CAPITAL UNIVERSITY OF SCIENCE AND
TECHNOLOGY, ISLAMABAD

Feature Selection using Genetic Algorithm for Android Malware Classification

by

Amir Jibran

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2022

My dissertation work is devoted to My Family, My Teachers and My Friends. I have a special feeling of gratitude for my beloved family. Special thanks to my supervisor whose uncountable confidence enabled me to reach this milestone

CERTIFICATE OF APPROVAL

Feature Selection using Genetic Algorithm for Android Malware Classification

by

Amir Jibran

(MCS181010)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Muhammad Nazir	HITECH
(b)	Internal Examiner	Dr. Mohammad Masroor Ahmed	CUST, Islamabad
(c)	Supervisor	Dr. Abdul Basit Siddique	CUST, Islamabad

Dr. Abdul Basit Siddique
Thesis Supervisor
May, 2022

Dr. Nayyer Masood
Head
Dept. of Computer Science
May, 2022

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
May, 2022

Author's Declaration

I, **Amir Jibran** hereby state that my MS thesis titled "**Feature Selection using Genetic Algorithm for Android Malware Classification**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Amir Jibran)
Registration No: MCS181010

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Feature Selection using Genetic Algorithm for Android Malware Classification**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Amir Jibran)
Registration No: MCS181010

Acknowledgement

I thanks to Allah Almighty for giving me the power and blessings to finish the thesis. Second, I'd want to thank my supervisor, Dr. Abdul Basit Siddique, for his assistance, time, and supervision. In this field of research, I gratefully acknowledge his aid, motivation, and advise. He assisted me with everything from comprehending the subject to writing the final thesis. My family and parents have been extremely supportive and encouraging during the completion of my MS thesis. Their prayers and guidance have brought me to this point.

(Amir Jibran)

Registration No: MCS181010

Abstract

Android has surpassed iOS as the most widely used smartphone operating system. When compared to prior years, the quick adoption of Android has resulted in a huge increase in the number of malwares. There are numerous antimalware solutions available that are designed to protect users' sensitive data in mobile systems from such attacks. The more accurate the model is, there are increased chances of predicting the malware effectively. Aim is to get higher accuracy with less dimensional data to reduce the computational complexity. Genetic algorithm proved to be the more efficient is reducing the complexity and in gaining higher accuracy to detect malware effectively.

Contents

List of Figures

List of Tables

Abbreviations

AUC	Accuracy
Ci	Chromosome ith Location
Df	Dataframe
FNR	False Negative Ratio
FPR	False Positive Ratio
Fi	Feature at ith Location
Gi	Gene ith Location
HTTP	Hypertext Transfer Protocol
JRIP	RIPPER
KNN	K nearest neighbors
MAX(F)	maximum Fittness
r	Random Number to Cut Chromosome
RF	Random Forest
RIDOR	Ripple-Down Rule learner
Rn	Random Number for Gene Value
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TNR	True Negative Ratio
TPR	True Positive Ratio

Chapter 1

Introduction

1.1 Usage of Smart Phones

Smartphone usage is expanding on a daily basis. For productivity enhancement and compatibility, many online applications have switched their products, availability, and functionalities to this system, and the mobile device has unquestionably become a new growing trend in this modern day. The widespread use of mobile devices has resulted in a significant shift in information security.

1.2 Android OS Popularity

The usage of mobile devices has resulted in an increase in associated threats, such as SMS spam threats, phishing, malware, license to kill spyware, and so on. Because of its open nature, the Android operating system has become the fastest growing mobile operating system, making it the operating system of choice for many consumers and developers. Figure 1.1 below shows the number of smartphone users worldwide.

Hundreds of mobile devices in over 190 countries run on Android. It has the largest installed base of any mobile platform and continues to expand rapidly. Every day, a million new Android users switch on their smartphones for the first time and

