**ANDROID MALWARE DETECTION USING MACHINE LEARNING**

Submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering degree in Computer Science and Engineering

by

**KAKUMANU GEETHA MADHAV ( Reg.No - 40110519 )**

**GUTTAPALEM SAMITHA ( Reg.No – 40110423 )**

****

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**SCHOOL OF COMPUTING**

**SATHYABAMA**

## INSTITUTE OF SCIENCE AND TECHNOLOGY

## (DEEMED TO BE UNIVERSITY)

## CATEGORY -1 UNIVERSITY BY UGC

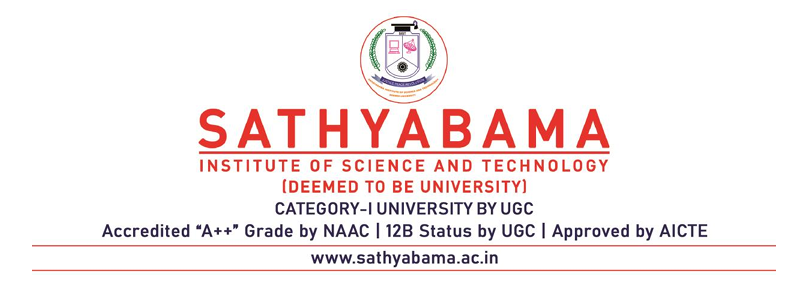
**Accredited with Grade “A++” by NAAC I 12B Status by UGC I Approved by AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHISALAI,**

**CHENNAI - 600119**

# 

# APRIL - 2024

****

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **KAKUMANU GEETHA MADHAV(40110519) and GUTTAPALEM SAMITHA(40110423)** who carried out the Project work entitled **“Android Malware Detection Using Machine Learning”**under my supervision from November 2023 to April 2024.

**Internal Guide**

**Ms. MANJU C NAIR M.E(PhD.)**

**Head of the Department**

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

**Submitted for Viva voice Examination held on**

**Internal Examiner External Examiner**

**DECLARATION**

I, **GUTTAPALEM SAMITHA (Reg.No– 40110423),** hereby declare that the Project Report entitled **“Android Malware Detection Using Machine Learning”** done by me under the guidance of **Ms. MANJU C NAIR M.E(PhD.)** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

**DATE: 12.04.2024**

## PLACE:CHENNAI SIGNATURE OF THE CANDIDATE

**ACKNOWLEDGEMENT**

I am pleased to acknowledge my sincere thanks to **Board of Management of** **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

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

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.** **MANJU C NAIR** **M.E(PhD.)** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

## ABSTRACT

# The Android smartphone, with its wide range of uses and excellent performance,has attracted numerous users. Still, this domination of the Android platform also has motivated the attackers to develop malware. The traditional methodology which detects the malware based on the signature is unfit to discover unknown applications. In this paper, we tried to detect whether an application is malware or not using Static Analysis. We considered all the permissions that an application asks for and took them as input to feed our machine learning models. We built different classifiers using different machine learning algorithms such as Support Vector Machine(Linear and RBF), Logistic Regression,Random Forest Algorithm, Gaussian Naive-Bayes, Decision Tree Method etc., and we compared their performances.

# Mobile gadgets have become ubiquitous in this age of lightning-fast technical development. On the other hand, malware has become a much bigger concern as Android has become more prevalent. The ever-changing nature of Android malware makes it imperative to investigate alternative techniques of detection, since traditional ones frequently fail to keep pace. A method for detecting Android malware using machine learning (ML) is suggested in this research. The suggested method improves mobile user security by efficiently identifying and classifying Android malware using ML techniques. A number of features are used by the system to get useful information from Android apps, including permission requests, API calls and system calls. Decision trees, support vector machines (SVMs), neural networks, and other ML techniques are given these characteristics in order to train and construct strong models that can distinguish between safe and dangerous applications.The suggested approach has shown encouraging results in experimental assessment, with high detection rates of Android malware. In addition, ML models can adjust and acquire knowledge from fresh malware samples, guaranteeing ongoing defense against new dangers. As a whole, this study demonstrates how ML may improve Android malware detection, which is a great step toward meeting the difficulties of mobile security head-on.

# Keywords:Static Analysis, Dynamic Analysis, Feature Selection

# **TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER**  **NO** | **TITLE** | | **PAGE**  **NO** |
|  | **ABSTRACT** | | v |
|  | **LIST OF FIGURES** | | viii |
|  | **LIST OF ABBREVATIONS** | | ix |
| **1** | **INTRODUCTION** | | 1 |
|  | 1.1 | OBJECTIVE | 2 |
|  | 1.2 | MOTIVATION | 2 |
|  | 1.3 | PROBLEM STATEMENT | 2 |
| **2** | **LITERATURE SURVEY** | | 3 |
|  | 2.1 | INFERENCES FROM LITERATURE SURVEY | 3 |
|  | 2.2 | OPEN PROBLEMS IN EXISTING SYSTEM | 7 |
| **3** | **AIM AND SCOPE OF THE PRESENT INVESTIGATION** | | 9 |
|  | 3.1 | AIM | 9 |
|  | 3.2 | SCOPE | 9 |
|  | 3.3 | RISK ANALYSIS OF THE PROJECT | 10 |
|  |  | 3.3.1 FEASIBILITY STUDY | 10 |
|  |  | 3.3.2 ECONOMICAL FEASIBILITY | 10 |
|  |  | 3.3.3 TECHNICAL FEASIBILITY | 11 |
|  |  | 3.3.4 OPERATIONAL FEASIBILITY | 11 |
|  | 3.4 | REQUIREMENTS SPECIFICATION | 11 |
|  |  | 3.4.1 HARDWARE SPECIFICATION | 11 |
|  |  | 3.4.2 SOFTWARE SPECIFICATION | 11 |
| **4** | **DESCRIPTION OF PROPOSED SYSTEM** | | 12 |
|  | 4.1 | SELECTED METHODOLOGY OR PROCESS MODEL | 13 |
|  |  | 4.1.1 DATASET | 13 |
|  |  | 4.1.2 FEATURE ENGINEERING | 13 |
|  | 4.2 | DATA PREPROCESSING | 14 |
|  | 4.3 | FEATURE SELECTION | 15 |
|  | 4.4 | DESCRIPTION OF PROGRAMMING LANGUAGE AND SOFTWARE | 15 |
|  |  | 4.4.1 PYTHON | 16 |
|  |  | 4.4.2 GOOGLE COLAB | 16 |
|  |  | 4.4.3 USING GOOGLE COLAB | 17 |
|  |  | 4.4.4 LOGISTIC REGRESSION | 18 |
|  |  | 4.4.5 K-NEAREST NEIGHBOURS | 19 |
|  |  | 4.4.6 SVM LINEAR | 19 |
|  |  | 4.4.7 DECISION TREE | 20 |
|  |  | 4.4.8 RANDOM FOREST METHOD | 21 |
|  |  | 4.4.9 GAUSSIAN NAIVE BAYES | 22 |
|  |  | 4.4.10 ANACONDA SPYDER SOFTWARE | 23 |
| **5** | **IMPLEMENTATION DETAILS** | | 25 |
| **6** | **RESULTS AND DISCUSSION PERFORMANCE ANALYSIS** | | 27 |
|  | 6.1 | RESULTS | 27 |
|  | 6.2 | PERFORMANCE ANALYSIS | 28 |
| **7** | **SUMMARY AND CONCLUSIONS** | | 31 |
|  | 7.1 | CONCLUSION | 31 |
|  | 7.2 | FUTURE WORK | 32 |
|  | **REFERENCES** | | 33 |
|  | **APPENDIX** | | 36 |
|  | 1. SOURCE CODE | | 36 |
|  | 1. SCREEN SHORT | | 54 |
|  | 1. RESEARCH PAPER | | 61 |
|  | 1. CERTIFICATE | | 68 |

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **PAGE NO** |
| 4.2.1 | DATA PREPROCESSING | 15 |
| 4.4.5.1 | K-NEAREST NEIGHBOURS | 19 |
| 4.4.6.1 | SUPPORT VECTOR MACHINE | 20 |
| 4.4.7.1 | ILLUSTRATION OF DECISION TREE | 21 |
| 4.4.8.1 | ILLUSTRATION OF RANDOM FOREST | 22 |
| 6.1.1 | BEST ACCURACYALGORITHM | 28 |
| B.1 | MALWARE PREDICTOR | 54 |
| B.2 | LIST OF LEARNING MODELS | 54 |
| B.3 | LOGISTIC REGRESSION MODEL | 55 |
| B.4 | SUPPORT VECTOR MACHINE MODEL | 55 |
| B.5 | RANDOM FOREST CLASSIFIER | 56 |
| B.6 | ADABOOST CLASSIFIER | 56 |
| B.7 | MULTINOMIAL NAIVE BAYES | 57 |
| B.8 | KNN CLASSIFIER | 57 |
| B.9 | ACCURACY BAR GRAPH | 58 |
| B.10 | PRECISION BAR GRAPH | 58 |
| B.11 | HOME PAGE | 59 |
| B.12 | SAFE URL | 59 |
| B.13 | MALWARE URL | 60 |

# LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| **ABBREVIATION** | **EXPANSION** |
| API | APPLICATION PROGRAM INTERFACE |
| APK | ANDROID APPLICATION PACKAGE |
| KNN | K-NEARST NEIGHBOURS |
| MDM | MOBILE DEVICE MANAGEMENT |
| SVM | SUPPORT VECTOR MECHINE |
| URL | UNIFO RESOURCE LOCATOR |
| WEKA | WAIKATO ENVIRONMENT FOR KNOWLEDGE ANALYSIS |

# CHAPTER 1

# INTRODUCTION

Recent years, with the rapid development of smartphones, Android is becoming more and more popular. According to the IDC report shown in the 2021 year, Android market share will reach 85.3% until the end of 2021. Android has been more and more indispensable with its open source character and advantages of free in our daily life. However, the number of malicious software is also growing rapidly. Therefore, how to detect the Android malware with a high accurate rate is a hot issue. Traditional detection approach based on signature is widely used both in Android devices and PC by extracting the signature from APK and comparing it with the malicious signature in the virus database.However, this approach is limited to detect unknown malwares which do exist in the virus database.

Due to its open platform and lack of restrictions on installed applications, Android is more prone to cyber attacks. Malware is software developed by attackers to steal, destroy, or damage information and assets. Malware threatens user’s privacy and security by enabling attackers to gain access, monitor and spy on victims' devices. Hence, the detection of Android malware is vital to create a safe environment for 2.5 billion Android users.

Malware detection approaches are divided into two; the first approach is using traditional non-machine learning techniques which include the detection using signature based, dynamic taint analysis and permissions based. While the second approach is with the use of machine learning algorithms to build a model that would detect and prevent malware from infecting end devices. However, traditional malware detection methods are unable to detect unseen malwares and are more time consuming when compared with machine learning algorithms. [4] In this paper a machine learning model is built to detect malware attacks on android devices. The model was tested on the dataset CICMal Droid2020 using four different classifiers.

# 1.1 OBJECTIVE

To implement a machine learning model that can detect the presence of any malware in an android application accurately.

# **1.2 MOTIVATION**

In June 2021, Android retained its status as the world’s most popular mobile operating system, with a market share of about 73 percent. As Android has grown in popularity,so has the amount of malware targeting the platform. Android has been the most popular target for malware makers since 2010, accounting for more than 90% of all mobile malware.

Android applications can be downloaded from a lot of sources including third-party app stores. These application stores serve as a point of entry for malware to spread quickly. The most common strategy used by malware creators is Repackaging, which involves embedding the harmful code segment within a legal application making the application malicious, and distributing it through third-party stores. Their malware programs are designed to get root access, stealing personal information, setting up mobile botnets and many more. So far, there are a large number of malware discovered, some of which are more dangerous than others. They are harmful because they have the potential to steal user’s personal information, as well as sending this data to remote servers.So, there is a need to develop models that can detect the presence of Android malware with accuracy and that is our motivation.

# **1.3 PROBLEM STATEMENT**

Smartphones have become the most used device in one’s day to day life. They facilitate users with a variety of applications that are enriched with powerful features. It is almost impossible for anyone these days to spend a day without their smartphones.Out of all smartphones, Android smartphones are the ones that are widely used. This increasing popularity of Android smartphones has also attracted malicious attackers. This malicious activity can be done by either a single application or a group of applications working together. The objective of this project is to create a model that can detect such malicious applications.

# CHAPTER 2

# LITERATURE SURVEY

# 2.1 INFERENCES FROM LITREATURE SURVEY

We studied the techniques that are proposed to identify Android malwares. In his work, Anshul et al. [1] presented an idea to detect Android Malwares by Network traffic analysis. Their approach is used to identify malware on Android that is operated by a remote server. These malwares either accept orders from the server or leak sensitive data to it. First, they analyzed the network traffic of android malwares and then the traffic of normal applications. They discovered the characteristics that distinguish malware traffic from non-malware traffic.. And in the second phase, they built a classifier using these network traffic features which can detect the malwares.

In another work, Anshul et al. [2] proposed a technique called the PermPair method. They approached the goal by considering every pair of permissions as the possible input feature and finally decided on each pair, if that combination is vulnerable. Their method includes data sets from 3 different sources called Genome ,Debris and Koodoos. Their approach had 3 phases. In the first phase, they constructed 4 different graphs by extracting permission pairs from each application.

Out of the 4 graphs, 3 graphs are for malwares and 1 graph is for benign applications. In the second phase, they dealt with merging 3 malicious graphs into a single malicious graph. At the end of this phase, they ended up with two graphs, one for malicious and one for benign. In the third and final phase of their method, they developed a model that calculates two scores called normal score and malicious score for every application and decides whether a particular application is malware or not.

The most commonly used properties in static and dynamic Android malware detection are permissions and network traffic features respectively. Static permissions cannot identify sophisticated malware, which is capable of update attacks. And coming to dynamic network traffic, it cannot detect malware samples without a network connection. There fore, a hybrid model integrating both of these properties is proposed. They extracted both permissions and network traffic features and made them into a single vector.

Using the K-medoids method, they partitioned the vectors into K clusters. And they used the K-Nearest Neighbours method, to classify whether a particular application is malicious or not. They made sure that K is odd, just to make sure out of K nearest neighbours, the count of malicious and benign neighbours is not the same.

In another work, Zhenlong Yuan et al. [3] proposed a technique to associate static features with dynamic features and then classify the given android applications as malicious or safe. They got the features they used as input to their model in three stages:

• Static Phase

• Sensitive APIs

• Dynamic Phase

Static phase includes the permissions that are obtained by unzipping the apk file and parsing xml files obtained later. Another file classes.dex accounts for the sensitive api calls . To obtain the dynamic features, they used to install the apk files on droidbox, which is an extended sandbox of Taintdroid, which is itself capable of recording the network traffic,information leak and other run time analytics that could play a major role in determining an app as malware or not. They implemented an online android detection engine based on deep learning models.They are able to develop a deep learning model that outperforms many state of art machine learning techniques.

In another work, Suleiman et al. [4] examined the use of varied machine learning techniques in a parallel classification approach to Android malware detection .The proposed method made use of a variety of properties,API calls, instructions, and other API-related topics were provided.The recent rise of Android usage Malware and its ever-increasing ability to be detected ,current signature-based techniques are avoided.The classification approach adapted by them is a realistic option.The method that not only gives a supple mental tool but also it has the potential to improve Android malware detection, but it also has some drawbacks.

It only permits different classifiers’ strengths to be combined. Assisting with further phases of analysis Moreover the planned from a performance standpoint, the technique is great since it is economical when it comes to classifying an unknown instance because the app’s static features are consumed and also chosen features are con sumed. Models for constituent categorization have a minimal computational cost before making a categorization judgement.

Xingquan Zhu et al. [5] proposed a feature based learning method that takes input as the permissions and api calls an app is intended to make during its life time. The proposed frame work consists of 4 different phases. They are:

• App Analyser (decompresses) the apk file

• Fetches permissions and api calls

• Feature generator (Generate binary datasets)

• Data mining models(SVM, Bagging, Decision Tree)

Though this approach doesn’t involve any dynamic analysis , this approach of com plimenting the permissions with the api calls improves the performance of the model and helps in the better performance of the model. To obtain input parameters from bundled Android apps.

Justin et al. [6] used an apk called Androguard. This androguard is used to obtain permission features from the jar files included in the android apk file. They then utilised the Scikit learn framework, a python library. It gives a straightforward interface to LIBSVM , to train a SVM having 1 class, utilising these obtained features. They aimed to reduce the high false positive rates generally seen in these android malware detection techniques.

They are only reported after testing data as malware if it differs sufficiently from the training data, which is ideal for their purposes because the range of benign applications are more widely available than malicious applications.For the Android operating system, they introduced a revolutionary malware detection solution which works on the principle of ML.

In Zi Wang et al. [7], they offered Droid Deep Learner, an Android malware characterization and identification strategy that uses a deep learning algorithm to distinguish malicious Android apps from benign ones.They carried out a series of tests to ensure that the suggested technique was both legitimate and effective,and compared the performance of the proposed Droid Deep Learner with that of SVM, which is one of the most effective and commonly used algorithms for malware detection.They ran a variety of experiments using real Android app datasets to validate the malware detection technique, and the findings suggest that the scheme can accurately identify malware in the Android. When compared to traditional solutions which works on SVM as backbone, experiment results suggest that the proposed system is capable of accurately identifying malware.

