# PERFORMANCE ENHANCEMENT IN ANDROID MALWARE DETECTION USING DIMENTIONALITY REDUCTION

**A** DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

**MASTER OF TECHNOLOGY**
**IN**
**INFORMATION SYSTEMS**

Submitted by

**SHAURYA SINGH**
**2K21/ISY/22**

Under the supervision of

**MR. RAHUL GUPTA**

(Assistant Professor)


**INFORMATION TECHNOLOGY**
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana load, Delhi-110042

**JUNE, 2023**

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

# CANDIDATE'S DECLARATION

I/We, **SHAURYA SINGH**, Roll No(s). **2K21/ISY/22** student of M.Tech. INFORMATION SYSTEMS, hereby declare that the project report titled "**Performance Enhancement in Android Malware Detection using Dimensionality Reduction**" which is submitted by me/us to the Department of INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associate ship, Fellowship or other similar title or recognition.

Place: Delhi SHAURYA SINGH

Date: 27th May, 2023

INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

## CERTIFICATE

I hereby certify that the Project report titled " Performance Enhancement in Android Malware Detection using Dimensionality Reduction " which is submitted by SHAURYA SINGH, 2K21/ISY/22, INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date: 27th May, 2023

**MR. RAHUL GUPTA**

**SUPERVISOR**
**(Asst. Professor)**

# ABSTRACT

Android is one of the most common operating systems in smart phones in current scenario. People are investing in smart phones and which in turn is leading to an exponential increase in android users worldwide. As the number of android users are increasing day by day, there is a wide range of target audience for malicious applications and related issues. By using Android Malware detection, I can detect such malicious applications and prevent the users from becoming the target of such malicious applications.

There are various strategies that have been already proposed for detecting Android malware have been proposed in the literature, there is still a need for attribute selection methods to be employed in Android malware detection systems. A machine learning-based malware detection method is suggested in this work to differentiate Android malware from benign apps. As the dimensionality reduction stage is introduced in this study to reduce the dimensions in a dataset and to decrease the time of training phase. When the results are obtained, it is observed that the accuracy of the mode is also improved by using dimensionality reduction.

# ACKNOWLEDGEMENT

I want to offer my thanks to my mentor Mr. Rahul Gupta for his constant support, patience and for making a very positive atmosphere for finishing my report. His motivation and help contributed tremendously to the successful completion of the report. He additionally assisted me with lot of Research, and I came to know about such countless new things. Last but not the least, I want to express my gratitude towards him for monitoring the progress and giving me significant feedback.

CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

## 1.1 GENERAL

Since the first known computer virus was discovered in 1970, albeit an experimental one, malicious software, often known as malware, has been a component of computing. Since then, malware has developed both in terms of diversity and the necessity to categorize its applications according to how they spread and/or how they harm their victims. The ongoing conflict between malware developers, who continuously innovate by finding new attack vectors and techniques, and security professionals, who are continually developing new kinds of malware defense and prevention. The public's ability to access the Internet increased its importance as a tool for a range of distant jobs, notably communication through electronic mail and social networking sites, personal banking, and other activities . For attackers, these jobs are particularly alluring since they include sensitive information. Additionally, virus creators can distribute harmful software very effectively on the Internet to gather the information [3].

Despite the fact that most attacks focus on laptops and desktops, the growing number of smartphones and resulting increase in their popularity made these mobile devices an interesting attack vector for malware makers [3]. In fact, Statista predicts that by 2023, there will be 7.33 billion mobile and smartphone users worldwide, up from 3.8 billion in 2016 [4]. This is a result of their portability, high computational power, capacity to connect to the Internet everywhere, and ability to extend functionality by downloading software applications from application stores.

One of the fastest growing smart phone operating systems is android which has

occupied around more than 80% of the shares in market due to its open-source approach, providing a free Integrated Development Environment (IDE). Unlike its primary rival iOS, which is extremely rigid and thorough, Android has no hardware constraints and a more flexible procedure for app approval. Additionally, publishing iOS programs requires paying a yearly subscription fee, providing another barrier to entry that provides insight into why Android is more popular. The operating system for Android also permits the installation of programs from unreliable sources and third-party application providers. There are many services and millions of applications for android operating systems with various functionalities like messaging, callings, applications based on locations, payment services, shopping applications and gaming etc. As a result, malware detection techniques of Android application stores are less effective since they are unable to keep up with the rate of malware development.

As the number of android users are increasing every coming day, it attracts the cybercriminals to a large extent. Cyber criminals develop malicious applications, or they perform activities that violates cyber security. According to the studies in 2017 malware count in android operating systems reaches around 20 lakhs.

| Country size | Android OS | iOS |
|---|---|---|
| US | 41.65% | 58.17% |
| Canada | 50.81 % | 48.83% |
| China | 78.31% | 20.68% |
| India | 95.07% | 3.54% |
| UK | 53.24% | 46.53% |
| Germany | 71.71% | 27.69% |

Table 1: Worldwide Smartphone Operating System Market Share 2020

To decide if a submitted application will be granted admission into the store, these businesses typically have an application screening procedure. For instance, the Google Play Store's application screening procedure involves examining to determine if it adheres to

Android base-level security, then going through an automated review and a manual review. Developers must also abide by the developer policies for their applications to be distributed. If these rules are broken, the developer gets notified that the application is not published, and once it fixes the problems, it can be resubmitted for evaluation. In addition, violations of the developer policies may result in the suspension of the application. Malware or other persistent violations may also result in the deactivation of developer-owned accounts. When an application is approved for review and added to the Google Play Store, Google Play Protect continuously monitors it to ensure it is secure for Android devices and uses machine learning to determine malicious software and activities like fraud and impersonating someone else.

There are different ways in which malicious applications are being developed. Based on the studies on how attacks are performed in current scenarios and present literature information.

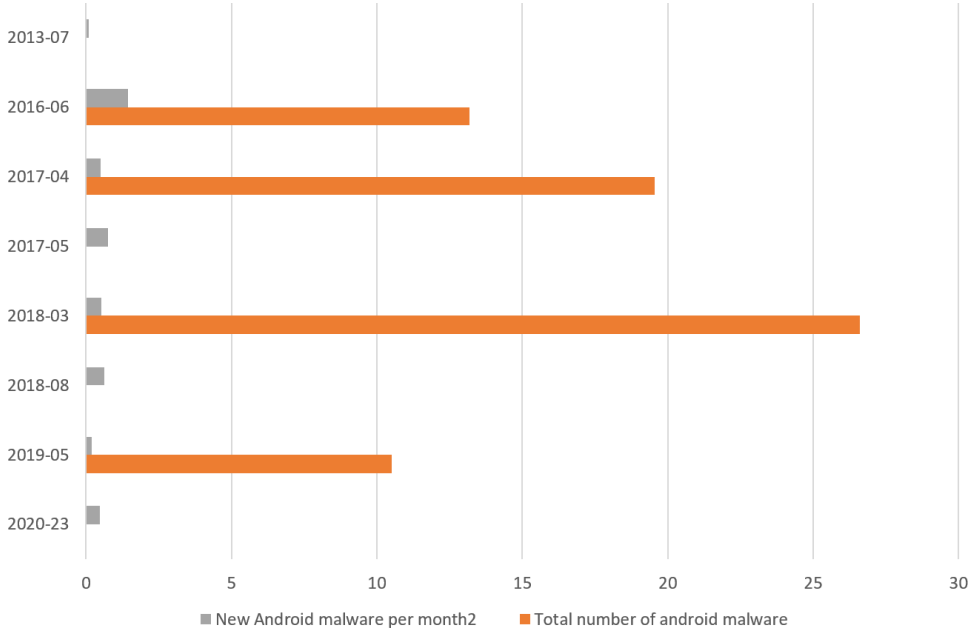


Fig. 1: Global Android Malware Statistics 2020

## 1.1 TYPES OF ATTACKS

Following lists the types of malwares commonly used for criminal attacks:

Trojan:
Trojan is malicious set of code of software that appears to be authentic and safe but have harmful intensions to control the system activities.

Worms:
Another type of malware is warms. Worms replicates itself and then it spreads over the system of networks.

Backdoor:
Backdoors are the part of programs of software that are hidden. Upon installing the software, such programs takes unauthorized access to the system and controls the system remotely by the attackers.

Botnets: Botnets are the network of infected or malicious systems or computers which is controlled by the master attacking party.

Covert channel:
The vulnerable devices that allow the contained information to leak between the operations or processes.

Automatic Calls and SMS:
There are few special numbers available which upon calling does criminal activities like increasing the users bill etc.

There are around 50+ variants of malwares present of each type of malware.

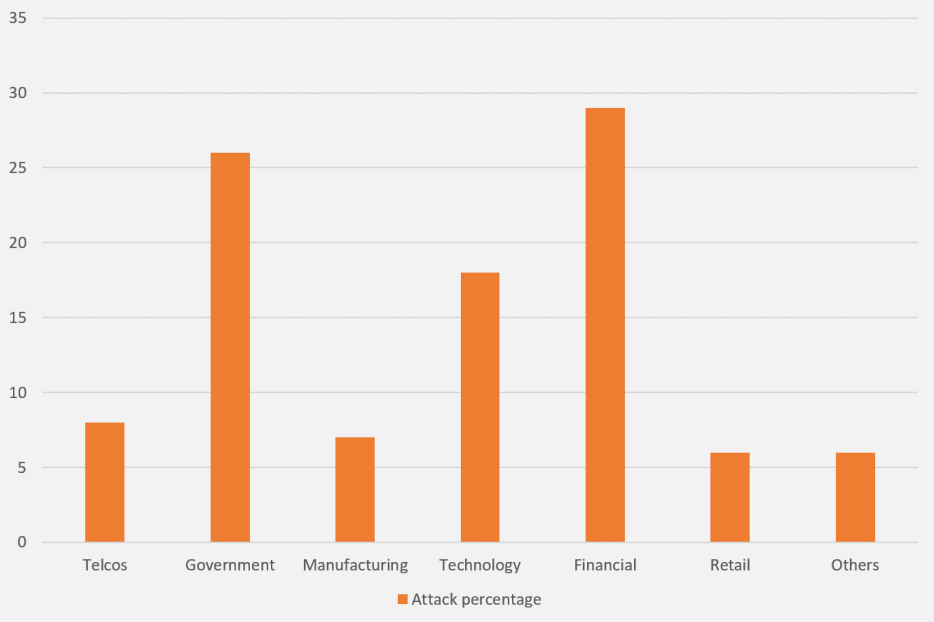


Fig.2: Malware attacks by industries

To help reduce the spread of Android malwares, several machine learning algorithms that are provided with characteristics that are extracted through application of static analysis to a significant number of Android applications offered will be tested in this thesis.

