080/

9.2 References

1. Alazab, M., Venkatraman, S., & Watters, P. (2012). A novel hybrid approach for malware detection in android. International Journal of Information and Computer Security, 4(1-2), 44-64.

2. Kolter, J. Z., & Maloof, M. A. (2006). Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research, 7(Sep), 2721-2744.

3. Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153-1176.

4. Saxe, J. B., Berlin, K., Cunningham, R. K., & Kemmerer, R. A. (2015). Pigs (polygraph-inspried graph similarity) filter: A tool for cyber attack attribution. In Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (pp. 51-62).

5. Rastogi, A., Garg, P., & Choudhary, S. (2017). Detecting malicious activity with machine learning: An approach towards enhancing cyber security. In Proceedings of the 2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS) (pp. 34-37).

6. Kaggle. (2021). Microsoft Malware Prediction. Retrieved from https://www.kaggle.com/c/microsoft-malware-prediction

7. Scikit-learn. (2021). Ensemble Methods. Retrieved from https://scikit-learn.org/stable/modules/ensemble.html

8. TensorFlow. (2021). Sequential Model API. Retrieved from https://www.tensorflow.org/guide/keras/sequential_model

9. FastAPI. (2021). Official Documentation. Retrieved from https://fastapi.tiangolo.com/

10. Amazon Web Services. (2021). EC2 Instance Types. Retrieved from https://aws.amazon.com/ec2/instance-types/

11. Demontis, A., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., & Giacinto, G. (2017). Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing, 15(4), 650-663.

12. Gharib, H., Saaidia, M., & Benferhat, S. (2020). Machine learning for malware detection: A survey. Journal of Big Data, 7(1), 1-35.

13. Hu, Y., Tian, Y., Mu, Y., & Huang, X. (2018). A multi-modal deep learning framework for android malware detection. Journal of Computer Science and Technology, 33(6), 1207-1218.

14. Kim, D., Lee, D., Lee, D., Lee, J., Lee, J., Kim, J., & Kim, H. (2020). Detection of malware in smart factory using machine learning. Electronics, 9(3), 469.

15. Liao, Y., & Xue, Y. (2019). Research on malware detection method based on machine learning. International Journal of Distributed Sensor Networks, 15(7), 1550147719865304.

16. Mariconti, E., Onwuzurike, L., Andriotis, P., & Stringhini, G. (2018). A game of hide and seek: Tracking down malicous servers in anonymity networks. In Proceedings of the 2018 World Wide Web Conference (pp. 1291-1300).

17. Mohaisen, A., Nyman, T., & Fu, X. (2016). Enhancing android malware detection with ensemble of classifiers. In Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (pp. 81-92).

18. Raff, E., Nicholas, C., McLean, M., Liu, D., Shirley, J., & Baxter, K. (2018). Malware detection by eating a whole exe. arXiv preprint arXiv:1803.04173.

19. Saxe, J. B., Berlin, K., Cunningham, R. K., & Kemmerer, R. A. (2017). Pigs can fly: Co-evolutionary learning for malware detection. IEEE Transactions on Cybernetics, 47(3), 648-661.

20. Zou, Y., Zhang, Y., Hu, J., & Jin, S. (2020). An android malware detection method based on improved convolutional neural network. International Journal of Security and Its Applications, 14(6), 167-176.

● **44% Overall Similarity**

Top sources found in the following databases:

- 42% Internet database
- Crossref database
- 38% Submitted Works database
- 16% Publications database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	core.ac.uk Internet	6%
2	geeksforgeeks.org Internet	4%
3	Darshna Patel, Saurabh Shah, Hitesh Chhinkaniwala. "Fuzzy logic base... Crossref	3%
4	BRAC University on 2021-01-31 Submitted works	3%
5	en.wikipedia.org Internet	3%
6	vitalflux.com Internet	3%
7	coursehero.com Internet	2%
8	analyticsvidhya.com Internet	2%

9	docs.aws.amazon.com Internet	2%
10	1library.net Internet	1%
11	jetir.org Internet	1%
12	bisag-n.gov.in Internet	1%
13	Charotar University of Science And Technology on 2021-04-20 Submitted works	1%
14	ijcaonline.org Internet	<1%
15	docplayer.net Internet	<1%
16	Segi University College on 2017-07-19 Submitted works	<1%
17	researchgate.net Internet	<1%
18	topdigitalmarketing.agency Internet	<1%
19	slideshare.net Internet	<1%
20	einfochips.com Internet	<1%

21	Saveetha Dental College and Hospital, Chennai on 2023-01-19 Submitted works	<1%
22	hatching.io Internet	<1%
23	American University of Culture and Education on 2023-04-04 Submitted works	<1%
24	Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. "The M... Crossref	<1%
25	Liverpool John Moores University on 2021-08-30 Submitted works	<1%
26	ijsrcseit.com Internet	<1%
27	sparkcognition.com Internet	<1%
28	Coventry University on 2021-04-30 Submitted works	<1%
29	University of Bradford on 2023-05-08 Submitted works	<1%
30	dspace.univ-jijel.dz:8080 Internet	<1%
31	ijert.org Internet	<1%
32	University of Greenwich on 2023-04-26 Submitted works	<1%

33	Kampala International University on 2019-12-13 Submitted works	<1%
34	Liverpool John Moores University on 2021-03-02 Submitted works	<1%
35	securitywizardry.com Internet	<1%
36	Coventry University on 2010-09-01 Submitted works	<1%
37	National University Of Science and Technology on 2019-09-05 Submitted works	<1%
38	Charotar University of Science And Technology on 2020-04-19 Submitted works	<1%
39	Saja I. Hani, Naji M. Sahib. "Detection of Malware under Android Mobil... Crossref	<1%
40	negd.gov.in Internet	<1%
41	rest.neptune-prod.its.unimelb.edu.au Internet	<1%
42	studentsrepo.um.edu.my Internet	<1%
43	Kingston University on 2011-02-04 Submitted works	<1%
44	University of Rome Tor Vergata on 2018-07-03 Submitted works	<1%

45	ro.ecu.edu.au Internet	<1%
46	Wilkes University on 2023-03-11 Submitted works	<1%
47	Visvesvaraya Technological University on 2015-03-05 Submitted works	<1%
48	chromeias.in Internet	<1%
49	science.gov Internet	<1%
50	University of Bradford on 2020-09-18 Submitted works	<1%
51	The Robert Gordon University on 2016-01-07 Submitted works	<1%
52	Universiti Teknologi MARA on 2019-02-26 Submitted works	<1%
53	University of Sydney on 2019-04-12 Submitted works	<1%
54	University of Ulster on 2013-05-07 Submitted works	<1%
55	City University of Hong Kong on 2010-04-19 Submitted works	<1%
56	Georgetown University on 2019-08-11 Submitted works	<1%

57	King's College on 2023-04-21 Submitted works	<1%
58	Universiti Tenaga Nasional on 2023-05-03 Submitted works	<1%
59	University of Warwick on 2011-09-26 Submitted works	<1%
60	A.M.S.N. Amarasinghe, W.A.C.H. Wijesinghe, D.L.A. Nirmana, Anuradh... Crossref	<1%
61	Imperial College of Science, Technology and Medicine on 2018-08-31 Submitted works	<1%
62	Kingston University on 2013-05-03 Submitted works	<1%
63	The University of Manchester on 2019-04-30 Submitted works	<1%
64	University of Abertay Dundee on 2019-05-14 Submitted works	<1%
65	University of Hertfordshire on 2021-05-10 Submitted works	<1%
66	dergipark.org.tr Internet	<1%