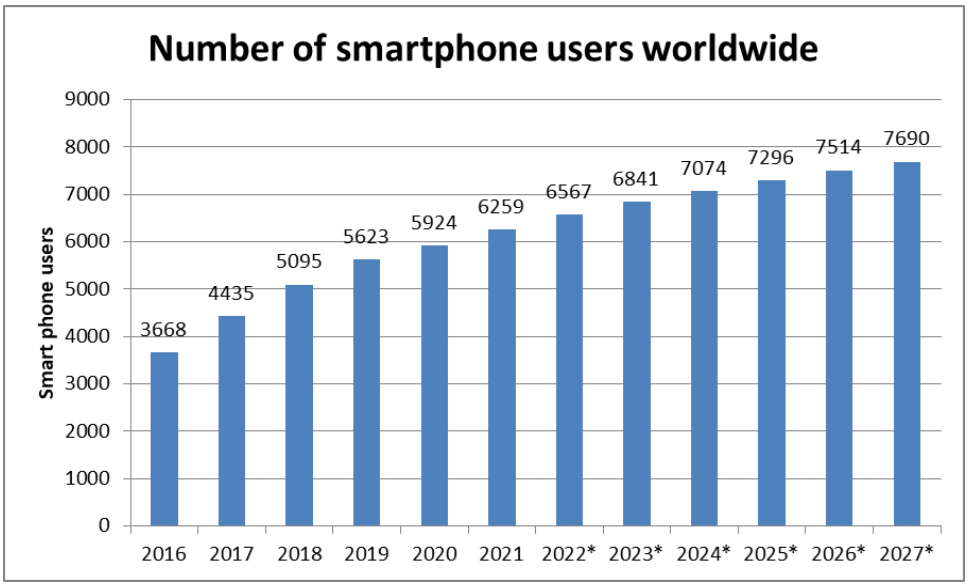

FIGURE 1.1: Number of Smartphone Users Worldwide

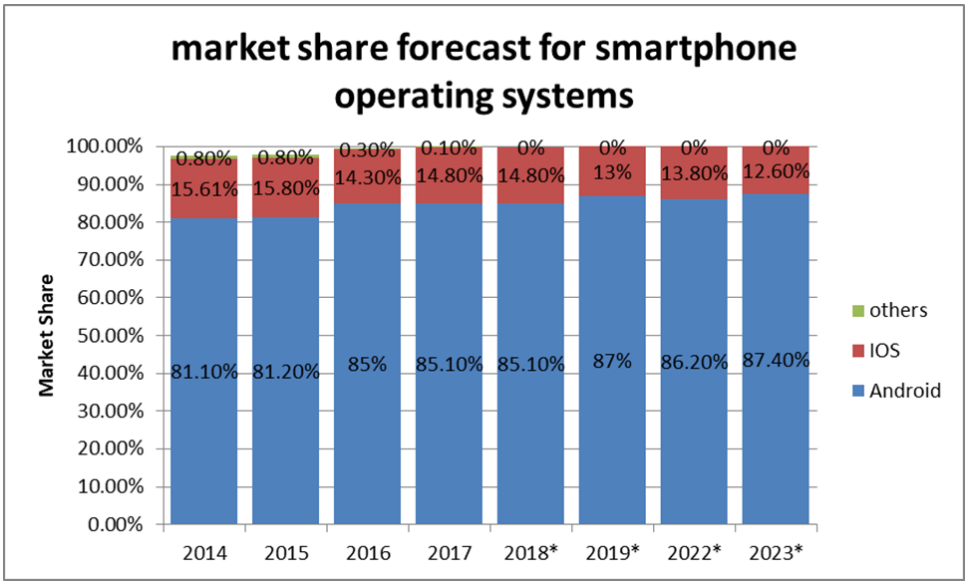

FIGURE 1.2: Market Share Forecast for Smartphone Operating Systems

start looking for apps, games, and other digital content [1]. Figure 1.2 below shows the market share forecast for smartphone operating systems

Android has quickly become the tremendously increasing mobile OS, thanks to the contributions of the open-source Linux community and more than 300 hardware, software, and carrier partners.

Every month, Android users download over 1.5 billion apps and games from Google Play. Its openness has made it a preferred among both users and developers, resulting in rapid growth in app consumption.

1.3 Malicious Intent of Malware

The most frequent types of mobile malware threats are viruses, worms, mobile bots, mobile security breaches, ransomware, spyware, and Trojans. Some mobile malware employs a variety of attack methods. In a cellular context, mobile viruses are utilized to transmit from one exposed phone to another.

1.4 Malware Penetration in Phones

A computer worm is a type of virus that spreads from infected machines to other devices while staying active on them. Cybercriminals can send worms using text messages that use the short chat app (SMS) or the Multimedia Messaging Service (MMS) and do not require user involvement to execute commands. The assault impersonates a trustworthy entity or person and sends out malicious links or attachments that can be used to steal victims' login credentials or account information. Ramsonware is a type of malware that encrypts data on a victim's device or the device itself and then demands payment before decrypting the data or restoring access. Unlike other forms of assaults, the victim is typically warned of its existence and given advice on how to regain the data. The client must deploy a Trojan horse malware. Trojans are frequently inserted into non-malicious executable files or apps on mobile devices by cybercriminals. The Trojan malware is launched when a user clicks or opens a file.

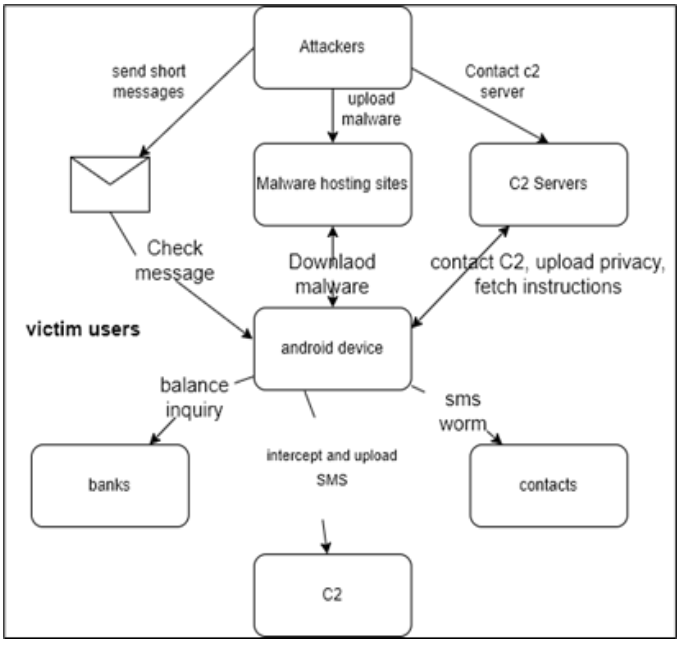

FIGURE 1.3: Threats of Android OS

1.5 Malware Detection

The practice or technique of discovering the source and potential impact of a malware sample is known as malware detection. Malware includes viruses, worms, bugs, Trojan horses, spyware, adware, and other items that appear or act maliciously. Any strange software that has the potential to harm your computer is classified as malware. Despite the widespread use of anti-malware software, malware threats are rapidly evolving around the world. Malware may infect anything with an Internet connection. As potential attackers create new and improved methods of evading detection, malware identification remains a challenge. This is where malware is investigated.

1.6 Feature Selection

Feature selection is a fundamental concept in machine learning that has a significant impact on your model's performance. The data attributes you use to train your machine learning models have a significant impact on the results you can get. Model performance can be harmed by features that are irrelevant or only partially relevant. The first and most critical phase in model design is feature selection and data cleaning. Feature selection is the process of selecting the features that contribute the most to the prediction variable or output that you are interested in, either automatically or manually. The presence of irrelevant characteristics in your data can reduce model accuracy and cause your model to train based on irrelevant features. The feature selection helps in Reducing Over fitting, Improving Accuracy and Reducing Training Time.

1.7 Genetic Algorithm

A genetic algorithm is a search heuristic based on Charles Darwin's natural selection hypothesis. This algorithm mimics natural selection, in which the fittest individuals are chosen for reproduction in order to create the following generation's children.

1. Initial population

   The procedure starts with a group of people known as a Population. Each person is a potential solution to the problem you're trying to solve. Genes are a set of factors (variables) that characterise an individual. A Chromosome is made up of a string of genes (solution).