## 1.2 OBJECTIVES

Unlike the majority of machine learning-based Android malware detection algorithms described in the literature, this research is conducted using real, up-to-date data on a large scale. In order to improve Android malware detection, the main objective of this thesis is to discover whether using machine learning techniques may be a successful strategy.

Additionally, the application of LDA, PCA, and t-SNE dimensionality reduction algorithms is demonstrated. The accuracy of the model's classification of the programs into legitimate software and malicious software is then evaluated by applying the reduced

dataset to machine learning methods. To comprehend and evaluate the effectiveness of the algorithms, a thorough comparison of the findings has been made after applying the machine learning models to the dataset.

## 1.4 STRUCTURE

This thesis' remaining sections are organized as follows: An overview of the mobile malware detection architecture, with a focus on the Android platform, will be presented in Chapter 2. This will be followed by an overview of the security and application components of the Android platform, and finally a review of the literature on static, dynamic, and hybrid-analysis-based machine learning detection approaches.

In Chapter 3, the technique suggested for the experimental procedures in this thesis, the justification for its acceptance, and the results are all explained. The outcomes of each experimental procedure will be presented and discussed in Chapter 4. At the end, of thesis, some topics for more research has been proposed.

# CHAPTER 2

# LITERATURE SURVEY

Mobile computing technology has been advancing quickly due to how desirable these features are to end users, and with the introduction of smartphones, this trend is continuing due to smartphones' ability to perform data processing tasks usually carried out by desktop computers while also offering a mobile platform [4]. Mobile applications provide a wide range of services, including trade, personal banking, health and fitness, messaging, and payment gateways. Device stores are primarily responsible for managing these software programs. Mobile devices are a great target for malware writers because of the enormous amount of data they hold, most of it is private and sensitive data. The most common threats from which Android platforms suffer include SMS trojans, spyware, botnets, aggressive adware, and ransomware. It is as a result of the fact that, out of the two operating system platforms that are most frequently used, iOS and Android, Android platforms are the most generally utilized. In the current market, Android makes up to 76.4 percent of all mobile operating systems. This is due to Android OS's open-source philosophy, which permits mobile industry manufacturers to use Android OS as the foundation for their individual OS versions. Various alternatives exist to Google Play. This gives malware attackers various places to propagate malicious software, especially for those that tend to be a benign program. For instance, third party stores or it may also be downloaded directly from the websites. Due to the convergence of these variables, there has been a huge increase in Android malware, making it impossible for conventional signature-based detection techniques to handle the growing volume. On order to stop the spread of malware in Android, it is crucial to build automatic and sophisticated methods of detection. By utilizing their potential for efficient malware identification and prevention on Android, machine learning techniques present a possible path toward achieving this goal.

## 2.1 ANDROID PLATFORM APPLICATION COMPONENTS AND SECURITY

Android is a mobile operating system that runs on an open-source Linux platform. It includes a number of crucial elements. Additionally, Android provides an application environment where programmers can set up their original applications. The AndroidManifest.xml file, activities, services, and broadcast receivers are the four main parts of these applications. These programs are created and then assembled into an Android application package (APK) file. Resources, assets, the AndroidManifest.xml file, and files with a.dex extension are all included in the APK file.

The AndroidManifest.xml file fulfils a significant function by supplying essential data about an application to numerous entities, including the Android operating system, Google Play, and the Android build tools. It is essential for guiding the system in its use of the activities, services, and content providers available within the application. The necessary permissions needed for the application to run are also specified [14].

Activities within an application can be thought of as discrete screens that act as windows for the application's User Interface (UI). Depending on its requirements, an application could include a number of different activities. In order to enable user interaction with the program, activities are therefore essential.

Services are parts of a program that can do actions like network operations in the background without being explicitly apparent to the user. They are capable of operating continuously for a long time. Services, however, can also carry out foreground duties, such playing music while a user uses another application. In these circumstances, it is crucial for the service to alert the user when it is in use [19]. Applications use receivers as a way to access the messages or events that interest them for use in later actions. For instance, the application that started an event might send out a broadcast message when that event occurs, such as when a file download is finished. All applications that have indicated a wish to receive such notifications are informed by this broadcast message. As a result, using the

broadcast message that was received, the recipient programs can do their intended tasks. From a security standpoint, the Android platform makes sure that users are protected by giving each application a distinct User ID (UID) and requiring that they run in separate processes. Additionally, apps run in their own application sandboxes that limit their ability to interact with other applications and provide them only a limited amount of access to the operating system. This security feature protects the operating system against potentially harmful programs while also protecting apps [21]. As a result, applications need to consciously share resources and data. This is made possible by the use of permissions, where an application expressly states that it needs a certain set of rights in order to access resources and device features outside of its sandbox [22]. Additionally, Android uses a technology called secure inter-process communication that enables applications that are executing in different processes to safely communicate with one another.

The security and privacy of Android users are crucially dependent on permissions. They are in charge of regulating how Android applications can access and make use of user data (including contacts and emails) and system features (like the camera and Near Field Communication). The fact that no application is by default given permission to carry out actions that could possibly harm the user, other applications, or the operating system highlights the importance of permissions in ensuring the security of the Android platform.

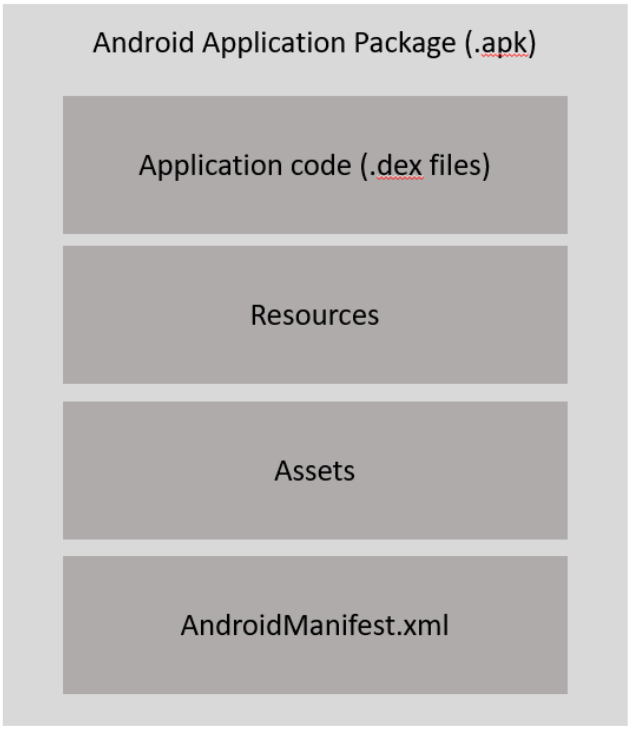


Fig 3 Android application package contents.

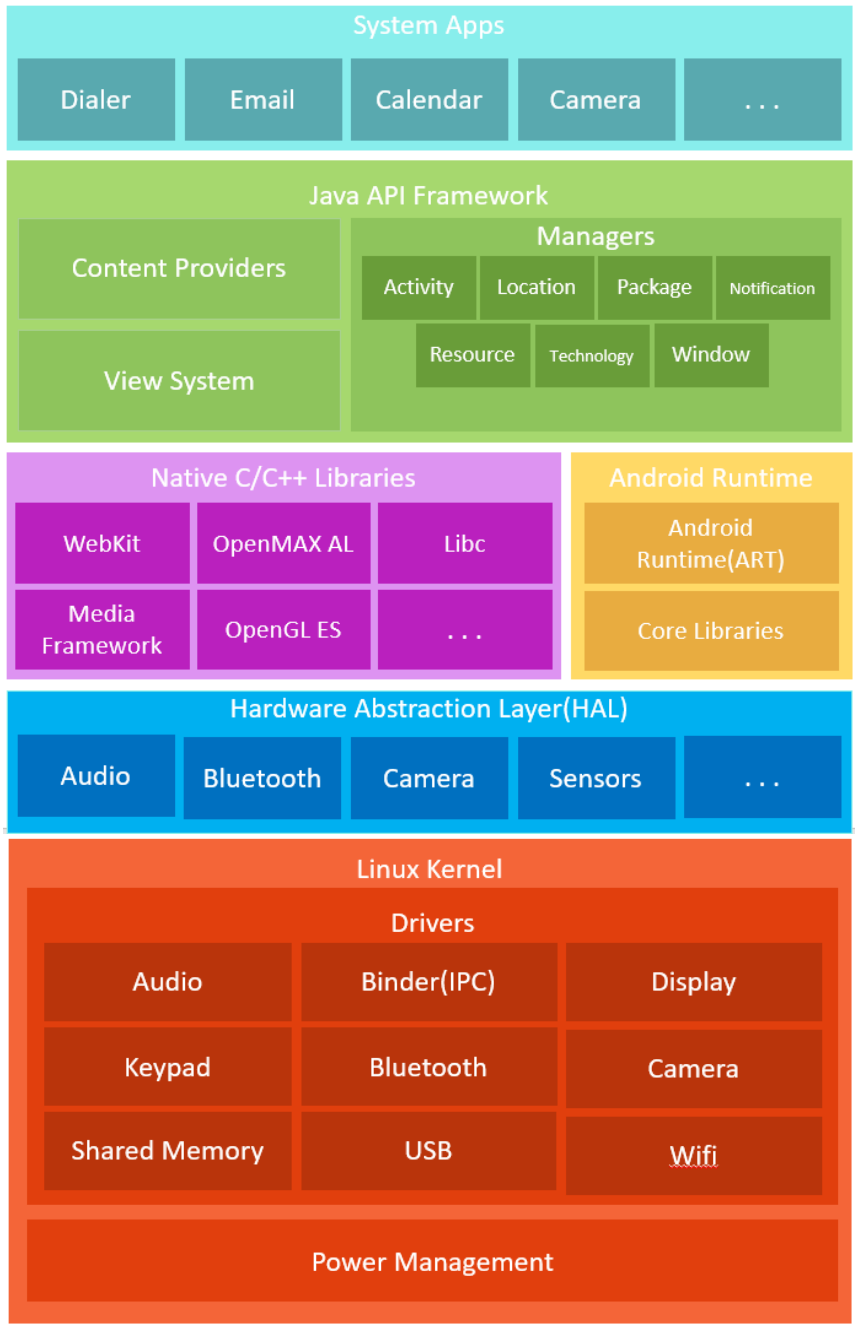


Fig. 4 Android platform main components.