In Hyo-Sik Ham et al. [8], a study is undertaken on the detection of malware using an Android device, which is the most commonly used as an attack point by attackers. This paper presents a feature selection and experimentation technique for lowering the number of malware detections that are false positives and improving device performance degradation that has been identified as a problem in the current system Machine learning based mobile malware detection.

This model also analyses malware detection methods and performance of machine learning classifiers.First, they must compare the agent’s resource consumption to that of an existing agent that is monitoring all features. In the future, more improved variations of Android malware could be collected to investigate their attributes.

Also, a research could be conducted on identifying new malwares that doesn’t exist in the known directory.As there are a many number of permissions an application requests for during the installation , It is important to pick out the most prominent permissions among those that affect the malicious activity of the application. Feature selection is the phase that facilitates the developers with this task. The dataset generated after this feature selection accounts for the more accurate results.

In this paper, Pehlivan et al. [9] used a tool called Waikato Environment for Knowledge Analysis (WEKA) to achieve feature selection. The metrics that are employed in evaluation of effective feature selection are True Positive Rate, False Positive Rate and Precision. They also took accuracy as one of their metrics. A feature is employed in the classification dataset on the basis of Gain Attribute Evaluation attribute. The higher is this attribute, the more impact the feature keeps on the output. They considered this gain attribute to decide whether a feature is to be included in the training data set or not.

In another work, Fereidooni et al. [10] used a tool named unitPdroid written in python3 to extract the prominent features i.e.,a technique used in feature engineering . The permissions have been extracted from the class.xml file unzipped from the apk file of the android application. The permissions are then processed and modified into binary features so that it could be easier to train our classification models. This approach also incorporates the suspicious API calls as a feature that can affect the malicious activity of the application. This paper employed an approach called dalvik byte code analysis to determine if an application is malware. This analysis is based on the kind of information that the application is forwarding to the remote servers. If the application forwards the sensitive information or anything serious, it’s rated more probably as the malware.They are able to achieve results more efficiently than many conventional state of art techniques.

# **2.2 OPEN PROBLEMS IN EXISTING SYSTEM**

# **Zero-Day Exploits:** Detecting zero-day vulnerabilities and exploits remains a significant challenge. Zero-day malware attacks leverage previously unknown vulnerabilities, making them difficult to detect using traditional signature-based methods.

# **Evasion Techniques:** Malware developers are continually developing evasion techniques to bypass detection systems, such as code obfuscation, encryption, and polymorphic malware. Detecting and mitigating these techniques is an ongoing challenge.

# **Privacy-Preserving Detection:** Balancing the need for malware detection with user privacy is a complex problem. Designing detection systems that can analyze app behavior without compromising user privacy is a critical concern.

# **Resource-Efficient Solutions:** Android devices vary in terms of computational power and available resources. Creating malware detection solutions that are resource-efficient and can run on a wide range of devices, including low-end smartphones, is an ongoing challenge.

# **User-Centric Approaches:** Developing solutions that involve users actively in the detection process while minimizing their efforts and disruptions. For instance, providing users with actionable information and control over the permissions granted to apps.

# **Adversarial Machine Learning:** Malicious actors can use adversarial machine learning techniques to manipulate or evade detection models. This necessitates the development of more robust and resilient machine learning models.

# **Dynamic Analysis:** Real-time analysis of apps' behavior is crucial for malware detection. However, this can be resource-intensive and may require sophisticated analysis tools to keep up with the evolving malware landscape.

# **Scalability:** As the number of Android apps grows exponentially, the scalability of malware detection systems becomes a concern. Efficiently processing and analyzing a vast number of apps and updates in real-time is an ongoing challenge.

# **User Education:** Educating users about the risks of downloading apps from unofficial sources, the importance of keeping their devices and apps up to date, and recognizing potential threats is an open problem. User awareness is a critical component of the overall security landscape.

# **Attribution and Tracking:** Identifying the source of malware, tracking malicious actors, and taking legal actions against them can be challenging, particularly in the case of sophisticated malware campaigns.

# **False Positives and Negatives:** Reducing false positives and false negatives in malware detection remains an ongoing challenge.

# CHAPTER 3

**AIM AND SCOPE OF THE PRESENT INVESTIGATION**

**3.1 AIM:**

Detecting malicious software specifically designed to target Android devices. Protecting user data, privacy, and device integrity. Preventing financial losses and identity theft resulting from malware attacks. Maintaining the stability and performance of Android devices.

**3.2 SCOPE:**

* **Detection Techniques:** Employing various methods such as signature-based detection, anomaly-based detection, behavior-based detection, and machine learning algorithms to identify and classify malware.
* **Malware Types:** Addressing a wide range of malware types including but not limited to viruses, trojans, ransomware, spyware, adware, and phishing apps.
* **App Stores:** Assessing applications from official and third-party app stores to identify potentially harmful software.
* **Static Analysis:** Examining the code and structure of Android applications without executing them to identify suspicious patterns or behaviors.
* **Dynamic Analysis:** Running applications in controlled environments or sandboxes to observe their behavior and detect malicious activities at runtime.
* **Permission Analysis:** Scrutinizing the permissions requested by applications to detect overly permissive or suspicious behavior.
* **Network Traffic Analysis:** Monitoring network communications initiated by applications to identify suspicious connections or data exfiltration attempts.
* **Heuristic Analysis:** Using predefined rules and patterns to flag potentially malicious behaviors or code structures.
* **Zero-Day Threats:** Developing techniques to detect previously unseen or unknown malware variants through advanced anomaly detection or machine learning models trained on large datasets.
* **User Education:** Educating Android users about safe browsing habits, app installation best practices, and recognizing signs of potential malware infections.
* **Collaboration:** Collaborating with cybersecurity communities, industry partners, and researchers to share threat intelligence, improve detection capabilities, and respond effectively to emerging threats.
* **Updates and Patch Management:** Ensuring timely delivery of security updates and patches to address vulnerabilities exploited by malware.

**3.3 RISK ANALYSIS OF THE PROJECT**

***3.3.1 Feasibility Study***

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

Three key considerations involved in the feasibility analysis are

● Economical feasibility

● Technical feasibility

● Operational feasibility

***3.3.2 Economical Feasibility***

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

***3.3.3 Technical Feasibility***

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands being placed on the client. The developed system must have modest requirements, as only minimal or null changes are required for implementing this system.

***3.3.4 Operational Feasibility***

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

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

**3.4 REQUIREMENTS SPECIFICATION**

***3.4.1 Hardware specifications:***

* Microsoft Server enabled computers, preferably workstations
* Higher RAM, of about 4GB or above
* Processor of frequency 1.5GHz or above

***3.4.2 Software specifications:***

* Python 3.6 and higher
* Google colab

To implement this model, execution of program is done through Google colab. Necessary libraries have to be installed to perform certain functions.

# CHAPTER 4

# DESCRIPTION OF PROPOSED SYSTEM

Behavioral Analysis and Machine Learning: Implement a hybrid approach that combines behavioral analysis and machine learning techniques. This system will continuously monitor and analyze app behavior to detect anomalies and malicious activities.Real-time Threat Intelligence: Integrate real-time threat intelligence feeds to stay updated on the latest malware threats and attack vectors. This information can help the system adapt to new threats more effectively.User-Centric Design: Develop a user-centric system that empowers users to actively participate in the detection process. This can include user-friendly dashboards and tools for understanding and managing app permissions.

# **Key Components:**

# **Dynamic Behavior Analysis:** Implement a real-time monitoring system that tracks app behavior, including network activity, file access, and system interactions. Use machine learning to create behavioral profiles for both known and unknown apps.

# **Machine Learning Models:** Employ machine learning models, such as deep learning neural networks, to detect patterns of malicious behavior. Train these models on large datasets of benign and malicious app behavior.

# **Anomaly Detection:** Use anomaly detection algorithms to identify deviations from normal app behavior. This can help in detecting zero-day exploits and previously unseen malware.

# **Static Analysis:** Continue to use static analysis methods, such as code scanning and permission analysis, to identify known vulnerabilities and risky app behaviors during the installation phase.

# **Privacy-Preserving Analysis:** Develop mechanisms to ensure user privacy is protected during the analysis process. Implement differential privacy techniques to aggregate and anonymize user data.

# **Resource Optimization:** Create resource-efficient algorithms to ensure that the detection system has minimal impact on device performance and battery life. This is crucial for user satisfaction.

# **4.1 SELECTED METHODOLOGY OR PROCESS MODEL**

# ***4.1.1 Dataset***

Any machine learning model needs a dataset over which it can be trained. So data collection is one of the most important steps. We’ve worked on 3 different datasets and compared their result against each other

**•**  1st dataset is collected from google comprised of 70 different application each having a set of 17 permission

• 2nd is downloaded from Kaggle which has 184 different permissions or we can say

features list for 29999 apps individually.

• 3rd one is downloaded from Kaggle. It has 138047 records with each record consisting 57 columns(permissions)

For training and testing purposes, we split the dataset into two parts. We used 80% of the dataset for training the machine learning model and the remaining 20% dataset was used for testing every machine learning model and calculating the performance of each model with metrics such as accuracy, f1-score, precision and recall.