2. Fitness function

   The fitness function determines a level of fitness (the ability of an individual to compete with other individuals). It assigns each person a fitness score. The fitness score determines the likelihood of an individual being chosen for reproduction.

3. Selection

   The goal of the selection phase is to find the fittest individuals and let them to pass their genes along to future generations. Based on their fitness scores, two pairs of people (parents) are chosen. Individuals who are physically fit have a better probability of being chosen for reproduction.

4. Crossover

   A genetic algorithm's most important phase is crossover. A crossover point is picked at random from within the genes for each pair of parents to be mated.

5. Mutation

   Some of the genes in some new kids can be susceptible to a mutation with a low random frequency. This means that some of the bits in the bit string can be switched around. Mutation happens to maintain population variety and avoid premature convergence.

1.8 Classifiers

1.8.1 K Nearest Neighbors

K-Nearest Neighbor is a Supervised Learning-based Machine Learning algorithm that is one of the most basic. The KNN algorithm assumes that the new case/data and existing cases are similar and places the new case in the category that is most similar to the existing categories. The below mentioned steps explains process of KNN.

1. Determine the number of neighbors (K).
2. Determine the Euclidean distance between K neighbors.
3. Using the obtained Euclidean distance, find the K closest neighbors.
4. Count the number of data points in each category among these k neighbors.
5. Assign the new data points to the category with the greatest number of neighbors.
6. Our model is now complete.

1.8.2 Random Forest

Many decision tree models are used in an ensemble classifier. It can be used for classification or regression, and the results include information on accuracy and variable relevance. Working of Random forest is described in below steps.

1. Let the number of training cases be N, and the number of variables in the classifier be M.
2. The number m of input variables is utilized to decide the choice at each tree node; m should be significantly less than M.
3. Choose N times with replacement from all N possible training instances to create a training set for this tree.
4. By anticipating the classes of the remaining examples, you may estimate the tree's inaccuracy.
5. Choose m variables at random for each node of the tree to base the choice at that node. Calculate the optimum split based on the training set's m variables.
6. Each tree has reached full maturity and has not been pruned.

1.8.3 Decision Tree

Decision Tree is a supervised learning technique that can be applied to classification and regression problems, however it is most commonly employed to solve classification problems. Internal nodes represent dataset attributes, branches represent decision rules, and each leaf node provides the conclusion in this tree-structured classifier. The Decision Node and the Leaf Node are the two nodes of a Decision tree. Leaf nodes are the output of those decisions and do not contain any

more branches, whereas Decision nodes are used to make any decision and have several branches. The decisions or tests are based on the characteristics of the given dataset. It's a graphical depiction for obtaining all feasible solutions to a problem/decision depending on certain parameters. It's termed a decision tree because, like a tree, it starts with a root node and grows into a tree-like structure with additional branches. A decision tree simply asks a question and divides the tree into sub trees based on the answer (Yes/No).

1.9 Problem Statement

The computational complexity increases due to increasing dimension of data. For better android malware analysis and to address the dimensionality problem of data, it is required to perform feature reduction. Android malware classification is a challenging area due to large features space. The classification accuracy can be improved by selecting efficient features using some metaheuristic such as genetic algorithm.

1.10 Research Questions

Q1: How relevant features can be identified for task of malware classification?
Q2: How accuracy of classification for malware detection can be improved using Genetic Algorithm?
Q3: What is the effect of feature reduction?

1.11 Purpose

The purpose of the proposed thesis work is to obtain high accuracy with less number of features set.

1.12 Scope

This research work provides necessary guidelines to improve the accuracy of android malware detection using Genetic Algorithm. The performance metrics as a result of Meta-Heuristic are demonstrated in this research.

1.13 Significance of Solution

This research work contributes in efficiently reducing the dimensionality of the dataset using Meta heuristics. Genetic Algorithm emerged as the effective method in gaining high accuracy with less dimensional data.

Chapter 2

Literature Review

2.1 Analysis Mode

Table 2.1 shows signature based, anomaly based and specification-based analysis modes for android malware detection. Signature based detection uses a trace and behavior to detect malware. A certain trace of android permissions can result in malware as certain android permissions get grant from user to access GPS location, read phone state and identity send SMS messages etc. can result in some malicious activity. Signature-based detection, also known as misuse detection, as defined by [2], keeps track of known intrusion strategies (attack signature) and identifies intrusion by comparing behavior to the database. It will take up less system resources to detect an intrusion [3]. It is also stated that this approach can accurately detect known attacks. Signature based approach can only detect known data accurately because for unknown data signature is also unknown. Anomaly-based approach is superior to signature-based approach in this regard since it can detect unfamiliar data with high accuracy.

Both behavior-based and network-based features are used in the anomaly-based approach. Program specifications that specify the intended behavior of security-critical applications will be used in specification-based detection, according to [4]. As per [5] the purpose of the policy specification language is to provide a straightforward mechanism to describe privileged programmed policies. Rather of

detecting the appearance of certain attack patterns, it monitors the execution of programes and detects deviations in their behavior from the specification. This technique is comparable to anomaly detection in that it detects attacks that are out of the ordinary. Rather of detecting the appearance of certain attack patterns, it monitors the execution of programes and detects deviations in their behavior from the specification. This technique is comparable to anomaly detection in that it detects attacks that are out of the ordinary. According to [6], the advantage of this technique is that it can detect attacks even if they haven't been seen before, and it has a low rate of false alarm. A high incidence of false alarms limited the implementation of anomaly detection in actuality, as [7] uses various machine learning approaches with a TPR of 93.3 percent but an FPR of 31.3 percent.