The analysis of recent developments in machine learning-driven Android malware detection is covered in the parts that follow. Based on the methods used to extract features from software for teaching the machine learning algorithms, these approaches can be divided into three groups: static analysis-based, dynamic analysis-based, and hybrid

analysis-based methods.

## 2.2 INTRODUCTION TO APPLICATION ANALYSIS TECHNIQUES

The Android operating systems provide a variety of security features. These safeguards make an effort to restrict such dangerous activity in the system. The Android permission control system is one of the key security measures offered by Android platforms. In this system, the user must explicitly grant each application's request for each permission. A few permissions include having access to contacts, files, and other sensitive data stored on the device, such as email passwords and security questions. With such a system, both the user and the developer must exercise greater caution when granting such permissions.

Android platform is vulnerable to malicious applications. Various studies have been carried out to develop the method to detect such malwares in android platforms. Based on the features that can be extracted from the applications and then performing the analysis, these analysis techniques are mainly categorized in two different types, namely static analysis and dynamic analysis.

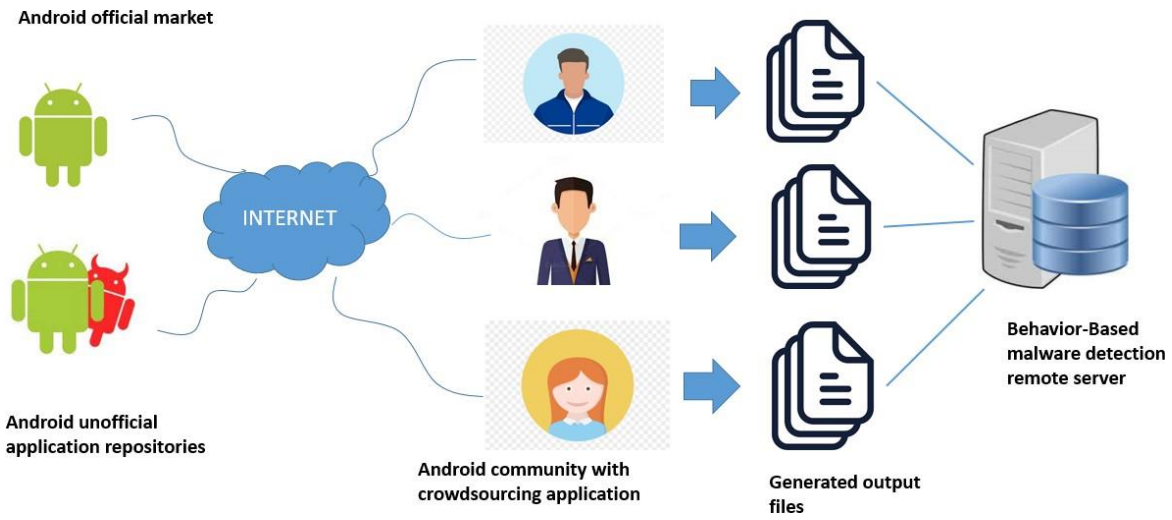


Fig.5 Behavior-Based Malware Detection Framework

Above information clearly concludes that there is an urgent need of an effective android malware detection method to prevent the android users from malicious attacks. In this report, various methodologies and implementations have been discussed and analyzed in the literature review. To detect the malwares various features, have to be extracted from the android applications. These features are basically categorized by analyzing them using any of the three different analysis techniques named:

- Static
- Dynamic
- Hybrid

The features are separated from the applications by static means in static analysis, for example without running or executing the application. Although static analysis if quite effective in detecting the existing malicious code but it may be challenging when malware uses self- decrypting code or naive code. In dynamic analysis features are extracted from the applications by executing it in run time environment. It is more capable of detecting the harmful applications with obfuscated codes and encrypted codes. Dynamic analysis on extracting the features in run time such as getting the network information, system calls, IP addresses etc. Static analysis fails when the codes are downloaded dynamically during execution. Hybrid analysis uses a combination of both the static and dynamic analysis techniques.

## 2.3 STATIC ANALYSIS BASED MALWARE DETECTION

A novel malware detection framework created by Peiravian and Zhu [15] uses permissions and Application Programming Interface (API) calls as features. They used the app Apktool to reverse engineer a specific APK and extract its AndroidManifest.xml file and class files in order to gather this information. A binary vector representing the permissions was created by extracting them from the AndroidManifest.xml file. The appropriate entry in the vector was set to 1 if the any permission was requested in the AndroidManifest.xml file; otherwise, it was put to 0. Similar to how API calls were extracted from the class files, each application was given a binary vector representation and

a class label indicating whether it was benign or malware.

To train their framework utilizing these attributes, the researchers used the Support Vector Machine (SVM), Decision Tree (DT), and Bagging methods. They used 1250 benign samples and 610 malware samples in their dataset. Three different feature combinations were used in the studies. The Bagging algorithm produced the greatest results when just employing permissions, with an AUC (specified in Chapter 3.4) of 0.956. SVM had the best performance when using simply API calls, with an AUC of 0.957. Finally, the best method attained an AUC of 0.991 when combining permissions and API calls.

An Android application that is installed on cellphones was created by D. Arp, M. Spreitzenbarth, M. Hübner, and others [16] as a lightweight malware detection technique. The malware detection procedure takes place immediately on the device, necessitating effective static feature extraction for machine learning algorithm training. In order to accomplish this, the researchers took features from two distinct sources: the AndroidManifest.xml file using AAPT and the Dalvik bytecode using a specially created disassembler to cut down on extraction time. Eight sets of features were the result of this.

Four feature sets were extracted from the AndroidManifest.xml file, including requested hardware components (e.g., GPS, camera access), requested permissions, application components (activities, services, content providers, and broadcast receivers), and filtered intents. The remaining four feature sets were extracted from the disassembled Dalvik bytecode, namely restricted API calls, used permissions, suspicious API calls, and network addresses.

The Android permission system restricts a group of sensitive API calls known as restricted API calls. Because illegal use without the appropriate permission could signify privilege escalation attacks, these API methods are useful as features. In order to determine which permissions are both requested and used, the used permissions set is produced by comparing the restricted API calls with the requested permissions. The suspicious API call set contains API calls that allow access to private information or resources, which could potentially result in harmful action. The network address set also records the network addresses that the application uses.

These feature sets are combined into a single, binary-dimensional vector with about 545,000 features, which is then used to represent the data. If a feature is present in the application, a value of 1 is assigned to each dimension in the vector; otherwise, a value of 0 is assigned. Every sample in the dataset, which consists of 131,611 samples from different Android application stores, including the Android Malware Genome Project, is subjected to this method.

The explanation of the detection findings that the malware detection tool Drebin generates is one of its supplementary features [16]. Drebin displays a screen with the top features indexed by their weights together with a detection score showing the confidence in the classification when an application is scanned. Along with descriptions of their functionality, these top features highlight the elements that have the greatest influence on determining whether an application is malicious or benign.

The explanation of the detection findings that the malware detection tool Drebin generates is one of its supplementary features. Drebin displays a screen with the top features indexed by their weights together with a detection score showing the confidence in the classification when an application is scanned. Along with descriptions of their functionality, these top features highlight the elements that have the greatest influence on determining whether an application is malicious or benign.

Using a dataset of 500 malicious applications and 500 benign applications, these attributes are used to train various machine learning algorithms. The training procedure makes use of cross-validation 10 times. Linear Discriminant (LD), Cubic SVM, Weighted K-NN, Complex Tree (DT), Linear SVM, and Course K-NN are among the methods that were examined in the study. The algorithm with the highest accuracy, 91.7 percent, was Cubic SVM. Additionally, the study found that using only intent filters as features produced subpar classification results, but combining permissions and intent filters produced the better results.

In a study on Android malware detection, C. Zhao, W. Zheng, L. Gong, and others

used a subset of API calls as features [18]. A subset of API calls was produced by using a regular expression pattern to retrieve the methods from each application's class.dex file using Androguard. Each API request received a sensitivity score that indicated its relationship to its recurrence in harmful programs. Tests were carried out, resulting in the top 20 most sensitive API calls being chosen in order to identify the ideal amount of API calls to be used as features.

Decision Tree (DT) and k-Nearest Neighbors (K-NN) classifiers were used for the ensemble model. A variety of performance indicators were assessed by the researchers, including accuracy, false positive rate (FPR), and true positive rate (TPR). They found that employing an ensemble model enhanced accuracy and FPR, resulting in an accuracy of over 90% on average. Regarding accuracy and TPR, the K-NN and DT classifier combination produced the best outcomes. The study also investigated the effects of various weights given to the classifiers in the ensemble. It was determined that the ensemble attained the maximum accuracy when giving K-NN and DT weights of 0.4 and 0.6, respectively, yielding an accuracy of about 90%.

## 2.4 DYNAMIC ANALYSIS-BASED MALWARE DETECTION

In contrast to static analysis, dynamic analysis involves running an application in a sandbox environment to watch how it behaves. Real-time monitoring and analysis of the operations of the program are made possible by this method. However, compared to static analysis, dynamic analysis approaches typically take longer.

System calls with zero variance are removed from the feature set to improve it, leaving a final feature vector with 43 attributes (excluding the class label). These features are used to train DT, RF, GBT, K-NN, SVM, ANN, and DL, among other machine learning algorithms. Before retraining and testing the algorithms, three feature weighting techniques—Information Gain (IG), Chi-square statistic, and correlation—are used to improve algorithm performance.