# ***4.1.2 Feature Engineering***

The feature set used for training has a big impact on machine learning. Several research have found that certain features are helpful in training machine learning-based malware classifiers. That is the reason we have used feature engineering in our implementation. In supervised learning, we will use Feature engineering, which is the process of selecting, manipulating, and changing raw data into features.

We use Feature Engineering for the following reasons:

• To remove imputation

• Handling outliers

• To achieve Normalization

• Null Value Handling

• To remove invalid data

• To achieve scaling

# **4.2 DATA PREPROCESSING**

The process of converting raw data into a comprehensible format is known as data preparation. We can’t work with raw data, thus this is a key stage in machine learning. Before using machine learning or data mining methods, make sure the data is of good quality.The purpose of data preprocessing is to ensure that the data is of good quality. The following criteria can be used to assess quality accuracy, completeness, consistency, trustworthy, understandablity.

Data Preporocessing involves the following steps:

# **• Data Cleaning:** Correcting or deleting incorrect, corrupted, improperly formatted duplicate, or incomplete data.We have removed those columns having missing values. We’ve removed any undesirable observations from our datasets, such as duplicates or irrelevant observations

**• Data Transformation:** Changing data from one format to another. For string columns and decimal columns such as price, they’re converted to binary.

# **• Data integration:** combining data from a variety of sources, including databases (both relational and non-relational), data cubes, files, and so on.

# **• Data reduction:** It is possible to reduce the amount of records, characteristics, or dimensions. It is carried out during feature selection using correlation matrix.

# 

***Fig No. 4.2.1 Data Preprocessing***

# 4.3 FEATURE SELECTION

Feature Selection is a dimensionality reduction procedure that reduces a large set of raw data into smaller groups for processing. Feature selection aims to maximise relevance while reducing repetition. To accomplish efficient data reduction, feature selection approaches can be utilised in data pre-processing. This is accomplished by Correlation Coefficient Matrix.

We need to establish an absolute value as the variable selection threshold. If the predictor variables are found to be associated, we can exclude the variable having the lowest correlation coefficient value with the target variable.

After the process of feature selection, we are left with 54 columns which we are going to consider as features that we use for training the machine learning models.We computed the correlation coefficients for each feature pair. Correlation coefficient between two variables tells us how those two variables are dependent on each other.

# 4.4 DESCRIPTION OF PROGRAMMING LANGUAGE AND SOFTWARE

We have worked on last dataset in detail and for remaining we have tested accuracy and compared its result. We made sure that the dataset contained enough examples for both the malware and benign applications. There are 41323 examples for benign applications and 96724 applications for malicious applications. So, we can say that the dataset is not skewed. Each permission column is a binary column,indicating a permission is asked or not.

# *4.4.1 Python:*

Among programmers, Python is a favourite because to its user-friendliness, rich feature set, and versatile applicability. Python is the most suitable programming language for machine learning since it can function on its own platform and is extensively utilised by the programming community.Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their own mistakes and perform routine tasks automatically. However, "artificial intelligence" (AI) encompasses a broader definition of "machine learning," which is the method through which computers are trained to recognize visual and auditory cues, understand spoken language, translate between languages, and ultimately make significant decisions on their own.

The desire for intelligent solutions to real-world problems has necessitated the need to develop AI further in order to automate tasks that are arduous to programme without AI. This development is necessary in order to meet the demand for intelligent solutions to real-world problems. Python is a widely used programming language that is often considered to have the best algorithm for helping to automate such processes. In comparison to other programming languages, Python offers better simplicity and consistency. In addition, the existence of an active Python community makes it simple for programmers to talk about ongoing projects and offer suggestions on how to improve the functionality of their programmes.

# ***4.4.2 Google colab:***

Google Colab (short for Google Colaboratory) is a cloud-based platform provided by Google that allows users to create and run Jupyter notebooks in a web-based environment. It provides free access to computational resources, including CPU and GPU, and is particularly popular in the fields of data science, machine learning, and deep learning.

# ***Key Features of Google Colab:***

# **Free and Cloud-Based:** Google Colab is free to use and hosted in the cloud. Users don't need to install any software on their local machines.

# **Jupyter Notebooks:** Users can create, edit, and run Jupyter notebooks directly in the browser. This makes it a convenient platform for data analysis, research, and collaborative work.

# **High-Performance GPUs:** Google Colab provides free GPU resources, allowing users to accelerate machine learning and deep learning tasks. This is particularly valuable for training large neural networks.

# **Integration with Google Drive:** Colab is tightly integrated with Google Drive, making it easy to store, share, and collaborate on notebooks and data files.

# **Pre-installed Libraries:** Colab comes with many pre-installed data science and machine learning libraries, similar to Anaconda, making it accessible for data analysis and model development.

# **Community and Collaboration:** Users can collaborate on notebooks in real-time, making it an excellent platform for teamwork and sharing knowledge within the data science and machine learning communities.

# **Support for Markdown:** In addition to running code, Google Colab supports Markdown, allowing users to create documentation and reports within their notebooks.

# ***4.4.3 Using Google Colab:***

To get started with Google Colab, you need a Google account. Here's a basic outline of how to use it:

# **Access Google Colab:** Open your web browser, go to Google Colab, and sign in with your Google account if you're not already logged in.

# **Create a New Notebook:** Click on "New Notebook" to create a new Jupyter notebook. You can then add code, text, and visualizations to your notebook cells.

# **Run Code:** Use the provided cells to write and run Python code. You can execute each cell independently or run the entire notebook.

# **Access GPU:** If you need GPU resources, you can enable them in the "Runtime" menu. Google Colab provides free GPU access for a limited time in each session.

# **Save and Share:** You can save your notebooks to Google Drive and share them with others for collaborative work.

# ***4.4.4 Logistic Regression :***

Logistic Regression is a predictive analytic technique built on the notion of probabilities. It is a Machine Learning algorithm that would be used for problems which involves classifying. Here, we will have a hypothesis function which will tell us the probability that a sample belongs to a particular class. In logistic regression, we use sigmoid function as hypothesis function.

# wps

# wps

Ideally we will want the logistic regression classifier to give an accurate prediction. For that, we have to find θ that minimizes the cost function J(θ). We do that using gradient descent algorithm

# wps

where wpsis the probability that i is a malicious application and wpsis the labelled class for that application.

# ***4.4.5 K-Nearest Neighbours:***

K-Nearest Neighbors is a straightforward machine learning approach that may be applied to regression problems as well as tasks involving classifying. In KNN, we look for K applications that are nearest to the given instance application. We note down the classes of all those K applications and count how many times each class appeared. The class that appeared the most will be the class of our instance application. Here the K nearest neighbours are chosen using the euclidean distance.

Here we have to choose the K value that gives the least training error rate. There isn’t any particular method to do that but when we plot error rate and k on a 2D-plane, we can take the K value that gives us least error.

# Screenshot 2024-04-03 231849

# *Fig No. 4.4.5.1* ***K-Nearest Neighbours***

# ***4.4.6 SVM Linear :***

A SVMLinear classifier is built to fit the data you supply and provide a hyperplane that f its well and classify your data into different classes. Following that, you may input some attributes to your classifier to check what the projected class is once you’ve obtained the hyperplane.

Support Vectors are the points which can be considered as edge cases. They are very nearer to the hyperplane. The two support vectors corresponding to either classes be nign and malicious respectively are equidistant to the hyperplane with maximum margin possible.

# Screenshot 2024-04-03 232414

# ***Fig No. 4.4.6.1 Support Vector Machine***

# ***4.4.7 Decision Tree:***

Decision Trees are a non-parametric supervised learning approach which can be used for classifying problems as well as regression problems. The sole objective is to construct a machine learning model that guesses the class of a given instance by learning basic decision rules from feature values.

First, we will calculate the entropy. It is also known as measure of uncertainity. Then for each attribute A, we calculate information gain. The attribute with maximum value for information gain will be selected as the root node and this process continues. The formulas for entropy and Information gain are:

# wps

# wps

# Screenshot 2024-04-03 233255

# *Fig No. 4.4.7.1 Illustration of Decision Tree*

# *4.4.8 Random Forest Method* :

Random Forest algorithm is a supervised learning method of machine learning. It produces a ”forest” out of a collection of multiple decision trees. These trees are often trained using an approach called bagging approach. The main idea of this approach is that combining many learning models enhances total output. In simple words, random forest is a method of combining many decision trees to get a more accurate and reliable prediction.

Random Forest is a Machine learning classifier that has various decision trees on different subsets of input datasets and takes the majority of them to improve the performance of the model. The more number of decision trees we take, the higher the performance of our model will be.

Decision Trees: Random Forest builds multiple decision trees during training. Each decision tree is trained on a random subset of the training data and uses a random subset of features at each split point. This randomness helps to decor relate the individual trees and reduce over fitting.