Arp, D. et al 2014 [8] used signature based static analysis mode for gathering features i.e. without running the application. Burguera, et al 2011 [9] used specification based dynamic analysis mode that gathered features through self-made application "crowd droid". Yerima et al 2014 [2] also used static analysis for feature extraction and malware analysis. Kumar et al 2017 [2] used dynamic based analysis mode. Yerima et al 2015[4] used signature based static analysis mode for malware detection. Feizollah et al 2018 [5] used anomaly based dynamic analysis mode. Amos et al 2013 [6] also used anomaly based dynamic analysis mode. Feizollah et al 2015 [7] used both dynamic anomaly and dynamic specification based analysis mode. Dash et al 2016[10] used anomaly based dynamic analysis mode. Yuan et al 2014 [11] used static signature based and dynamic specification based analysis mode. Feizollah et al 2017 [12], Almin et al 2015 [13], Idrees et al 2014 [14], Yerima et al 2015 [4], Peiravian et al 2013 [20] have used signature based static analysis mode for malware analysis. Karbab et al 2018 [15] used dynamic signature based mode. Canfora et al 2015 [16] used dynamic specification based analysis mode. Hassen et al 2017 [17] and Kang et al 2015 [18] used static specification-based analysis mode. Saxe et al 2015 [19] used dynamic signature based analysis mode. Narouei et al 2013 [20] and Zheng et all 2013 [21] used hybrid signature-based analysis mode. Sato et al 2013 [22], Sanz et al 2013 [23] and Milosevic et al 2017 [24] used static signature based analysis mode. Okazaki et al 2002 [25] used hybrid signature based and hybrid anomaly based analysis mode.

TABLE 2.1: Analysis Mode Comparison

	Signature based detection			Anomaly based detection			Specification based detection		
Paper Ref	**Static**	**Dynamic**	**Hybrid**	**Static**	**Dynamic**	**Hybrid**	**Static**	**Dynamic**	**Hybrid**
Arp, D. 2014 [8]	1	0	0	0	0	0	0	0	0
Burguera, 2011 [9]	0	0	0	0	0	0	0	1	0
Yerima 2014 [2]	1	0	0	0	0	0	0	0	0
Kumar 2017 [2]	0	0	0	0	1	0	0	0	0
Yerima 2015 [26]	1	0	0	0	0	0	0	0	0
Feizollah 2018 [5]	0	0	0	0	1	0	0	0	0
Amos 2013 [6]	0	0	0	0	1	0	0	0	0
Feizollah 2015 [7]	0	0	0	0	1	0	0	1	0
Dash 2016 [10]	0	0	0	0	1	0	0	0	0
Yuan 2014 [11]	1	0	0	0	0	0	0	1	0
Feizollah 2017 [12]	1	0	0	0	0	0	0	0	0
Almin 2015 [13]	1	0	0	0	0	0	0	0	0
Idrees 2014 [14]	1	0	0	0	0	0	0	0	0
Karbab 2018 [15]	0	1	0	0	0	0	0	0	0
Canfora 2015 [16]	0	0	0	0	0	0	0	1	0
Hassen 2017 [17]	0	0	0	0	0	0	1	0	0

	Signature based detection			Anomaly based detection			Specification based detection		
Paper Ref	**Static**	**Dynamic**	**Hybrid**	**Static**	**Dynamic**	**Hybrid**	**Static**	**Dynamic**	**Hybrid**
Kang 2015 [18]	0	0	0	0	0	0	1	0	0
Saxe 2015 [19]	0	1	0	0	0	0	0	0	0
Yerima 2015 [4]	1	0	0	0	0	0	0	0	0
Peiravian 2013 [27]	1	0	0	0	0	0	0	0	0
Narouei 2013 [20]	0	0	1	0	0	0	0	0	0
Zheng et all 2013 [21]	0	0	1	0	0	0	0	0	0
Sato 2013 [22]	1	0	0	0	0	0	0	0	0
Sanz 2013 [23]	1	0	0	0	0	0	0	0	0
Milosevic 2017 [24]	1	0	0	0	0	0	0	0	0
Okazaki 2002 [25]	-	-	1	-	-	1	0	0	0
Sekar 2002 [28]	0	0	0	1	0	0	0	0	0

Arp, D. et al 2014 [8] used signature based static analysis mode for gathering features i.e. without running the application. Burguera, et al 2011 [9] used specification based dynamic analysis mode that gathered features through self-made application “crowd droid”. Yerima et al 2014 [2] also used static analysis for feature extraction and malware analysis. Kumar et al 2017 [2] used dynamic based analysis mode. Yerima et al 2015[26] used signature based static analysis mode for malware detection. Feizollah et al 2018 [5] used anomaly based dynamic analysis mode. Amos et al 2013 [6] also used anomaly based dynamic analysis mode.

2.2 Static Features and Dynamic Features

2.2.1 Static Feature Analysis

Static features can be extracted from an android application without running it. There are tools like android asset packaging tool that can dissemble the APK file and extract the intents used, API calls etc. The API calls that access contacts, network operator information, call state, device id and subscriber id can result in any malicious activity. A certain classifier can be trained on these features to detect anomaly. Permissions can be extracted from android manifest file. CHANGE NETWORK STATE , ACCESS FINE LOCATION, SEND SMS, CHANGE WIFI STATE are some of the network and application based permissions used in malicious activities by the hackers. Android applications use intents for communicating to other applications (explicit intents) and for intra app communication (implicit intents) as well. Some commands used in Linux which can be extracted from an APK file can also trigger the unusual activity by invoking hidden scripts, embedded malicious binary files etc. Technique for analysis used by Arp, D. et al 2014 [8] is static through android asset packaging tool. Static features extracted as a result are applications and network based permissions, implicit and explicit intents and the API calls. Yerima et al 2015[26] also used static analysis but have excluded the intents based features whether they are implicit or explicit. Commands, API calls and permissions based features are extracted by Yerima et al 2014 [2]. Burguera, et al 2011 [9], Kumar et al 2017 [2], Feizollah et al 2018 [5], Amos et al 2013 [6], Feizollah et al 2015 [7], Dash et al 2016[10], Canfora et al 2015 [16], Saxe et al 2015 [19] and Narouei et al 2013 [20] have not used any static analysis method, instead used dynamic analysis method. Yuan et al 2014 [11] gathered features without running the app that is static analysis and extracted features like permissions by analyzing manifest file and API calls. Feizollah et al 2017 [12] used features by dissembling dex file and used intents based features , in addition also used permissions as feature set. Almin et al 2015 [13] only used permissions like WRITE_CALL_LOG, CALL_PHONE, WRITE_SMS, SEND_SMS, INTERNET, CHANGE_WIFI_STATE etc.