The frequency of system calls is used as a characteristic in Singh and Hofmann's

[19] suggested malware detection method. The Monkey tool is used to run each application in an emulator during the initial stage. Clicks, touches, gestures, and system-level events are all generated at random by Monkey. A feature vector with 337 items is produced as a result of monitoring 337 different Linux system calls during execution. The count of a particular system call's runtime invocations is represented by each element.

System calls with zero variance are removed from the feature set to improve it, leaving a final feature vector with 43 attributes (excluding the class label). These features are used to train DT, RF, GBT, K-NN, SVM, ANN, and DL, among other machine learning algorithms. Before retraining and testing the algorithms, three feature weighting techniques—Information Gain (IG), Chi-square statistic, and correlation—are used to improve algorithm performance.

In their study [20] on malware identification, Bhatia and Kaushal used the frequency of runtime system call invocations as characteristics. 50 harmful and 50 benign samples made up their collection. The Monkey tool was used to run each application in the dataset for a full minute inside of an Android Virtual Machine (VM). 500 gestures were produced throughout this run, each with a 500 millisecond delay. To extract the frequency of the called system calls during the execution time, the Linux command strace was simultaneously run.

A matrix was created from the retrieved data, with each row representing the frequency of system calls for a single application and each column representing the frequency of a specific system call across all apps. The classification job was chosen for the J48 and RF algorithms.

A malware detection method was created by Afonso, de Amorim, Grégio, Junquera, and de Geus [21] that makes use of the frequency of API and system calls made at runtime. They used the APIMonitor tool, which was run for five minutes on an emulator using MonkeyRunner, to extract the API calls. Additionally, they added additional monitoring of API requests relating to network access, process execution, string and file manipulation,

and information reading to the file used by APIMonitor to collect API calls. During this time, system calls were extracted using the Linux program trace.

The gathered data was combined into a feature vector with 74 API calls and 90 system calls, yielding 164 features in total. Each characteristic indicated how frequently a particular API or system call was used. A dataset with 2,295 harmful samples and 1,485 benign samples was used for training and evaluation. To find the best algorithm for the suggested method, a number of machines learning algorithms, including RF, J48, LR, NB, BN, SMO, and IBk, were trained on this dataset. Using the aforementioned dataset, RF had the greatest performance with an F1-score of 0.96. Then, RF was tested on a different dataset made up of 1,483 benign samples and 2,257 malware samples.

## 2.5 HYBRID ANALYSIS BASED MALWARE DETECTION

In their method to categorize apps, Zhao, Xu, and Zhang [22] combined static and dynamic analysis techniques to extract permissions, API calls, and runtime behaviors as static features and runtime behaviors as dynamic features. They used the Androguard program to retrieve permissions from the AndroidManifest.xml file during the static analysis phase. A binary permission feature vector with 45 dimensions representing the presence of each permission in an application was created by optimizing the permission feature set by deleting features that were infrequently used.

They examined the classes.dex files using Androguard and the reverse-engineering program baksmali to extract API calls. Using the Relief filter feature selection technique, the acquired API feature vector was further refined to produce a final API call feature set with 22 dimensions, each of which represents an API call.

Each program was installed and run on an emulator during the dynamic analysis phase. The DroidBox tool monitored and picked up on harmful behaviors like automatic network connections, malicious SMS sending, and private information logging while the Monkey tool generated runtime behaviors. Each instance of each behavior was noted, then

the Relief algorithm was used to remove extraneous features. The approach produced a final feature vector with 20 dimensions that included information on battery life, user activity, network features, and other factors.

87 dimensions were added to the feature vector created from the retrieved static and dynamic features. Randomly chosen subsets of 150 harmful and 150 benign samples were utilized for training and testing on a dataset made up of 359 malware samples and 500 benign samples. The performance of many machines learning algorithms, including SVM, K-NN, NB, DT, and RF, was assessed by the authors. The best method when static analysis features were taken into account was RF, which had an accuracy of 92.07 percent. The maximum accuracy, however, was attained by RF when using both static and dynamic analytic characteristics, which resulted in a 94.89 percent accuracy.

Based on the results of the APK extraction procedure, Liu, Zhang, Li, and Chen [23] suggested a novel approach that combines static and dynamic analysis. Static analysis is performed on the application if Apktool can successfully decompile it. However, dynamic analysis is used if the decomplication process fails to produce pertinent data, indicating the employment of code obfuscation techniques. Each application's AndroidManifest.xml file is extracted during the static analysis stage, and the permissions are then extracted and mapped to a feature vector of 151 dimensions. Additionally, baksmali is used to extract API calls, which are then mapped to a feature vector with 3262 dimensions. The final feature vector, which has 3413 dimensions, is created by merging these two feature vectors.

A system call feature vector of 345 dimensions is constructed during the dynamic analysis phase, and each dimension corresponds to the frequency of system calls that were invoked. To extract these features, the ADB (Android Debug Bridge) program is used. The Monkey tool is then used to run the application, and the Linux command trace is used to track the called system calls.

The authors trained the K-NN, SVM, and NB algorithms using a dataset that included 500 malicious samples and 500 benign ones. When permissions were used as the feature

set, SVM had the highest accuracy (96.53%). SVM also performed best with an accuracy of 99.07 percent when using API calls as features. SVM beat other methods, obtaining an accuracy of 99.28% when permissions and API calls were combined. Finally, the best algorithm was NB, which had a 90% accuracy when system calls were utilized as features.

# CHAPTER 3

# PROPOSED METHODOLOGY

## 3.1 OVERVIEW

Despite the growing number of malware applications, there is still no effective and reliable approach for detecting them. We believe that the issue of identifying malware can be handled using Machine Learning approaches, given the rising application of Machine Learning in numerous fields.

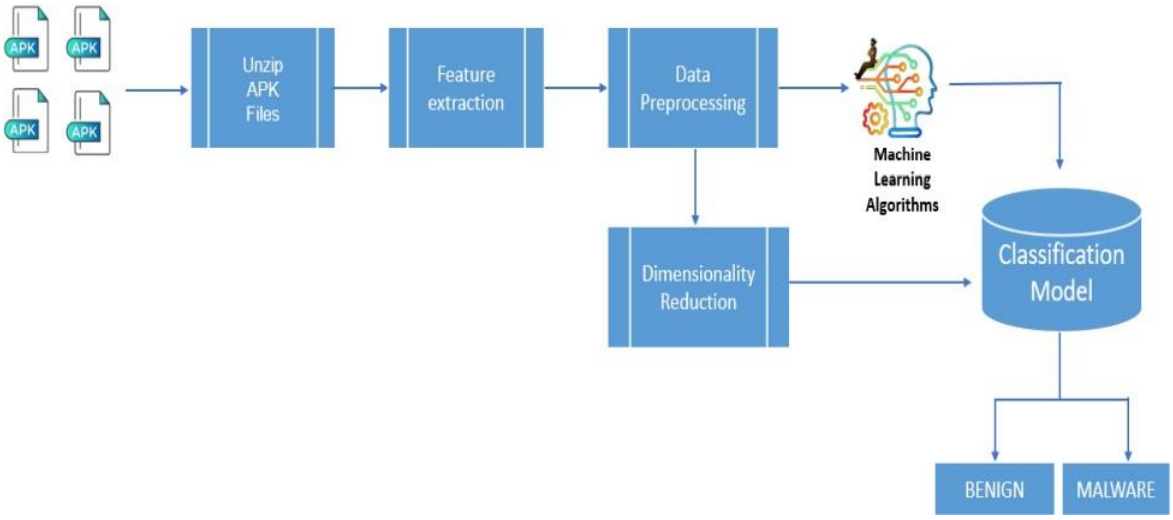


Fig.6 Proposed Methodology flow chart

Our project intends to conduct a thorough and systematic investigation into malware detection using machine learning techniques, as well as develop an effective ML model that can classify applications into benign (0) and malicious (1) categories depending on the requested app permissions. This research proposes:

- A technique which is available publicly and a metadata-based machine learning malware detection strategy.
- Because feature selection is crucial in malware detection systems for Android, this research includes permissions that are most distinctive. This phase has a great impact on the enhancement of the detection model.
- Based on the permissions, design a model that can properly anticipate the Malicious Application.

## 3.2 FEATURE EXTRACTION

The Android operating system employs the Android Package Kit (APK) file format to distribute and install apps. It comprises all the components necessary for an application to be installed correctly on the device. The Java programming language is used to create Android applications. The manifest file (AndroidManifest.xml) and classes.dex are included in the installation package, which is a compressed (ZIP) bundle of files. This file contains many stuffs such as application source codes, photos, API requests, and application permissions. Using the AAPT module in the first stage, the apk files in the malicious and benign applications are made accessible to the package content. In this project I am using permissions that are used by the applications as attributes. So rather than using all the permissions, the main aim is to create a feature vector for each application and store it in a csv file. At last, while storing it in the csv file, the applications are labeled as benign and malware. Following that, preprocessing methods were used to produce the best categorization results.

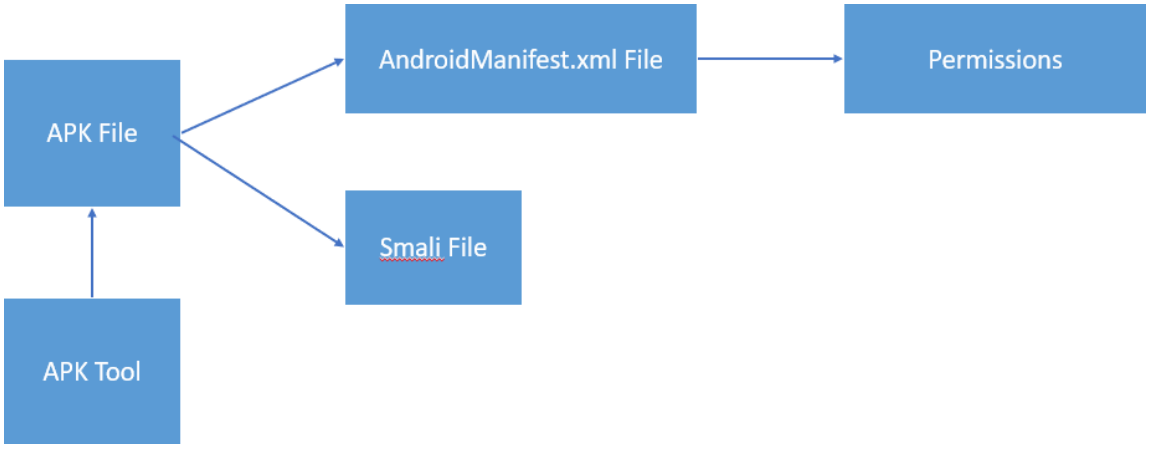


Fig.7 Feature Extraction.