Bootstrap Aggregating (Bagging): Random Forest employs a technique called bagging, which involves sampling the training data with replacement to create multiple subsets, each of which is used to train a decision tree. This process helps to introduce diversity among the trees.

Voting Mechanism: During the prediction phase, each decision tree in the Random Forest independently predicts the target variable (in classification tasks, it votes for the class; in regression tasks, it provides a numerical prediction). The final prediction is then determined by aggregating the predictions of all the trees. In classification tasks, this is typically done by majority voting, while in regression tasks, it is often done by averaging the predictions.

# Screenshot 2024-04-04 115340

# *Fig No. 4.4.8.1 Illustration of Random Forest*

# *4.4.9 Gaussian Naive Bayes :*

A collection of classification algorithms which are based on a theorem named after Bayes together forms a Naive Bayes classifier. It isn’t a single algorithm. It is a group of algorithms sharing one common principle i.e., every pair of features which are used in classification are independent of each other.

For two events wps Bayes theorem tells that:

# wps

Using this principle for classification task, we can say that:

# wps

As we have two classes malicious and safe, we will calculate the probabilities for the application to be in malicious class and to be in safe class. The class for which the probability value is higher, is the class to which the application belongs to.

# wps

# *4.4.10 Anaconda Spyder Software:*

Spyder is an open-source integrated development environment (IDE) specifically designed for Python programming, with a focus on scientific computing, data analysis, and machine learning tasks. It features a comprehensive set of tools tailored to meet the needs of Python developers working in these domains. At its core is a feature-rich code editor that supports syntax highlighting, code completion, and code folding to enhance the coding experience. One of its standout features is the integrated IPython console, allowing for interactive code execution and real-time result visualization, along with the ability to plot graphs directly within the console. Spyder also provides a Variable Explorer for inspecting variables in memory, a built-in debugger for efficient debugging, and a profiler for optimizing code performance.

It seamlessly integrates with popular scientific libraries like NumPy, pandas, SciPy, and scikit-learn, facilitating a unified environment for scientific computing and data analysis tasks. Spyder's customizable layout, documentation integration, and cross-platform support further contribute to its appeal among researchers, data scientists, and developers seeking an efficient and powerful Python IDE.

# ***Key features of Spyder include:***

# **Editor:** Spyder features a multi-pane editor with syntax highlighting, code completion, and integrated code analysis tools to help developers write Python code efficiently.

# **Interactive Console:** Spyder includes an IPython console within the IDE, allowing developers to execute Python code interactively, view results immediately, and debug their code efficiently.

* **Variable Explorer:** The Variable Explorer in Spyder allows users to inspect and manipulate variables in memory during program execution, making it easier to debug and understand code behavior.

# **Debugger:** Spyder includes a built-in debugger that allows developers to step through code, set breakpoints, and inspect variables to identify and fix errors in their Python programs.

# **Documentation Browser:** Spyder integrates a documentation browser that provides quick access to documentation for Python libraries and modules, making it easier for developers to find information and learn about different functions and classes.

# **Integration with other tools:** Spyder seamlessly integrates with other tools commonly used in the data science and scientific computing ecosystem, such as NumPy, pandas, matplotlib, and scikit-learn.

**CHAPTER 5**

**IMPLEMENTATION DETAILS**

User Feedback and Reputation Systems:Incorporate user feedback and reputation systems to identify potentially harmful applications based on community reports or ratings. Users can report suspicious behavior, rate applications, or provide feedback on their experiences. This data can be aggregated and analyzed to flag potentially malicious applications.

App Signature Verification:Verify the digital signatures of applications to ensure they haven't been tampered with or repackaged. Legitimate developers sign their apps with cryptographic keys, and verifying these signatures can help detect unauthorized modifications or fake versions distributed by malicious actors.

System Updates and Patch Management:Regularly update the malware detection system to incorporate new threat intelligence, improve detection algorithms, and address vulnerabilities. Additionally, ensure that the underlying Android operating system is up-to-date with the latest security patches to mitigate known vulnerabilities exploited by malware.

User Education and Awareness:Educate users about safe app installation practices, such as downloading applications only from official app stores like Google Play Store, avoiding apps from unknown sources, reviewing permissions before installation, and being cautious of suspicious links or advertisements.

Continuous Monitoring and Response:Implement mechanisms for continuous monitoring of application behavior, both at runtime and post-installation. This includes real-time monitoring of system events, network traffic, and file system activities to detect and respond to emerging threats promptly.

Privacy Protection Mechanisms:Ensure that the malware detection system respects user privacy by implementing robust data protection measures. Limit the collection and storage of sensitive user data to what is strictly necessary for malware detection.

Integration with Mobile Device Management (MDM) Solutions:Integrate with Mobile Device Management solutions to enforce security policies, remotely install updates, and blacklist or quarantine suspicious applications across managed devices. This provides centralized control and visibility over the security posture of deployed Android devices.

Collaboration and Information Sharing:Foster collaboration and information sharing among security researchers, industry partners, and cybersecurity organizations to exchange threat intelligence, share best practices, and collectively combat the evolving landscape of Android malware threats.

|  |  |
| --- | --- |
| **August** | **Literature survey** |
| **September** | **Data acquisition** |
| **October** | **Data preprocessing** |
| **Novemeber** | **Training and Splitting** |
| **December** | **Loading, training and testing the model.** |
| **January** | **Predicting the output and generating the final report** |

# CHAPTER 6

# RESULTS AND DISCUSSION PERFORMANCE ANALYSIS

# 6.1 RESULTS

As android malwares are increasing day to day, we wanted to present a model that can detect malwares accurately. We considered the permissions asked by an android application as the features to our machine learning model. This comes under Static Analysis. The idea of considering these permissions as features worked well. We got good accuracy from most of the machine learning models that were implemented. The Random forest got the highest accuracy . Out of all the remaining models, SVM and KNN performed well. And Gaussian Naive Bayes model got the least accuracy among all the models that we implemented.

# Screenshot 2024-01-30 164054

# Screenshot 2024-01-30 164113

# *Fig No. 6.1.1 Best Accuracy* ***Algorithm***

Firstly, it employs supervised learning to classify applications based on labeled samples. These samples are manually labeled by security experts, indicating whether each application is benign or malicious. Through this process, the ML model can learn from the labeled data and make accurate predictions for new, unseen applications. Moreover, the system employs unsupervised learning to discover previously unknown and emerging malware. Unsupervised learning techniques enable the ML model to detect anomalies and identify potentially new forms of malware without relying solely on pre-existing knowledge. This capability is particularly important for staying ahead of cybercriminals who continually develop new and sophisticated attacks.

**6.2 PERFORMANCE ANALYSIS**

To enhance the effectiveness of the system, it regularly updates its ML models using up-to-date data from various sources, including reputable security vendors, app stores, and crowdsourcing initiatives. This continuous learning process enables the system to adapt and improve its accuracy in detecting rapidly evolving malware threats. Overall, this ML-based system for Android malware detection enables efficient and reliable identification of malicious applications, safeguarding Android users from potential security risks and ensuring the security of their devices and data.

we can see the confusion matrices for various models. Confusion matrix compares the predicted classes with actual classes of applications. We can obtain the values of True Positive, False Positive, True Negative and False Negative from these confusion matrices.

Accuracy can be computed by dividing the number of correctly classified applications by total number of applications.

# wps

Precision can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that are predicted to be in that particular class.

# wps

Recall can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that should actually be in that particular class.

# wps

F1-score is a metric involving both Precision and Recall and can be computed as:

# wps

For Sequential Neural Network, we took 3 hidden layers of length 16, 8 and 1.

We used relu activation function for the hidden layers and sigmoid function for the

output layer.

|  |  |  |
| --- | --- | --- |
| **LEARNING MODELS** | **TRAINING ACCURACY** | **TESTING ACCURACY** |
| **Logistic Regression Model** | 66.5% | 67.6% |
| **Support Vector Machine Model** | 61.6% | 71.4% |
| **Random Forest Classifier** | 66.4% | 77.6% |
| **Adaboost Classifier** | 66.5% | 67.5% |
| **Multinomial Naive Bayes** | 66.2% | 67.5% |
| **KNN Classifier** | 65.6% | 71.5% |

# CHAPTER 7

# SUMMARY AND CONCLUSIONS

# 7.1 CONCLUSION

Last but not least, the ML-based solution for Android malware detection is a great and efficient way. The technology is able to detect and categorize possible malware in Android apps with high accuracy by using ML algorithms. Users are able to safeguard their devices and data against harmful apps using this feature. This system is dependable and flexible enough to keep up with emerging threats since it can learn and adapt to new malware types regularly. In sum, our ML-based approach provides an extensive and powerful answer to the problem of Android malware detection.