Table 2.2 below shows the static features selected in different twenty seven research papers that we are evaluating.

TABLE 2.2: Static Features

	Static Features										
	Permissions		Intents		API Calls					Commands	
Paper Ref	**App**	**Network**	**Implicit**	**Explicit**	**access contacts**	**call state**	**device id**	**subscriber id**	**META DATA**	**RECEIVERS**	**SERVICES**
Arp, D. 2014 [8]	1	1	1	1	1	1	1	1	0	0	0
Burguera, 2011 [9]	0	0	0	0	0	0	0	0	0	0	0
Yerima 2014 [2]	1	1	0	0	1	1	1	1	1	1	1
Kumar 2017 [3]	0	0	0	0	0	0	0	0	0	0	0
Yerima 2015[26]	1	1	0	0	1	1	1	1	1	1	1
Feizollah 2018 [5]	0	0	0	0	0	0	0	0	0	0	0
Amos 2013 [6]	0	0	0	0	0	0	0	0	0	0	0
Feizollah 2015 [7]	0	0	0	0	0	0	0	0	0	0	0
Dash 2016[10]	0	0	0	0	0	0	0	0	0	0	0

		Static Features										
		Permissions		Intents		API Calls					Commands	
Paper	Ref	App	Network	Implicit	Explicit	access contacts	call state	device id	subscriber id	META DATA	RECEIVERS	SERVICES
Yuan 2014	[11]	1	1	0	0	1	1	1	1	0	0	0
Feizollah 2017	[12]	1	1	1	1	0	0	0	0	0	0	0
Almin 2015	[13]	1	1	0	0	0	0	0	0	0	0	0
Idrees 2014	[14]	1	1	1	1	0	0	0	0	0	0	0
Karbab 2018	[15]	0	0	0	0	1	1	1	1	0	0	0
Canfora 2015	[16]	0	0	0	0	0	0	0	0	0	0	0
Hassen 2017	[17]	0	0	0	0	0	0	0	0	0	0	0
Kang 2015	[18]	0	0	0	0	0	0	0	0	0	0	0
Saxe 2015	[19]	1	1	0	0	1	1	1	1	0	0	0

	Static Features										
	Permissions		Intents		API Calls					Commands	
Paper Ref	**App**	**Network**	**Implicit**	**Explicit**	**access contacts**	**call state**	**device id**	**subscriber id**	**META DATA**	**RECEIVERS**	**SERVICES**
Yerima 2015 [4]	1	1	0	0	1	1	1	1	0	0	0
Peiravian 2013 [27]	0	0	0	0	0	0	0	0	0	0	0
Narouei 2013 [20]	0	0	0	0	0	0	0	0	0	0	0
Zheng et all 2013 [21]	1	1	1	1	0	0	0	0	0	0	0
Sato 2013 [22]	1	1	0	0	0	0	0	0	0	0	0
Sanz 2013 [23]	1	1	0	0	0	0	0	0	0	0	0
Milosevic 2017 [24]	0	1	0	0	0	0	0	0	1	1	1
Okazaki 2002 [25]	1	1	1	0	1	1	1	1	1	1	1

Features comprising intents and permissions are used by Idrees et al 2014 [14]. Kang et al 2015 [18] used network based permissions and API calls as there feature set. Yerima et al 2015 [4]

Static features can be extracted from an android application without running it. There are tools like android asset packaging tool that can dissemble the APK file and extract the intents used, API calls etc. The API calls that access contacts, network operator information, call state, device id and subscriber id can result in any malicious activity. A certain classifier can be trained on these features to detect anomaly. Permissions can be extracted from android manifest file. Features comprising intents and permissions are used by Idrees et al 2014 [14]. Kang et al 2015 [18] used network based permissions and API calls as there feature set. Yerima et al 2015 [4], Peiravian et al 2013 [27] have developed their feature set by adding static features like permissions (network and application based) and API calls. Zheng et all 2013 [21] used permissions, intents and API calls as their static features set. Sato et al 2013 [22] used permissions like WRITE _EXTERNAL _STORAGE, CHANGE _WIFI _STATE, CLEAR _APP _CACHE, INSTALL _PACKAGES, INTERNET, CAMERA, CHANGE _CONFIGURATION, CHANGE _NETWORK _STATE and intents , explicit and implicit both. Sanz et al 2013 [23] and Milosevic et al 2017 [24] made same feature set by analyzing android manifest file and used Okazaki et al 2002 [25] used network based permissions and commands as their feature set through static analysis of android applications, like GET_META_DATA, GET_RECEIVERS, GET_SERVICES, GET_ SIGNATURES, GET_PERMISSIONS etc. Permissions, API calls, commands and only implicit intents based features are used by Sekar et al 2002 [28].

2.2.2 Dynamic Feature Analysis

Dynamic features are extracted while running an application and recording some traces in background in CMD. They can be network based and behavior based. Network based features can be extracted from the traffic generated over the network which includes protocols either HTTP or TCP, source and destination ports, packets send and received etc. behavior based includes system calls like open() and kill() etc. A certain malware can open some libraries and can kill them according to their wish causing potential damage to the system Table 2.3 below shows the dynamic features selected in different twenty seven research papers that we are evaluating.