## 3.3 DATASET

Dataset has 30,000 applications. It contains permissions of of all the applications in the form of vectors (explained in feature extraction). 183 features have been extracted from the feature extraction phase. It contains all permissions which includes dangerous permissions, default permissions, administrative permissions etc. The last column is labeled as class, where the benign applications are labeled ‘0’ and malicious applications are labeled as ‘1’. Out of 30,000, there are 20,000 malwares and 9999 benign applications.

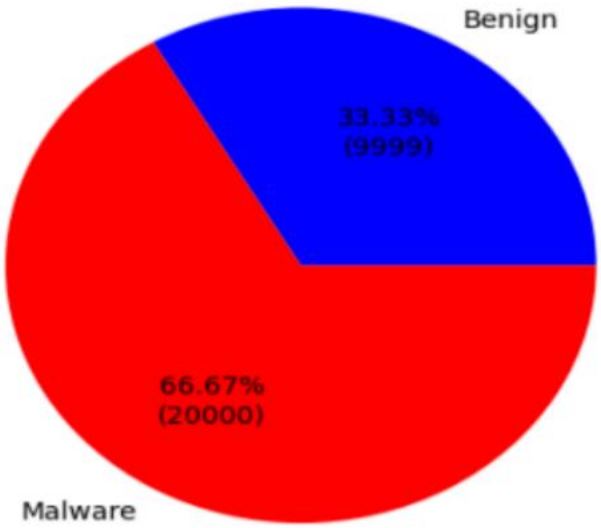


Fig.8 Dataset Distribution

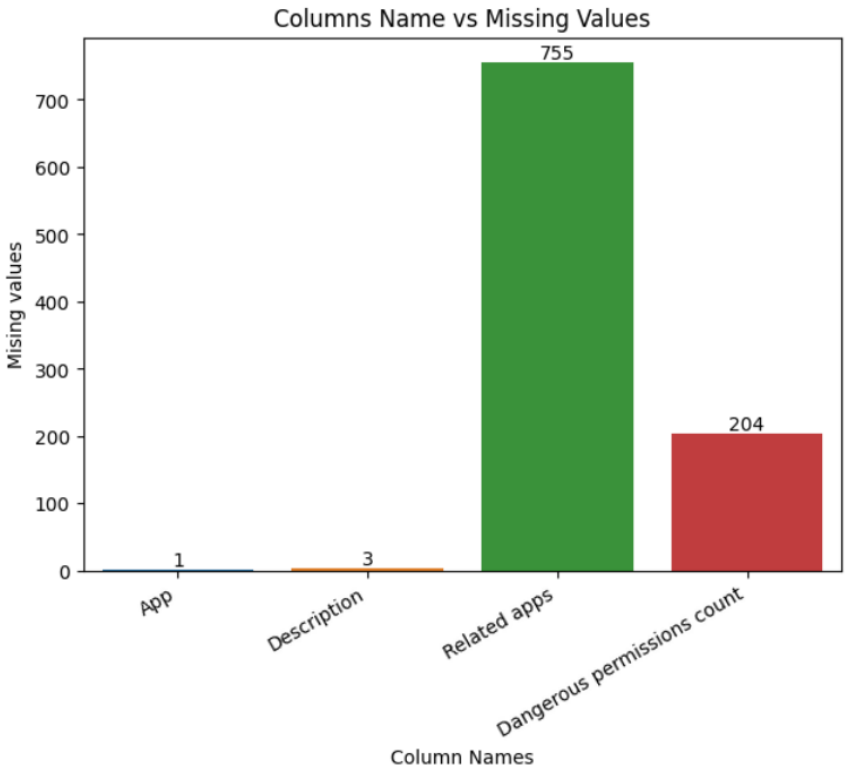


Fig.9 Dataset: Missing Values vs Column name

Missing values in dataset is taken care by filling them with the mean of the remaining values and then ignoring the NA values in the Application column.

| App | Default : Access DRM content. (S) | Default : Access Email provider data (S) | Default : Access all system downloads (S) | Default : Access download manager. (S) | Default : Advanced download manager functions. (S) | Default : Audio File Access (S) | Default : Install DRM content. (S) | Default : Modify Google service configuration (S) | Default : Modify Google settings (S) | Default : Move application resources (S) | ... | Your personal information : read calendar events (D) | Your personal information : write to user defined dictionary (S) | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Canada Post Corporation | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| Word Farm | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| Fortunes of War FREE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| Better Keyboard: Avatar Purple | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| Boxing Day | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| Beechwood Hotel | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 |
| ZDefense: Tower Defense | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 |
| Super Task Killer FREE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 |
| Ambassador Hotel Taipei | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 |
| Brain Ball Game | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 |

29998 rows × 174 columns

Fig.10 Dataset Description after preprocessing

## 3.4 DIMENTIONALITY REDUCTION

### 3.4.1 FEATURE SELECTION

Feature selection plays the most crucial role in machine learning methods. There are various elements inside an application file(apk). It includes permissions, Java code, certification, behavior of the application on the device and its behavior on the network. The outcome of the experiment gets enhanced when the most helpful subset of features is chosen from many options. There are various benefits of deploying feature selection phase:

- The dimensions of the dataset can be reduced by using feature selection methods since you can quickly visualize the trend in data with fewer data.
- While analyzing the datasets it involves a large amount of data to be processed, so by applying feature selection the dimension of the dataset reduces which in turn reduces the time required for actual implementation. The training time of machine learning algorithms is significantly reduced when a meaningful subset of features

is selected.

- Feature selection helps machine learning algorithms provide more accurate results by removing noisy and irrelevant data from datasets.

### 3.4.2 Principal Component Analysis (PCA)

Columns in a dataset are converted into a new set of features called Principal Components using the Principal Components Analysis (PCA) technique. With this method, a significant amount of the data in the entire dataset is successfully condensed into fewer feature columns. This enables the visualization of any class or cluster separation as well as dimensionality reduction. The information contained in a column is the variance that is included in it. Using the fewest number of columns feasible to convey data is the fundamental goal of principal components analysis.

There are two main reasons for which PCA is used:

- Dimensionality Reduction: The data from a large number of columns is transformed into principal components (PC) so that the first few PCs can explain a significant portion of the overall data (variance). In Machine Learning models, these PCs can be utilized as explanatory variables.
- Visualize Classes: For data with more than three dimensions, visualizing the separation of classes (or clusters) is difficult (features). It's frequently feasible to notice a definite separation between the first two PCs.

PCA has many uses, including data compression, feature selection, and data visualization. In terms of data visualization, PCA makes it possible to portray high-dimensional data in two or three more manageable dimensions, improving interpretability. Another use of PCA is feature selection, which enables the discovery of important variables in a dataset. Last but not least, PCA helps with data compression by lowering the dimensionality of a dataset while maintaining important information.

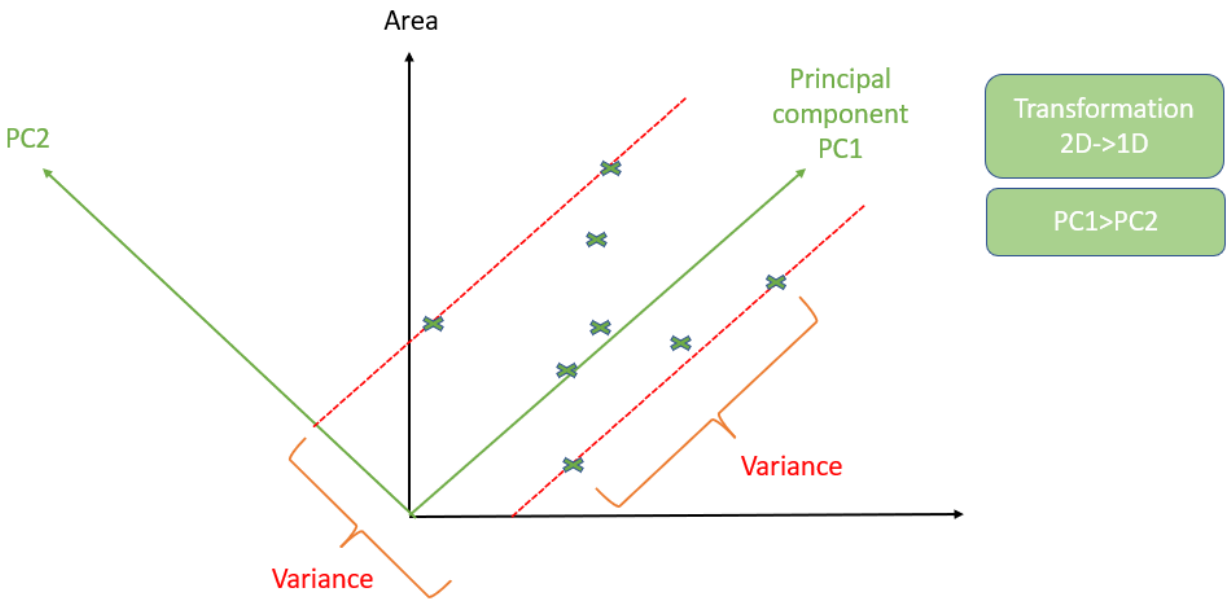


Fig.11 Principal Component Analysis

### 3.4.3 Linear Discriminant Analysis (LDA)

LDA is a further method for dimensionality reduction. LDA is typically used to categorize supervised issues. This approach projects the features of a higher dimensional space into a lower dimensional space. It is used to model distinctions between groups when dividing two or more classes.

Let's look at an example where two classes are shown on a 2-D plane with an x-axis and a y-axis. LDA develops a new axis or straight line that successfully divides the data points of the classes on a plane in order to efficiently categorizes between the classes. LDA lowers a two-dimensional plane to a one-dimensional plane while maximizing class separation.