But, a set of android applications operating together can carry out a malicious activity. We call them colluding apps. In this, the malicious activity is carried out by more than one application. Each application participating in collusion does a small part of the malicious action. These applications communicate with each other through covert channels. Sometimes when a malicious activity cannot be performed by a single application, it might be possible that a group of applications coordinating with each other can perform that malicious activity. This phenomenon is called Application Collusion. It is an emerging threat.

The reason behind why we are calling this as an emerging threat is because most of the android malware detectors scan the applications individually when determining whether it is a malware or not. But as the malicious activity here is being carried out by a group of applications, those traditional detectors cannot detect this. So, we need a model to detect these colluding applications. Till now, very little research has been done on this and there is scarcity for datasets

# **7.2 FUTURE WORK**

Improving the current model and investigating new approaches to increase efficiency and accuracy should be the focus of future work on the system for Android malware detection using Machine Learning (ML). To begin, in order to better enhance the detection process, researchers might investigate more complex ML techniques including ensemble approaches and deep learning. To improve classification results, more complex characteristics may be extracted from Android malware samples using deep learning methods such as Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

The second way to improve the system's performance is to use dynamic analytic methods. The technology is able to identify possible malicious activities more efficiently by monitoring the runtime behavior of Android applications and capturing real-time information about their activity. To further improve processing speed and decrease computing overhead, researchers might look into feature selection approaches to find the most important characteristics and lower the dataset's dimensionality. Last but not least, it would be beneficial to compile a big collection of malware samples, both known and unknown, tailored to Android malware detection. Training and evaluating the system under varied real-world circumstances would be much easier with such a dataset, resulting in more robust and accurate outcomes.

We are trying to create or obtain a few applications that can perform collusion, so that we can do some research on them which may eventually help in creating a model that can detect colluding applications. Firstly, we have to obtain a template for collusion. Then, we have to try to split a malicious task into various steps and make each application perform one of the steps. By doing this, we can achieve collusion.

# REFERENCES

[1] A. Arora, S. Garg, and S. K. Peddoju, “Malware detection using network traffic of analysis in android based mobile devices,” in 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 66–71, 2014.

1. A. Arora, S. K. Peddoju, and M. Conti, “Permpair: Android malware detection using permission pairs,” IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1968–1982, 2020.
2. Z. Yuan, Y. Lu, and Y. Xue, “Droiddetector: android malware characterization and detection using deep learning,” Tsinghua Science and Technology, vol. 21, no. 1, pp. 114–123, 2016.
3. S. Y. Yerima, S. Sezer, and I. Muttik, “Android malware detection using parallel machine learning classifiers,” in 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 37–42, 2014.
4. N. Peiravian and X. Zhu, “Machine learning for android malware detection using permission and api calls,” in 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300–305, 2013.
5. J. Sahs and L. Khan, “A machine learning approach to android malware detection,” in 2012 European Intelligence and Security Informatics Conference, pp. 141–147, 2012.
6. Z. Wang, J. Cai, S. Cheng, and W. Li, “Droiddeeplearner: Identifying android malware using deep learning,” in 2016 IEEE 37th Sarnoff Symposium, pp. 160–165, 2016.
7. H.-S. Ham and M.-J.Choi, “Analysis of android malware detection performance using machine learning classifiers,” in 2013 International Conference on ICT Convergence (ICTC), pp. 490–495, 2013.
8. U. Pehlivan, N. Baltaci, C. Acart¨urk, and N. Baykal, “The analysis of feature selection methods and classification algorithms in permission based android malware detection,” in 2014 IEEE Symposium on Computational Intelligence in CyberSecurity (CICS), pp. 1–8, 2014.
9. H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, “Anastasia: Android malware detection using static analysis of applications,” in 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5, 2016.
10. Doberstein, C., Charbonneau, É., Morin, G., & Despatie, S. (2022). Measuring the acceptability of facial recognition-enabled work surveillance cameras in the public and private sector. Public Performance & Management Review, 45(1), 198-227.
11. Selwyn, N., Campbell, L., & Andrejevic, M. (2023). Autoroll: Scripting the emergence of classroom facial recognition technology. Learning, Media and Technology, 48(1), 166-179.
12. Hsu, C. S., Tu, S. F., & Chiu, P. C. (2022). Design of an e-diploma system based on consortium blockchain and facial recognition. Education and Information Technologies, 1-25.
13. Parhi, M., Roul, A., Ghosh, B., & Pati, A. (2022). Ioats: An intelligent online attendance tracking system based on facial recognition and edge computing. International Journal of Intelligent Systems and Applications in Engineering, 10(2), 252-259.
14. Urquhart, L., & Miranda, D. (2022). Policing faces: the present and future of intelligent facial surveillance. Information & communications technology law, 31(2), 194-219.
15. Gupta, S., Modgil, S., Lee, C. K., & Sivarajah, U. (2023). The future is yesterday: Use of AI-driven facial recognition to enhance value in the travel and tourism industry. Information Systems Frontiers, 25(3), 1179-1195.
16. Karpagam, M., Jeyavathana, R. B., Chinnappan, S. K., Kanimozhi, K. V., & Sambath, M. (2023). A novel face recognition model for fighting against human trafficking in surveillance videos and rescuing victims. Soft Computing, 27(18), 13165-13180.
17. McElroy, E., & Vergerio, M. (2022). Automating gentrification: Landlord technologies and housing justice organizing in New York City homes. Environment and Planning D: Society and Space, 40(4), 607-626.
18. Zhang, X., Zhang, X., & Dolah, J. B. (2022). Intelligent Classroom Teaching Assessment System Based on Deep Learning Model Face Recognition Technology. Scientific Programming, 2022.
19. Revathy, G., Raj, K. B., Kumar, A., Adibatti, S., Dahiya, P., & Latha, T. M. (2022). Investigation of E-voting system using face recognition using convolutional neural network (CNN). Theoretical Computer Science, 925, 61-67.

# APPENDIX

# **SOURCE CODE**

!pip install google-play-scraper #for scraping info about apk files from google repository

!pip install androguard #for analyaing apk

!pip install scikit-plot #for plotting from sklearn

!pip install pyfiglet #importing for designing the CLI

#imported necessary libraries

from IPython.display import Javascript

Defresize\_colab\_cell(): #to increase size of colab cell

display(Javascript('google.colab.output.setIframeHeight(0,true,{maxHeight: 10000})'))

get\_ipython().events.register('pre\_run\_cell', resize\_colab\_cell)

import warnings #to supress warnings so that if any uncritical warningss will not show while executing

warnings.filterwarnings("ignore", category=DeprecationWarning)

from google.colab import drive #mounting google drive for taking dataset and other files from google drive

drive.mount('/content/drive')

from androguard.misc import AnalyzeAPK

import androguard

import pickle

from google\_play\_scraper import app

import pandas as pd

import numpy as np

import IPython

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#import shap

import scikitplot as skplt

import pickle

import pyfiglet

import sys

from termcolor import colored, cprint

from sklearn import model\_selection

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, accuracy\_score, roc\_auc\_score, precision\_score, roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.metrics import ConfusionMatrixDisplay,confusion\_matrix

#from sklearn.metrics import plot\_confusion\_matrix

import numpy as np

train\_acc = []

test\_acc = []

train\_prec = []

test\_prec = []

#Reading data into a variable

import pandas as pd

df = pd.read\_csv("/content/drive/MyDrive/final project-20230922T140331Z-001/final project/new\_permissions.csv") #reading the dataset for training

df.head()

df.tail()

df=df.dropna()

df.tail()

df.columns.isnull()

df.info()

for i in df.columns: #list all columns and their default values

print("{} : {}".format(i,list(pd.unique(df[i]))))

target = df.pop("LABEL")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df.iloc[:,:-1],target)

x\_train.shape

y\_train.shape

x\_train.shape, x\_test.shape, y\_train.shape, y\_test.shape

class ModelTraining:

def proj\_banner(self): #function for showing banner

ascii\_banner = pyfiglet.figlet\_format(" Malware")

ascii\_banner2 = pyfiglet.figlet\_format(" Predictor")

te = "\n © IIITMK, Thiruvananthapuram © packed by \n\n\n"

a111 = colored(ascii\_banner, 'magenta', attrs=['blink'])

a122 = colored(ascii\_banner2, 'magenta', attrs=['blink'])

co = colored(te, 'red', attrs=['blink'])

print(a111)

print(a122)

print(co)

def LogisticRegression\_Model(self,x\_train,x\_test,y\_train,y\_test): #function for logistic regression

from sklearn.linear\_model import LogisticRegression

model1 = LogisticRegression(solver='lbfgs', max\_iter=1000)

model1.fit(x\_train, y\_train)

y\_pred1 = model1.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* Logistic Regression \*\*\*\*"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred1)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred1)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model1.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr1, tpr1, thresholds1 = roc\_curve(y\_test, y\_pred1)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr1, tpr1)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

y\_pred1\_train = model1.predict(x\_train)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred1\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred1\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model1.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t, tpr\_t, thresholds\_t = roc\_curve(y\_train, y\_pred1\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t, tpr\_t)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred1) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model1.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show() # Save the model

filename='/content/drive/MyDrive/ColabNotebooks/logistic\_regression\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model1, file)

p\_text = "\nROC Curve :\n" # plotting ROC curve graph

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Calculate ROC curve

fpr\_t, tpr\_t, thresholds\_t = roc\_curve(y\_train, y\_pred1\_train) # Plot ROC curve

plt.plot(fpr\_t, tpr\_t)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Logistic Regression')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show()