TABLE 2.3: Dynamic Features

Paper Ref	Duration	DP	Pkt Sent	Pkt Rev	PLBytes Sent	PLBytes Rev	TCP size	System calls	Packet size	Source port
Arp, D. 2014 [8]	0	0	0	0	0	0	0	0	0	0
Burguera, 2011 [9]	1	0	0	0	0	0	0	1	0	0
Yerima 2014 [2]	0	0	0	0	0	0	0	0	0	0
Kumar 2017 [3]	1	1	1	1	1	1	0	0	0	0
Yerima 2015[26]	0	0	0	0	0	0	0	0	0	0
Feizollah 2018 [5]	1	0	1	1	0	0	1	0	0	0
Amos 2013 [6]	1	1	-	-	-	-	1	-	1	1
Feizollah 2015 [7]	1	-	-	-	-	-	-	1	1	1
Dash 2016[10]	0	1	0	0	0	0	1	1	0	1
Yuan 2014 [11]	0	0	0	0	0	0	0	0	0	0
Feizollah 2017 [12]	0	0	0	0	0	0	0	0	0	0
Almin 2015 [13]	0	0	0	0	0	0	0	0	0	0
Idrees 2014 [14]	0	0	0	0	0	0	0	0	0	0
Karbab 2018 [15]	0	0	0	0	0	0	0	0	0	0
Canfora 2015 [16]	0	0	0	0	0	0	0	1	0	0
Hassen 2017 [17]	0	0	0	0	0	0	0	1	0	0

Paper Ref	Duration	DP	Pkt Sent	Pkt Rev	PLBytes Sent	PLBytes Rev	TCP size	System calls	Packet size	Source port
Kang 2015 [18]	0	0	0	0	0	0	0	0	0	0
Saxe 2015 [19]	0	0	0		0	0	0	1	0	0
Yerima 2015 [4]	0	0	0	0	0	0	0	0	0	0
Peiravian 2013 [27]	0	0	0	0	0	0	0	0	0	0
Narouei 2013 [20]	0	0	0	0	0	0	0	1	0	0
Zheng et all 2013 [21]	0	0	0	0	0	0	0	0	0	0
Sato 2013 [22]	0	0	0	0	0	0	0	0	0	0
Sanz 2013 [23]	0	0	0	0	0	0	0	0	0	0
Milosevic 2017 [24]	0	0	0	0	0	0	0	0	0	0
Okazaki 2002 [25]	1	1	1	1	0	0	1	0	0	1
Sekar 2002 [28]	0	0	0	0	0	0	0	0	0	0

The method adopted by Burguera, et al 2011 [9] is dynamic analysis and have used behavior based features like system call and also network traffic related features like duration period of the connection. Kumar et al 2017 [2] used wireshark for analyzing the network traffic and extracted features like duration period , destination port, no of packets sent and received and payload bytes sent and received. The same method is adopted by Feizollah et al 2018 [5] as Burguera, et al 2011 [9] the only difference is the feature selection.

The method adopted by Burguera, et al 2011 [9] is dynamic analysis and have used behavior based features like system call and also network traffic related features like duration period of the connection. Kumar et al 2017 [2] used wireshark for analyzing the network traffic and extracted features like duration period , destination port, no of packets sent and received and payload bytes sent and received. The same method is adopted by Feizollah et al 2018 [5] as Burguera, et al 2011 [9] the only difference is the feature selection. In their proposed method they used duration, no of packets sent and received and the size of TCP protocol. Amos et al 2013 [6] also used traffic based features which includes duration time, source and destination port, TCP size and packet size. Feizollah et al 2015 [7] gathered behavior based features like system calls and network related features like duration, packet size and source port. Dash et al 2016[10] choose source and destination ports and TCP size as their network based features and also system calls as their dynamic features through running the android application individually. Feature set of Canfora et al 2015 [16],Hassen et al 2017 [17] and Saxe et al 2015 [19] contains only system calls that were dynamically extracted. Narouei et al 2013 [20] used suspicious system calls like read (), open (), access (), chmod () and chown () as their feature set. The feature set of Okazaki et al 2002 [25] includes source and destination port of the network connection used, TCP size used, duration time for which the connection exits and finally the number of packets sent and received.

2.3 Data set and Evaluation

In this Section, the dataset and validation techniques of twenty seven research papers have been presented. Dataset attribute contains furthermore four sub attribute which were training dataset, test dataset, malicious apps which are tested and benign app. Training dataset is required to make your algorithm better by learning some pattern. Test dataset is set of data (Malicious. Benign app) to find accuracy of any proposed algorithm. Benign app are those which were not harmful where Malicious apps are those apps which were harmful. Also in following table we have online/offline learning, percentage split method and as K-fold validation technique.

Arp, D. et al 2014 [8] used 5,560 Malicious and 123,453 Benign app's as a dataset and use offline learning with 66/33 Percentage split Method to evaluated their machine learning algorithm. Burguera, et al 2011 [9] used 10 Malicious and 50, Benign app's as a dataset and use offline learning. Yerima et al 2014 [2] used 2,925 Malicious and 3,938 Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Kumar et al 2017 [2] does not provide data set information. They use offline learning with 70/30 Percentage split Method and 10 fold cross validation method to evaluate their machine learning algorithm. Yerima et al 2015[26] they used 2,925 Malicious and 3,938 Benign app's as a dataset and use offline learning with 10 fold cross validation approach to evaluated their machine learning algorithm. Feizollah et al 2018 [5] used 800 Malicious and 100 Benign app's as a dataset and use offline learning. Amos et al 2013 [6] used 1330 Malicious and 408 Benign app's as a dataset and use offline learning with 10 fold cross validation to evaluated their machine learning algorithm. Feizollah et al 2015 [7] used 600 Malicious and Benign app's as a dataset for android malware analysis. Dash et al 2016[10] used 5,246 Malicious and Benign app's as a dataset and use offline learning with 20 fold cross validation technique to evaluated their machine learning algorithm. Yuan et al 2014 [11], they used 300 dataset as a training dataset and 200 as a test dataset with 250 Malicious and 250 Benign app's and use offline learning. Feizollah et al 2017 [12] used 600 dataset as a training dataset and 100 dataset as a testing dataset with 380 Malicious and 320 Benign app's and use offline learning, Percentage split Method to evaluated their machine learning algorithm. Almin et al 2015 [13] not provided information of their dataset and validation technique. Idrees et al [14] used 292 apps as training dataset and 340 apps as a test dataset with 45 Malicious and 300 Benign app's and use offline learning.

Karbab et al 2018 [15] used 33,000 Malicious and 38,000 Benign app's as a dataset and use offline learning with 2,3,5,10 fold method to evaluated their machine learning algorithm. Table 2.4 shows the data set collected (malicious and benign) and evaluation techniques applied on that dataset, in different twenty seven research papers that we are evaluating.