### 3.4.4 t-SNE (t-distributed Stochastic Neighbor Embedding)

T-distributed stochastic neighbor embedding (t-SNE) is a statistical technique for displaying high-dimensional data by allocating a location on a two- or three-dimensional map to each datapoint. The Stochastic Neighbor Embedding algorithm was developed by Sam Roweis and Geoffrey Hinton in the beginning, and Laurens van der Maten gave the t-distributed variant for it. This nonlinear dimensionality reduction method visualizes high-dimensional data in a low-dimensional space of two or three dimensions. The unsupervised non-linear dimensionality reduction method t-SNE (t-distributed Stochastic Neighbor Embedding) is useful for data exploration and high-dimensional data visualization. It is possible to use the approach to separate data that cannot be separated linearly because it separates data in a non-linear manner.

A good tool for understanding how data is organized and structured in higher dimensions is t-SNE. Its main use is to convert complicated information into two- or three-dimensional representations, which makes it easier to understand the data's underlying relationships and patterns. The similarity measure between pairs of instances in both higher and lower dimensions spaces is determined by the t-SNE technique. Then, two similarity measures are optimized. Three distinct steps are taken to complete these operations.

- Points can have neighbors in both higher and lower dimensions according to the t-SNE algorithm. A Gaussian kernel is first used to assess the pairwise similarity between each data point in the high-dimensional space. The chance of choosing points as neighbors is affected by their proximity, favoring closer points over those that are farther apart.

- The suggested approach seeks to decrease the data's dimensionality while preserving the relative similarity of the data points.

- The algorithm's goal is to reduce the difference in probability distributions between the original high-dimensional data and its corresponding lower-dimensional embedding. The lower-dimensional embedding is gradually changed until it converges to a stable state in an iterative process employing gradient descent.

The development of clusters and sub-clusters of related data points in the lower-dimensional space is made possible by the optimization process. This enables the visualization of the higher-dimensional data's structure and linkages, giving insights into its underlying patterns.

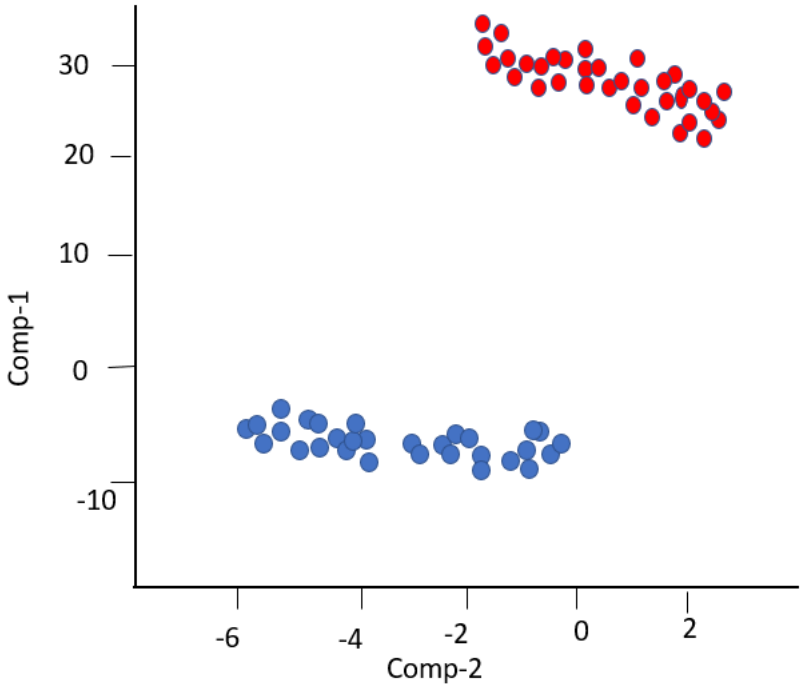


Fig.12 t-SNE Data Visualization

## 3.5 CLASSIFICATION ALGORITHMS

### 3.5.1 XG BOOST

Machine learning models are frequently trained using the highly optimized and scalable gradient boosting library known as XGBoost. It uses an ensemble learning

strategy, integrating the results of various weak models to produce a single, stronger prediction. Extreme Gradient Boosting (XGBoost), sometimes known as "Extreme Gradient Boosting," has become extremely popular because of its prowess in handling massive datasets and delivering cutting-edge results in a variety of machine learning applications, including regression and classification. Researchers and practitioners in the field like it because of its effective use and excellent results.

To handle enormous datasets and real-world data effectively, XGBoost has some noteworthy features. Its capability to handle missing values well, which prevents the need for labor-intensive data preprocessing, is one of its main advantages. With the aid of this capability, XGBoost is able to handle missing value datasets from the real world with ease. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on huge datasets quickly. Because of its enhanced scalability and efficiency due to parallelization, XGBoost is a strong option for machine learning applications.

Due to its adaptability, it may be used for many different jobs. Additionally, XGBoost provides significant levels of customizability, enabling users to adjust model parameters in accordance with their own needs. Users can improve performance and provide better results in their apps because to this flexibility. Extreme Gradient Boosting, or XGBoost, is a machine learning package that has been enhanced and was developed by academics at the University of Washington. It was created in C++ and was created primarily to improve Gradient Boosting training, leading to better performance and efficiency.

The XGBoost algorithm constructs decision trees in a sequential manner. Each independent variable is given a weight, and the decision tree uses these weighted variables as input to produce predictions. If the tree incorrectly predicts a variable, its weight is increased, and the new variables are used in the subsequent tree. Each new tree in this iterative process gets better than the one before it. These distinct predictors are used to make the final forecast, creating a strong and reliable model. Regression, classification, ranking, and custom prediction issues can all be solved using XGBoost.

### 3.5.2 SVM (Support Vector Machine)

The Support Vector Machine (SVM) is a supervised learning technique used in machine learning for both classification and regression tasks. SVM is mostly renowned for its success in resolving classification issues, even though it may also be used to solve regression problems.

The SVM algorithm's fundamental goal is to locate the ideal decision boundary, or hyperplane, that can successfully divide data points in an n-dimensional space into different classes. Future data points can be accurately categorized using this hyperplane based on how they are situated in relation to the decision boundary.

The Support Vector Machine (SVM) technique creates a hyperplane that distinguishes between various categories by using support vectors, which are extreme points or vectors. These key cases are chosen by the SVM approach in order to precisely define the decision boundary. SVM successfully divides data into distinct groups by concentrating on the support vectors, which is how it derives its name. In an n-dimensional space, a hyperplane represents the ideal decision border that can distinguish between classes. The objective is to determine the border that will classify data points most accurately, although there may be several lines or decision boundaries to divide the classes. In SVM, the hyperplane is the name of this ideal decision boundary.

The features included in the dataset determine the hyperplane's dimensionality. For instance, the hyperplane will be a straight line in the case of two features (as seen in the image). On the other hand, the hyperplane will be a two-dimensional plane if the dataset comprises three features. The maximizing of margin is a guide while creating the SVM hyperplane. One must select a hyperplane that maximizes the distance between the data points in order to achieve the greatest separation between the classes. Support vectors are the nearest data points or vectors that significantly affect the position of the hyperplane. Because they support the hyperplane, they are referred to as support vectors.

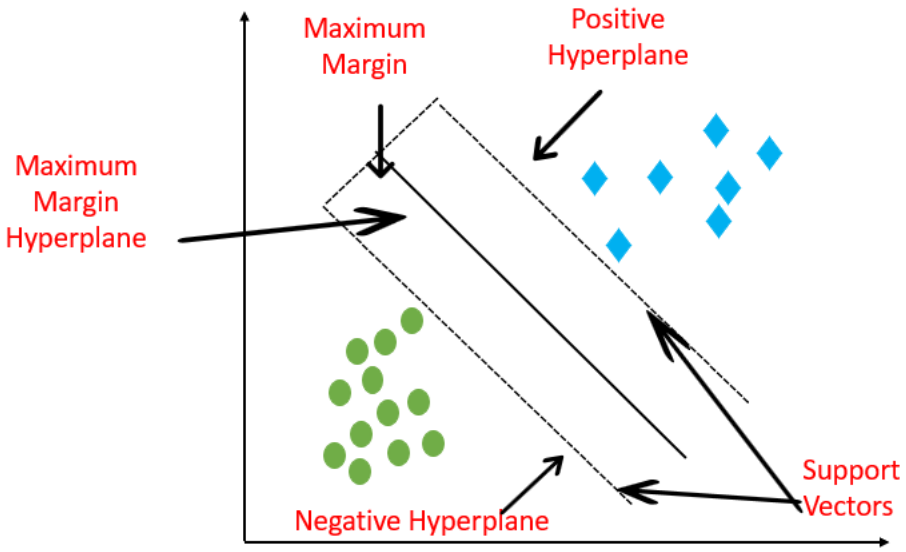


Fig.13 SVM classification using boundary decision or hyperplane

### 3.5.3 DECISION TREE

Although it is most frequently used for classification tasks, Decision Tree is a supervised learning technique that may be used to solve both regression and classification issues. It functions as a tree-like structure where the internal nodes stand in for a dataset's features, the branches for the decision-making processes, and each leaf node for the associated result.

Decision Nodes and Leaf Nodes are the two different sorts of nodes that make up a decision tree. While Leaf Nodes reflect the outcomes and have no further branches, Decision Nodes are in charge of making decisions and have several branches. Based on the features included in the submitted dataset, decisions or tests are carried out.

By taking into account multiple factors and their accompanying outcomes, a decision tree serves as a visual representation that presents viable solutions to a problem or choice. The phrase "decision tree" refers to a process that follows a structure resembling a tree, starting at the root node and branching out to form a tree-like arrangement. The CART

(Classification and Regression Tree) technique is used to create decision trees. A decision tree basically asks a question, and depending on the response (Yes/No), divides the tree into subtrees. Up until the leaf nodes, which stand in for the final results, this iterative process continues.