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def SVM\_Model(self,x\_train,x\_test,y\_train,y\_test): #function for SVM

from sklearn.svm import SVC

model2 = SVC(probability= True)

model2.fit(x\_train,y\_train)

y\_pred2 = model2.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* Support Vector Machine \*\*\*\*"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred2)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred2)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model2.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr2, tpr2, thresholds2 = roc\_curve(y\_test, y\_pred2)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr2, tpr2)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

y\_pred2\_train = model2.predict(x\_train)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred2\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred2\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model2.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t2, tpr\_t2, thresholds\_t2 = roc\_curve(y\_train, y\_pred2\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t2, tpr\_t2)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred2) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model2.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show() # Save the model

filename = '/content/drive/MyDrive/Colab Notebooks/svm\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model2, file)

p\_text = "\nROC Curve :\n" # plotting ROC curve graph

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text)

# Calculate ROC curve

fpr\_t2, tpr\_t2, thresholds\_t2 = roc\_curve(y\_train, y\_pred2\_train) # Plot ROC curve

plt.plot(fpr\_t2, tpr\_t2)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Support Vector Machine')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show()

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def RandomForest\_Classifier(self,x\_train,x\_test,y\_train,y\_test): #function for random forest classifier

from sklearn.ensemble import RandomForestClassifier

model3 = RandomForestClassifier()

model3.fit(x\_train,y\_train)

y\_pred3 = model3.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* Random Forest Classifier \*\*\*\*\n"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred3)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred3)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model3.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr3, tpr3, thresholds3 = roc\_curve(y\_test, y\_pred3)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr3, tpr3)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

y\_pred3\_train = model3.predict(x\_train)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred3\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred3\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model3.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t3, tpr\_t3, thresholds\_t3 = roc\_curve(y\_train, y\_pred3\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t3, tpr\_t3)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred3) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model3.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show()

p\_text = "\nROC Curve :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

plt.plot(fpr\_t3, tpr\_t3)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Random Forest Classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show() # Save the model

filename = '/content/drive/MyDrive/Colab Notebooks/random\_forest\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model3, file)

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def AdaBoost\_Classifier(self,x\_train,x\_test,y\_train,y\_test): #function for adaboost clasifier

from sklearn.ensemble import AdaBoostClassifier

model4 = AdaBoostClassifier()

model4.fit(x\_train,y\_train)

y\_pred4 = model4.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* AdaBoost Classifier \*\*\*\*"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred4)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred4)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model4.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr4, tpr4, thresholds4 = roc\_curve(y\_test, y\_pred4)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr4, tpr4)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

y\_pred4\_train = model4.predict(x\_train)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred4\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred4\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model4.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t4, tpr\_t4, thresholds\_t4 = roc\_curve(y\_train, y\_pred4\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t4, tpr\_t4)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred4) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model4.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show() # Save the model

filename = '/content/drive/MyDrive/Colab Notebooks/adaboost\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model4, file)

p\_text = "\nROC Curve :\n" # plotting ROC curve graph

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Calculate ROC curve

fpr\_t4, tpr\_t4, thresholds\_t4 = roc\_curve(y\_train, y\_pred4\_train) # Plot ROC curve

plt.plot(fpr\_t4, tpr\_t4)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for AdaBoost Classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show()

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def Multinomial\_NaiveBayes(self,x\_train,x\_test,y\_train,y\_test): #function for naive bayes

from sklearn.naive\_bayes import MultinomialNB

model5 = MultinomialNB()

model5.fit(x\_train,y\_train)

y\_pred5 = model5.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* Multinomial Naive Bayes \*\*\*\*"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred5)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred5)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model5.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr5, tpr5, thresholds5 = roc\_curve(y\_test, y\_pred5)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr5, tpr5)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

y\_pred5\_train = model5.predict(x\_train)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred5\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred5\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model5.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t5, tpr\_t5, thresholds\_t5 = roc\_curve(y\_train, y\_pred5\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t5, tpr\_t5)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred5) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model5.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show() # Save the model

filename = '/content/drive/MyDrive/Colab Notebooks/multinomial\_nb\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model5, file)

p\_text = "\nROC Curve :\n" # plotting ROC curve graph

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Calculate ROC curve

fpr\_t5, tpr\_t5, thresholds\_t5 = roc\_curve(y\_train, y\_pred5\_train) # Plot ROC curve

plt.plot(fpr\_t5, tpr\_t5)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Multinomial Naive Bayes')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show()

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def KNN\_Classifier(self,x\_train,x\_test,y\_train,y\_test): #function for KNN model

from sklearn.neighbors import KNeighborsClassifier

model6 = KNeighborsClassifier()

model6.fit(x\_train,y\_train)

y\_pred6 = model6.predict(x\_test)

p\_text = "\n\n\n\*\*\*\* KNN Classifier \*\*\*\*"

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

print(prin\_text)

p\_text = "\nTraining :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_test, y\_pred6)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_test, y\_pred6)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

train\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_test, model6.predict\_proba(x\_test)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr6, tpr6, thresholds6 = roc\_curve(y\_test, y\_pred6)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr6, tpr6)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

o\_banner = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold','blink'])

print(o\_banner\_text)

p\_text = "\nTesting :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

y\_pred6\_train = model6.predict(x\_train)

p\_text = "Accuracy score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = accuracy\_score(y\_train, y\_pred6\_train)\*100

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_acc.append(p\_ans)

p\_text = "Precision score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = precision\_score(y\_train, y\_pred6\_train)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

test\_prec.append(p\_ans)

p\_text = "ROC AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = roc\_auc\_score(y\_train, model6.predict\_proba(x\_train)[:,1])

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

fpr\_t6, tpr\_t6, thresholds\_t6 = roc\_curve(y\_train, y\_pred6\_train)

p\_text = "AUC score : "

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

p\_ans = auc(fpr\_t6, tpr\_t6)

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans) # Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred6) # Print Confusion Matrix

p\_text = "\nConfusion Matrix :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Display Confusion Matrix

disp = ConfusionMatrixDisplay(conf\_matrix, display\_labels=model6.classes\_)

disp.plot(cmap='Blues', values\_format='d') # Show the plot

plt.show() # Save the model

filename = '/content/drive/MyDrive/Colab Notebooks/knn\_model.pkl'

with open(filename, 'wb') as file:

pickle.dump(model6, file)

p\_text = "\nROC Curve :\n" # plotting ROC curve graph

prin\_text = colored(p\_text, 'cyan', attrs=['bold', 'blink'])

print(prin\_text) # Calculate ROC curve

fpr\_t6, tpr\_t6, thresholds\_t6 = roc\_curve(y\_train, y\_pred6\_train) # Plot ROC curve

plt.plot(fpr\_t6, tpr\_t6)

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for KNN classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

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

plt.grid(True)

plt.show()

o\_banner = "\n------------END------------\n"

o\_banner\_text = colored(o\_banner, 'white', attrs=['bold', 'blink'])

print(o\_banner\_text)

def evaluateAPK(self): #model evaluation function

content=input("Enter path to the apk file:") #inputtting location of apk file from user #spinner = Halo(text='', spinner='moon',interval=100) #spinner.start()

a,d,dx=AnalyzeAPK(content)

filename=a.get\_package() #get package name of apk

inp=pd.read\_csv("/content/drive/MyDrive/finalproject/finalproject/Feature\_selection.csv") #loads a empty dataset

df5=pd.DataFrame(inp) #loading csv to dtataframe

# df5=df5.drop(df5.iloc[:,0:15],axis=1) #selelcting the needed colmns from dataset

df5=df5.drop(df5.iloc[:,-3:],axis=1)

#df5=df5.rename(columns={'rating\_count': 'rating\_number'}) #changing the name of dataset to make it same as original dataset

df5=df5.append(pd.Series(int(0), index=df5.columns), ignore\_index=True) #adding a new row with all cells are 0

perm=a.get\_permissions() #androguard function for getting permission

for item in df5.columns: #comparing the list of permissions with column name and if same change the value in cell as 1 and break. if not same set it back as 0

for i in perm:

if i==item:

df5.at[0,item]=1

break

else:

df5.at[0,item]=0

df5.insert(len(df5.columns), 'LABEL','0')