TABLE 2.4: Data Set And Evaluation

Paper Ref	**Training**	**Test**	**Malicious**	**Benign**	**Online/ offline learning**	**Percentage split Method**	**K-fold cross validation technique**
Arp, D. 2014 [8]	-	-	5,560	123,453	offline	66/33	0
Burguera, 2011 [9]	-	-	10	50	offline	-	-
Yerima 2014 [2]	-	-	2,925	3.938	offline	-	10
Kumar 2017 [3]	-	-	-	-	offline	70/30	10
Yerima 2015[26]	-	-	2,925	3,938	offline	-	10
Feizollah 2018 [5]	-	-	800	100	offline	-	-
Amos 2013 [6]	-	-	1330	408	online	-	10
Feizollah 2015 [7]	-	-	600	-	-	-	
Dash 2016[10]	-	-	5,246	offline	-	20	
Yuan 2014 [11]	300	200	250	250	offline	-	-
Feizollah 2017 [12]	600	100	380	320	Offline	1	0
Almin 2015 [13]	0	0	0	0	Offline	0	0
Idrees 2014 [14]	292	340	45	300	Offline	0	0
Karbab 2018 [15]	0	0	33000	38000	Offline	0	2, 3, 5 & 10-fold
Canfora 2015 [16]	1600	400	1000	1000	Offline	0	0
Hassen 2017 [17]	80%	20%	1200	1113	Offline	1	10-Fold

Paper Ref	Training	Test	Malicious	Benign	Online/ offline learning	Percentage split Method	K-fold cross validation technique
Kang 2015 [18]	-	-	4554	51179	Offline	0	5-Fold
Saxe 2015 [19]	-	-	350,016	81,910	Offline	0	4-Fold
Yerima 2015 [4]	-	-	2925	3938	Offline	0	10-Fold
Peiravian 2013 [27]	-	-	>1200	>1200	Offline	0	10-Fold
Narouei 2013 [20]	9/10	1/10	11 000	4700	Offline	0	10-Fold
Zheng et all 2013 [21]	0	0	1440	563	Offline	0	0
Sato 2013 [22]	94	271	130	235	Offline	1	0
Sanz 2013 [23]	0	0	333	333	Offline	0	10, 50 and 100-fold
Milosevic 2017 [24]	0	0	200	200	Offline	0	0
Okazaki 2002 [25]	0	0	0	0	0	0	0
Sekar 2002 [28]	9/10	1/10	2925	3938	Offline	0	10-Fold

Canfora et al 2015 [16] used 1600 dataset as training dataset and 400 as a test dataset with 1000 Malicious and 1000 Benign app's as a dataset and use offline learning. Hassen et al 2017 [17] they used 80% dataset as a training dataset and 20% dataset as a test dataset with 1200 Malicious and 1113 Benign app's as a dataset and use offline learning, Percentage split Method and 10 fold cross validation technique to evaluated their machine learning algorithm. Kang et al 2015 [18] used 4,554 Malicious and 51,179 Benign applications as a dataset and use offline learning with 5 fold cross validation technique to evaluated their machine learning algorithm.

TABLE 2.5: Machine Learning Techniques

Paper Ref	Probabilistic			Function Based		Tree Based				Rule Based				Clustering	
	Naïve Bayes	K Star	Prism	Simple Logistic	SVM	J48	C4.5	ID3	RF	JRIP	RIDOR	PART	NN	K Means	C Means
Arp, D. 2014 [8]	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Burguera, 2011 [9]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Yerima 2014 [2]	1	0	0	1	0	1	0	0	0	0	1	1	0	0	0
Kumar 2017 [3]	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0
Yerima 2015[26]	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
Feizollah 2018 [5]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Amos 2013 [6]	1	0	0	1	0	1	0	0	1	0	0	0	1	0	0
Feizollah 2015 [7]	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Dash 2016[10]	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Yuan 2014 [11]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Feizollah 2017 [12]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Almin 2015 [13]	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Idrees 2014 [14]	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Karbab 2018 [15]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Canfora 2015 [16]	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Hassen 2017 [17]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kang 2015 [18]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Saxe 2015 [19]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Yerima 2015 [4]	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0

Paper Ref	Probabilistic			Function Based		Tree Based				Rule Based				Clustering	
	Naïve Bayes	K Star	Prism	Simple Logistic	SVM	J48	C4.5	ID3	RF	JRIP	RIDOR	PART	NN	K Means	C Means
Peiravian 2013 [27]	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
Narouei 2013 [20]	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0
Zheng et all 2013 [21]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sato 2013 [22]	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Sanz 2013 [23]	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0
Milosevic 2017 [24]	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0
Okazaki 2002 [25]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sekar 2002 [28]	1	0	0	1	0	0	0	0	1	0	0	0	0	0	

Saxe et al 2015 [19] used 350,016 Malicious and 81,910 Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Yerima et al 2015 [4] used 2,925 Malicious and 3,938 Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Peiravian et al 2013 [27] used more than 2400 Malicious and Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Narouei et al 2013 [20] used 9/10 dataset as a training dataset and 1/10 as a test dataset with 11000 Malicious and 4,700 Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Zheng et al 2013 [21] used 1440 Malicious and 563 Benign app's as a dataset and use offline learning with 10 fold cross validation technique to evaluated their machine learning algorithm. Sato et al 2013 [22], they used 94 dataset as training dataset and 271 as test data set having 130 Malicious and 235 Benign app's and use offline learning. Sanz et al 2013 [23].

Okazaki et al 2002 [25], they didn't provide the dataset information. Sekar et al 2002 [28], they used 9/10 dataset examples as training dataset and 1/10 dataset examples as test dataset having 2,925 Malicious and 3,938 Benign app's and use offline learning with 10 fold cross validation technique to evaluate their machine learning algorithm.