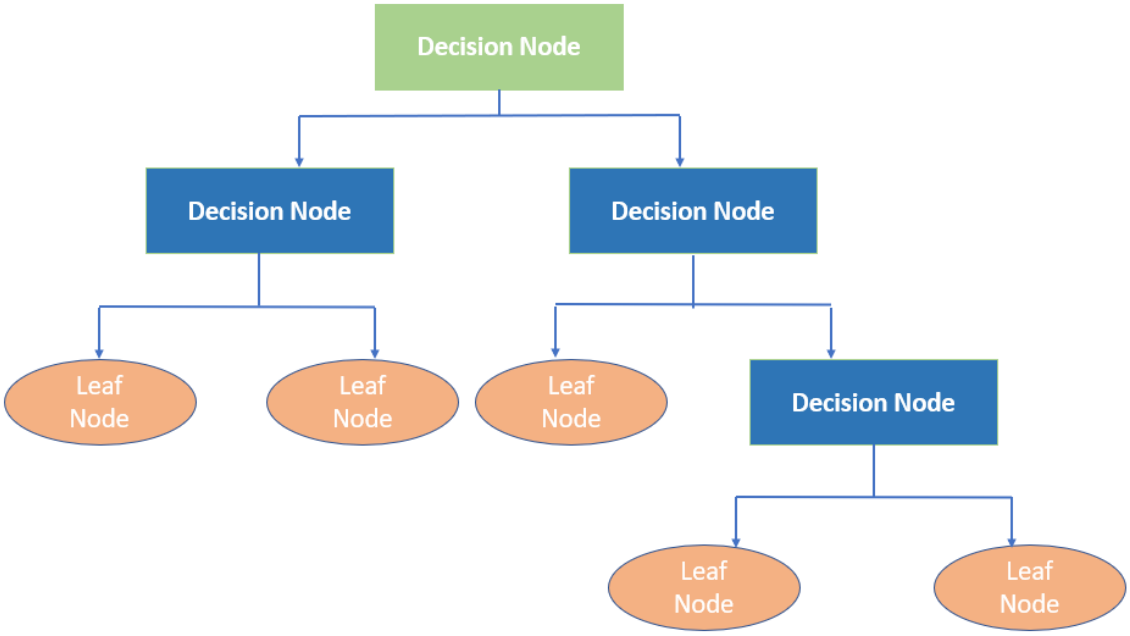


Fig.14 Decision Tree

### 3.5.4 MLP (Multilayer Perceptron)

Through the use of machine learning, computers can learn from data without explicit programming. It has numerous uses in many different fields, such as predicting, speech recognition, and picture recognition. The MLP classifier, or multi-layer perceptron, is a widely used machine learning technique. An artificial neural network, or ANN, is a sort of computer system created to simulate how the human brain works. The MLP classifier is one such ANN.

A training dataset is necessary for the MLP classifier to learn, making it a supervised learning method. The input data (x) and associated output labels are included in this dataset (y). The MLP classifier gains knowledge about how to map inputs to output

labels by utilizing the input data. The MLP classifier may make predictions on new data once it has successfully mastered this mapping. The MLP classifier can forecast an output label for a new data point (x) when it is entered (y).

A flexible machine learning technique appropriate for both classification and regression task is the MLP classifier. It is extensively used in a variety of fields, including as voice and picture recognition. It has an input layer, a hidden layer, and an output layer and is a member of the neural network family. The data are received by the input layer, they are processed by the hidden layer, and the classification results are produced by the output layer. It is a type of neural network that is distinguished by having numerous layers of neurons or nodes. The first layer, also referred to as the input layer, is in charge of taking in data. Information is saved and processed on the next layer, which is referred to as the hidden layer. The third layer, which is the output layer, produces the categorization results last.

A set of weights that define a decision boundary between numerous classes must be learned in order for the MLP Classifier to work. After the classifier has been trained using a particular dataset that includes data points from each class, these weights are acquired. After being trained, the classifier can be used to predict the class of new data points.

A training dataset made up of data points corresponding to each class to be classified is necessary in order to train an MLP Classifier. Each data point needs to have class-specific labels and descriptive features or attributes. From the training data, the classifier derives a set of weights that it uses to determine the distinction between classes. By computing the distance between the new data point and the decision boundary established by the weights, the classifier may then predict the class of new data points.

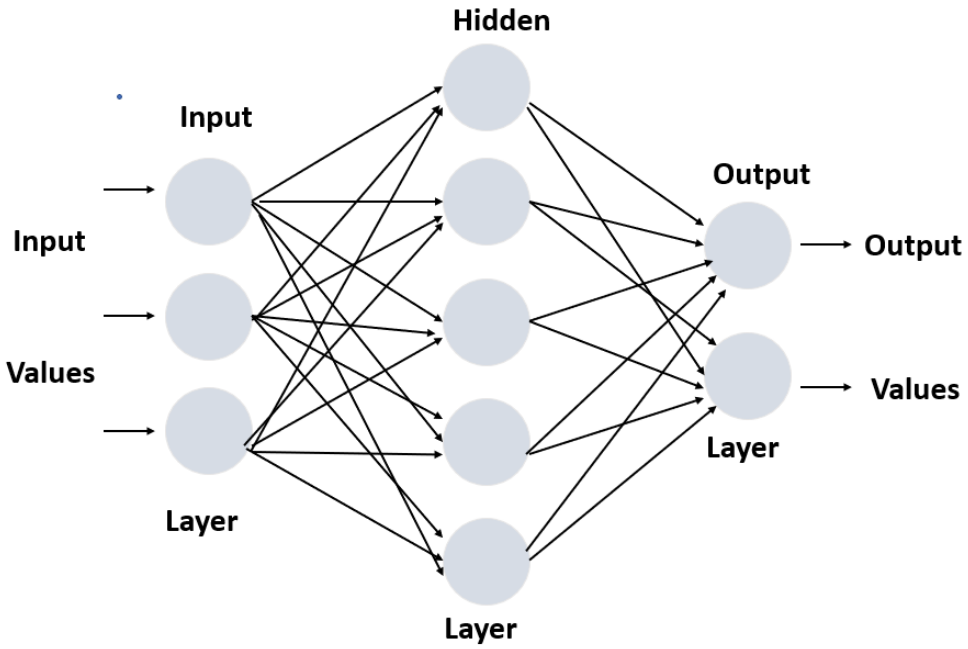


Fig.15 Multilayer Perceptron

# CHAPTER 4

# IMPLEMENTATION AND RESULTS

## 4.1 ALGORITHMS USED AND PERFORMANCE MEASURE

There are few classification algorithms applied in this study namely SVM, XGboost, MLP and decision tree. Support Vector Machine is a classification-based technology that uses a classifier algorithm. If the data size is small, it is quite handy. For huge datasets, this approach is ineffective. Decision tree is another technique for supervised learning. It can be implemented for both regression and classification problems. MLP stands for Multi-Layer Perceptron, is a neural network that consists of multiple layers for input and out and also there are some hidden layers between them. An application of gradient-boosting decision trees is XGboost. On large datasets, XGboost performs quickly, is simple to use, and is effective. Prior to the application of above algorithms, PCA and LDA is applied to reduce the dimensions of the dataset and time of processing.

This nonlinear dimensionality reduction method visualizes high-dimensional data in a low-dimensional space of two or three dimensions. The unsupervised non-linear dimensionality reduction method t-SNE (t-distributed Stochastic Neighbor Embedding) is useful for data exploration and high-dimensional data visualization. It is possible to use the approach to separate data that cannot be separated linearly because it separates data in a non-linear manner. It is a good tool for understanding how data is organized and structured in higher dimensions is t-SNE. Its main use is to convert complicated information into two- or three-dimensional representations, which makes it easier to understand the data's underlying relationships and patterns.

The following assessment measures were used to assess the classification algorithm implementations: overall accuracy, true positive rate, false positive rate, and precision. These four basic parameters are described below.

- True Positive (TP): This metric indicates the number of benign applications that were accurately identified.
- True Negative: A true negative is an outcome in which the model predicts the negative class correctly.
- False Positive (FP): This metric specifies the number of malware apps that have been wrongly recognized. To put it another way, a false positive classification misidentifies a malicious application as a benign application.
- False Negative (FN): This metric provides the number of benign applications that were wrongly recognized.

The generated evaluation metrics are shown below in terms of the core measures:

- Accuracy: The percentage of correct predictions made by our model is called accuracy.
- Precision: The proportion of true positive predictions among all cases categorized as positive is known as precision.
- Recall: The fraction of actual positive labels properly detected by the model is measured by recall.

## 4.2 RESULT EVALUATION AND CONCLUSION

A statistical method called PCA (Principal Component Analysis) is used for multivariate analysis with the aim of lowering the dimensionality of a dataset. This is done by creating new, uncorrelated variables by computing linear combinations of the original variables. With the first main component having the highest variance, the second holding the second-highest variance, and so on, the objective is to keep as much variance as possible. The dataset can be reduced to any number of dimensions the user chooses.

To effectively describe high-dimensional datasets in 2D and 3D space, the t-SNE (t-distributed Stochastic Neighbor Embedding) dimensionality reduction approach was developed [24]. It is noteworthy that the t-perplexity SNE's adjustable parameter roughly estimates the number of close neighbors that each data point has. Additionally, it has a parameter for learning rate that may be altered to accelerate the process. When utilizing t-SNE, it is customary to select a perplexity number between 5 and 50. The graphical representation was obtained by combining many t-SNE projections with different learning rate and perplexity settings. The results in the representation, is graphically shown below.