#df5.at[0,'rating\_number']=float('4')

flag=0

for item in df5.columns: #to know howmany vaild features ar added

if df5.at[0,item]==1:

flag+=1

p\_text = '\n'+str(flag)+' features are added\n'

prin\_text = colored(p\_text, 'blue', attrs=['bold','blink'])

print(prin\_text)

df5.to\_csv('/content/drive/MyDrive/Colab Notebooks/extractor.csv',index=False) #saving the updataed data to csv

import joblib

list3=['rf']

for item in list3: #running all the model from saved model files using pickle module and predicting whether it malware or not

list2='/content/drive/MyDrive/Colab Notebooks'+str(item)+'.sav'

#Loading the saved model with joblib

pipe = joblib.load(list2) # New data to predict

pr = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/extractor.csv')

#pred\_cols = list(pr.columns.values)[:-1]

df2=pd.DataFrame(pr)

pre=df2.iloc[:, :-1]

Pre # apply the whole pipeline to data

pred21 =pipe.predict(pre)

"""if pred21==1:

p\_text = 'According to '+item+' this application tend to be : Malware'

prin\_text = colored(p\_text, 'red', attrs=['bold','blink'])

print(prin\_text)

elif pred21==0:

p\_text = 'According to '+item+' this application tend to be : Benign'

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)"""

try:#check for application details available from playstore

result = app(filename,lang='en',country='us')

p\_text = 'Title :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = result['title']

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'App ID :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = result['appId']

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Avg Rating :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = str(result['score'])

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Developer :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = result['developer']

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Developer Address :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = result['developerAddress']

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Genre :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = result['genre']

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans) #print(result)

except Exception: #if exception occurs this art is run

p\_text = 'Title :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = a.get\_app\_name()

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'App ID :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = a.get\_package()

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Avg Rating :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = 'Not Found'

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Developer :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = 'Not Found'

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Developer Address :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = 'Not Found'

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = 'Genre :'

prin\_text = colored(p\_text, 'magenta', attrs=['bold','blink'])

p\_ans = 'Not Found'

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

p\_text = '\n\nThis application found to be a : '

prin\_text = colored(p\_text, 'blue', attrs=['bold','blink'])

if pred21==1:

p\_ans = ' Malware '

prin\_ans = colored(p\_ans, 'red', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

print("\n\n") #display(IPython.display.Audio("Malware\_Project/Audio/en\_malware.mp3",autoplay=True)) #playing audio that this app is maware

elif pred21==0:

p\_ans = ' Benign '

prin\_ans = colored(p\_ans, 'green', attrs=['bold','blink'])

print(prin\_text,prin\_ans)

print("\n\n") #display(IPython.display.Audio("Malware\_Project/Audio/en\_benign.mp3",autoplay=True)) #playing audio that this app is maware #spinner.stop()

def Observations(self): #for plotting the outputs

p\_text = "\nAccuracy :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

barWidth = 0.25

fig = plt.subplots(figsize =(12, 8))

br1 = np.arange(len(train\_acc))

br2 = [x + barWidth for x in br1]

plt.bar(br1, train\_acc, color ='r', width = barWidth,

edgecolor ='grey', label ='Train')

plt.bar(br2, test\_acc, color ='g', width = barWidth,

edgecolor ='grey', label ='Test')

plt.xlabel('Machine Learning Models', fontweight ='bold', fontsize = 15)

plt.ylabel('Accuracy', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(train\_acc))],

['Logistic Regression', 'SVM', 'Random Forest', 'Adaboost', 'Multinomial NB', 'KNN'])

plt.legend()

plt.show() #printing the accuracy comparison bar graph

p\_text = "\n\n\nPrecision :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

barWidth = 0.25

fig = plt.subplots(figsize =(12, 8))

br3 = np.arange(len(train\_prec))

br4 = [x + barWidth for x in br3]

plt.bar(br3, train\_prec, color ='r', width = barWidth,

edgecolor ='grey', label ='Train')

plt.bar(br4, test\_prec, color ='g', width = barWidth,

edgecolor ='grey', label ='Test')

plt.xlabel('Machine Learning Models', fontweight ='bold', fontsize = 15)

plt.ylabel('Precision', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(train\_prec))],

['Logistic Regression', 'SVM', 'Random Forest', 'Adaboost', 'Multinomial NB', 'KNN'])

plt.legend()

plt.show() #printing prcision comparison bargraph

def main(): #main functionwhich uses the clas w built

cl = ModelTraining()

cl.proj\_banner()

while True:

try:

p\_text = "\n\n\n1. Train Using Maching Learning Models\n2. Evaluate the model with new dataset\n3. Observations\n4. Exit\n\n"

prin\_text = colored(p\_text, 'blue', attrs=['bold','blink'])

print(prin\_text)

choice = int(input("Enter your choice :"))

if choice == 1:

while True:

p\_text = "\n\n\nSelect the Machine Learning Model you wanted to Train :\n"

prin\_text = colored(p\_text, 'cyan', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 1. Logistic Regression Model"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 2. Support Vector Machine Model"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 3. Random Forest Classifier"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 4. Adaboost Classifier"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 5. Multinomial Naive Bayes"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 6. KNN Classifier"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 7. All Models Together"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

p\_text = " 8. Go Back\n"

prin\_text = colored(p\_text, 'green', attrs=['bold','blink'])

print(prin\_text)

ch = int(input("Enter your choice :"))

if ch == 1:

cl.LogisticRegression\_Model(x\_train, x\_test, y\_train, y\_test)

elif ch == 2:

cl.SVM\_Model(x\_train, x\_test, y\_train, y\_test)

elif ch == 3:

cl.RandomForest\_Classifier(x\_train, x\_test, y\_train, y\_test)

elif ch == 4:

cl.AdaBoost\_Classifier(x\_train, x\_test, y\_train, y\_test)

elif ch == 5:

cl.Multinomial\_NaiveBayes(x\_train, x\_test, y\_train, y\_test)

elif ch == 6:

cl.KNN\_Classifier(x\_train, x\_test, y\_train, y\_test)

elif ch == 7:

cl.LogisticRegression\_Model(x\_train, x\_test, y\_train, y\_test)

cl.SVM\_Model(x\_train, x\_test, y\_train, y\_test)

cl.RandomForest\_Classifier(x\_train, x\_test, y\_train, y\_test)

cl.AdaBoost\_Classifier(x\_train, x\_test, y\_train, y\_test)

cl.Multinomial\_NaiveBayes(x\_train, x\_test, y\_train, y\_test)

cl.KNN\_Classifier(x\_train, x\_test, y\_train, y\_test)

else:

p\_text = "\nGoing Back To Main Menu!\n"

prin\_text = colored(p\_text, 'red', attrs=['bold','blink'])

print(prin\_text)

break

elif choice == 2:

cl.evaluateAPK()

elif choice == 3:

cl.Observations()

else:

p\_text = "\nExiting....Thank you for using!\n"

prin\_text = colored(p\_text, 'red', attrs=['bold','blink'])

print(prin\_text)

break

except Exception as e:

print(e)

if \_\_name\_\_ == "\_\_main\_\_":

main() #running the CLI

# **SCREEN SHOTS**

# **Screenshot 2024-04-07 004338**

***Fig No. B.1 Malware Predictor***

# Screenshot 2024-04-07 004358

# *Fig No. B.2 List Of Learning Models*

# Screenshot 2024-04-07 004816Screenshot 2024-04-07 004837

# *Fig No. B.3 Logistic Regression Model*

# *Screenshot 2024-04-07 004856Screenshot 2024-04-07 004919*

# *Fig No. B.4 Support Vector Machine Model*

# *Screenshot 2024-04-07 004934Screenshot 2024-04-07 004956*

# *Fig No. B.5 Random Forest Classifier*

# *Screenshot 2024-04-07 005015Screenshot 2024-04-07 005030*

# *Fig No. B.6 Adaboost Classifier*

# *Screenshot 2024-04-07 005048Screenshot 2024-04-07 005101*

# *Fig No. B.7 Multinomial Naive Bayes*

# *Screenshot 2024-04-07 005120Screenshot 2024-04-07 005134*

# *Fig No. B.8 KNN Classifier*

# *Screenshot 2024-02-08 105811*

# *Fig No. B.9 Accuracy Bar Graph*

# *Screenshot 2024-02-08 105829*

# *Fig No. B.10 Precision Bar Graph*

# *WhatsApp Image 2024-04-08 at 00.35.11_fe7ec01b*

# *Fig No. B.11 Home Page*

# *WhatsApp Image 2024-04-08 at 00.33.49_ec46d5d1*

# *Fig No. B.12 Safe Url*

# *WhatsApp Image 2024-04-08 at 00.34.04_49702c27*

# *Fig No. B.13 Malware Url*

# C.RESEARCH PAPER

# *Screenshot 2024-04-07 164221*

# *Screenshot 2024-04-07 164243*

# *Screenshot 2024-04-07 164300*

# *Screenshot 2024-04-07 164319*

# *Screenshot 2024-04-07 164337*

# *Screenshot 2024-04-07 164401*

# *Screenshot 2024-04-07 164425*

# D.CERTIFICATE