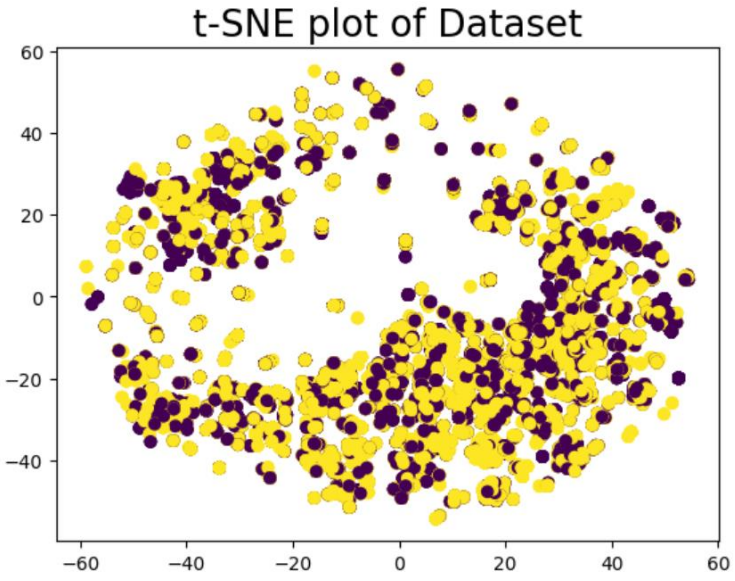


Fig.16: 2- dimensional t-SNE projection of dataset (benign and malware)

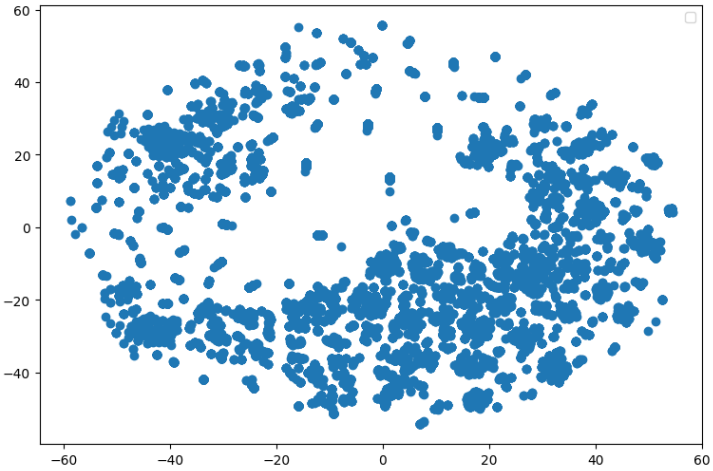


Fig.17: 2-dimentional t-SNE projection of benign applications from dataset

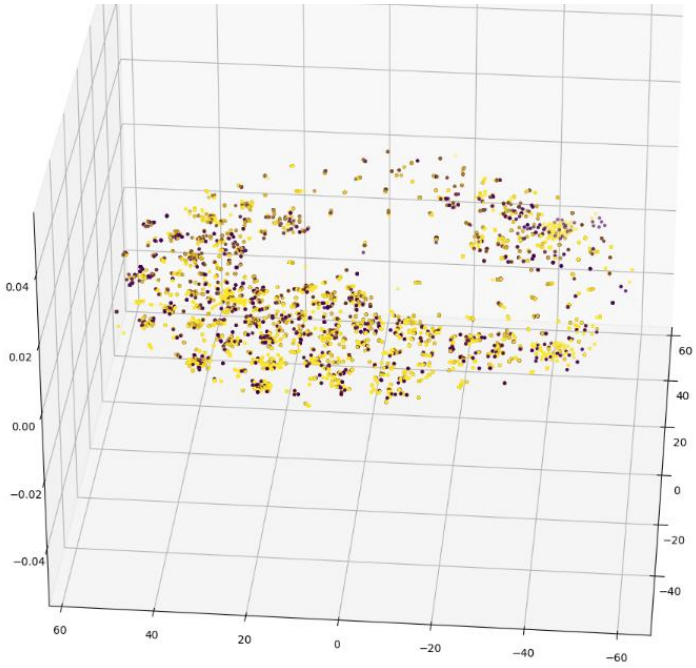


Fig. 18: 3-dimensional t-SNE projection of dataset (benign and malware)

The dataset used in this study was subjected to dimensionality reduction using the PCA and LDA methods. The accuracy of the suggested model is deteriorating as the dataset's number of columns increases. Now, PCA is applied to further decrease the dimensionality of the dataset and to improve accuracy (Principal component Analysis). After PCA is applied to the supplied dataset, the dimension is reduced from 173 to 10.

The same dataset is also subjected to LDA in a similar manner. By using such dimensionality reduction techniques, the processing time of the model is reduced, and its accuracy has increased noticeably. Following this, a few machine learning methods have been employed to categorize apps into malicious and benign ones.

| Dimensionality Reduction | Algorithm | Accuracy obtained |
|---|---|---|
| **PCA** | Xgboost | 76% |
| | SVM | 85% |
| | Decision Tree | 79% |
| | MLP | 85% |
| **LDA** | Xgboost | 76% |
| | SVM | 69% |
| | Decision Tree | 69% |
| | MLP | 69% |

Table 2: Result Comparison Table

Above are the results obtained by using LDA and PCA algorithms and then examining the accuracy by applying classification algorithms.

By observing the above table, it can be concluded that algorithms like SVM and MLP obtain better accuracy as compared to other algorithms like xgboost and Decision tree when PCA is used in dimensionality reduction phase. Similarly, xgboost yields better results when applied with LDA.

## 4.2 FUTURE WORK

In this study, malware detection techniques and its proposed methodology has been discussed. An additional phase named dimensionality reduction has been introduced and the results obtained with and without this phase is observed and compared. This phase has a great impact in the performance and accuracy of the model. The future work will be focused in further more improving the accuracy of the model and also applying new techniques of dimensionality reduction. Comparative study of various algorithms would be done to ensure that the better results.

# REFERENCES


[1] Şahin, Durmuş Özkan, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. "A novel permission-based Android malware detection system using feature selection based on linear regression." *Neural Computing and Applications* (2021): 1-16.

[2] ARSLAN, Recep Sinan. "FG-Droid: Grouping Based Feature Size Reduction for Android Malware Detection through Machine Learning." (2021).

[3] Feizollah, Ali, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. "A review on feature selection in mobile malware detection." *Digital investigation* 13 (2015): 22-37.

[4 Li, Jin, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. "Significant permission identification for machine-learning-based android malware detection." *IEEE Transactions on Industrial Informatics* 14, no. 7 (2018): 3216-3225.

[5] Pan, Ya, Xiuting Ge, Chunrong Fang, and Yong Fan. "A systematic literature review of android malware detection using static analysis." *IEEE Access* 8 (2020): 116363-116379.

[6] Pehlivan, Uğur, Nuray Baltaci, Cengiz Acartürk, and Nazife Baykal. "The analysis of feature selection methods and classification algorithms in permission based Android malware detection." In *2014 IEEE symposium on computational intelligence in cyber security (CICS)*, pp. 1-8. IEEE, 2014.

[7] Kouliaridis, Vasileios, Nektaria Potha, and Georgios Kambourakis. "Improving android malware detection through dimensionality reduction techniques." In *International Conference on Machine Learning for Networking*, pp. 57-72. Springer, Cham, 2020.

[8] Şahin, Durmuş Özkan, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. "Permission-based Android malware analysis by using dimension reduction with PCA and LDA." *Journal of Information Security and Applications* 63 (2021): 102995.

[9] Vega Vega, Rafael, Héctor Quintián, José Luís Calvo-Rolle, Álvaro Herrero, and Emilio Corchado. "Gaining deep knowledge of Android malware families through dimensionality reduction techniques." *Logic Journal of the IGPL* 27, no. 2 (2019): 160-176.

[10] Sangal, Aviral, and Harsh Kumar Verma. "A static feature selection-based android malware detection using machine learning techniques." In *2020 International conference on smart electronics and communication (ICOSEC)*, pp. 48-51. IEEE, 2020.

[11] Fatima, Anam, Ritesh Maurya, Malay Kishore Dutta, Radim Burget, and Jan Masek. "Android malware detection using genetic algorithm based optimized feature selection and machine learning." In *2019 42nd International conference on telecommunications and signal processing (TSP)*, pp. 220-223. IEEE, 2019.

[12] Chakravarty, Sujata. "Feature selection and evaluation of permission-based Android malware detection." In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pp. 795-799. IEEE, 2020.

[13] Zarni Aung, Win Zaw. "Permission-based android malware detection." *International Journal of Scientific & Technology Research* 2, no. 3 (2013): 228-234.

[14] Talha, Kabakus Abdullah, Dogru Ibrahim Alper, and Cetin Aydin. "APK Auditor: Permission-based Android malware detection system." *Digital Investigation* 13 (2015): 1-14.

[15] Peiravian, Naser, and Xingquan 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. IEEE, 2013.

[16] Arp, Daniel, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and C. E. R. T. Siemens. "Drebin: Effective and explainable detection of android malware in your pocket." In Ndss, vol. 14, pp. 23-26. 2014.

[17] Kumaran, Monica, and Wenjia Li. "Lightweight malware detection based on machine learning algorithms and the android manifest file." In 2016 IEEE MIT Undergraduate Research Technology Conference (URTC), pp. 1-3. IEEE, 2016.

[18] Zhao, Chunlei, Wenbai Zheng, Liangyi Gong, Mengzhe Zhang, and Chundong Wang. "Quick and accurate android malware detection based on sensitive APIs." In 2018 IEEE international conference on smart internet of things (SmartIoT), pp. 143-148. IEEE, 2018.

[19] Singh, Latika, and Markus Hofmann. "Dynamic behavior analysis of android applications for malware detection." In 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT), pp. 1-7. IEEE, 2017.

[20] Bhatia, Taniya, and Rishabh Kaushal. "Malware detection in android based on dynamic analysis." In 2017 International conference on cyber security and protection of digital services (Cyber security), pp. 1-6. IEEE, 2017.

[21] Afonso, Vitor Monte, Matheus Favero de Amorim, André Ricardo Abed Grégio, Glauco Barroso Junquera, and Paulo Lício de Geus. "Identifying Android malware using dynamically obtained features." Journal of Computer Virology and Hacking Techniques 11 (2015): 9-17.

[22] Zhao, Yang, Guangquan Xu, and Yao Zhang. "HFA-MD: An efficient hybrid features analysis based Android Malware Detection Method." In Quality, Reliability, Security and Robustness in Heterogeneous Systems: 13th International Conference, QShine 2017, Dalian, China, December 16-17, 2017, Proceedings 13, pp. 248-257. Springer International Publishing, 2018.

[23] Liu, Yu, Yichi Zhang, Haibin Li, and Xu Chen. "A hybrid malware detecting scheme for mobile Android applications." In 2016 IEEE International Conference on Consumer Electronics (ICCE), pp. 155-156. IEEE, 2016.

[24] Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9, no. 11 (2008).