2.4 Machine Learning Algorithms

Table 2.5 shows different machine learning techniques used by authors. The technique of machine which Arp, D. et al 2014 [8] follow is function based machine learning in which they used SVM machine learning algorithm to classify there dataset. Burguera, et al 2011 [9] doesn't use any machine learning algorithm for classification. For clustering K-mean is used. Yerima et al 2014 [2] used Probabilistic, Function Based ML, Tree Based ML and Rule Based ML in which they perform classification with Naïve Bayes, Simple Logistic, J48, PART and RIDOR. Kumar et al 2017 [2] perform Tree Based ML, Rule Based ML classification in which they used J48, RF, JRIP, RIDOR, PART. For classification Yerima et al 2015[26] used Probabilistic, Function Based ML and Tree Based ML techniques in which includes Naïve Bayes, simple logistic, J48 and RF. Feizollah et al 2018 [5] used no machine learning algorithm for classification, for clustering they used C-Means. Amos et al 2013 [6] used Neural Network, Probabilistic, Function Based ML and Tree Based ML techniques. Feizollah et al 2015 [7] used Tree Based ML (RF) is used for classification. Dash et al 2016[10] used Function Based ML is used in which they use SVM. Yuan et al 2014 [11] used Neural Network for classification of dataset. Feizollah et al 2017 [12] used Probabilistic machine learning technique is use in which they used Naïve Bayes algorithm. Almin et al 2015 [13] used Naïve Bayes for classification and K-mean for clustering. Idrees et al [14] used Probabilistic machine learning technique in which Naïve Bayes, K-star Prism is used. Karbab et al 2018 [15] used Neural Network for classification. Canfora et al 2015 [16] used Function Based machine learning is used for classification which includes SVM. Hassen et al 2017 [17] and Kang et al 2015 [18] doesn't use any

machine learning based technique. Saxe et al 2015 [19] used Neural Network for classification. Yerima et al 2015 [4] used Probabilistic, Function Based ML and Tree Based ML which specifies following Naïve Bayes, Simple Logistic and RF algorithms to perform classification. Function and Tree Based machine learning technique is used for classification by Peiravian et al 2013 [27] in which they used simple logistic and J48. Probabilistic and Tree Based machine learning is used for classification by Narouei et al 2013 [20] in which they use Naïve Bayes, J48 and RF. Zheng et al 2013 [21] and Okazaki et al 2002 [25] didn't use any machine learning based algorithm. Sato et al 2013 [22] used Tree based Machine learning (J48) for classifying the dataset. Sanz et al 2013 [23] used Probabilistic (Naïve Bayes), Function Based ML (SVM), Tree Based ML (J48,RF) and Neural Network for classification. Probabilistic (Naïve Bayes), Function (SVM), Tree (C4.5, RF) and rule based (JRIP) machine learning algorithm is used by Milosevic et al 2017 [24]. Probabilistic (Naïve Bayes), Function Based ML (Simple Logistic) and Tree Based ML (RF) is used to classifying the dataset by Sekar et al 2002[28].

2.5 Performance Metrics

Table 2.6 shows the performance matrices of various authors. In this section, performance metrics of different literature have been compared to analyze the efficiency of research works. Ensemble techniques are also discussed in this table. In ensemble technique, multiple classifiers are ensemble to improve results efficiency by voting, bagging, or boosting.

The following are the evaluating metrics that are used by the researchers.

1. TPR: The True positive rate (TPR) is the percentage of true forecasts in positive class predictions

$$TPR = TP/P$$

2. TNR: The True negative rate (TPR) is the percentage of true forecasts in negative class predictions

$$TNR = TN/N$$

3. FPR: The false positive rate is the percentage of inaccurate positive predictions

$$FPR = FP/N$$

4. FNR: The false negative rate is the percentage of inaccurate negative predictions

$$FNR = FN/P$$

5. Accuracy: The proportion of correct predictions contained within your model is measured by accuracy

$$Accuracy = (TP + TN)/(TP + TN + FP + FN)$$

6. AUC: The area under the recall and false positive rate curves is determined by the area under the receiver operating characteristic curve. It compares the sensitivity to the rate of fallout. The True positive rate vs the False positive rate with respect to a threshold T is plotted parametrically as the Area under ROC.

In Arp, D. et al 2014 [8] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, True positive rate (tpr) of 96%, and 1% false positive rate (fpr). In Burguera, et al 2011 [9] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 100%, which is very promising and 0% fpr. In Yerima et al 2014 [2] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 95.8%, true negative rate (tnr) of 95.7%, false negative rate(fnr) of 4.2%, 3.30% fpr. They obtained 96.3% accuracy with error rate of 3.7% and area under the roc curve was 97%. Alongside they performed ensemble techniques, which are average probability (avg prob), product probability (prod prob), maximum probability (max prob), and voting based ensemble technique. In Kumar et al 2017

[2] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 97.9%, true negative rate (tnr) of 97.9%, false negative rate(fnr) of 0%, 2.10% fpr. They obtained 98.20% accuracy, and area under the roc curve (auc) was 98.9 %. Alongside they performed ensemble techniques, which are avg prob, prod prob, max prob, and voting based ensemble technique. In Yerima et al 2015[26] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 97.3 %, 2.30 % fpr, tnr of 97.9 %. They obtained 97.50% accuracy, with 2.50% error rate and auc was 99.3%. Alongside they performed ensemble techniques, which are bagging and boosting based ensemble technique. In Feizollah et al 2018 [5] they didn't provide performance metrics. In Amos et al 2013 [6] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 93.3%, and 31.03% fpr. In Feizollah et al 2015 [7] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 99.6%, and 0.4% fpr.Alongside they performed boosting based ensemble technique. In Dash et al 2016[10] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, They achieved 94% of accuracy. In Yuan et al 2014 [11] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, They achieved 96.50% of accuracy. In Feizollah et al 2017 [12], Almin et al 2015 [13], Idrees et al 2014 [14] they didn't provide performance metrics. In Karbab et al 2018 [15] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, They achieved tpr of 96.99%, and 0.06-2% fpr.

TABLE 2.6: Performance Metrics

Paper Ref	**TPR**	**TNR**	**FPR**	**FNR**	**Accuracy**	**Error**	**AUC**
Arp, D. 2014 [8]	96%	-	1%	-	-	-	-
Burguera, 2011 [9]	100%	-	0%	-	-	-	-
Yerima 2014 [2]	95.8%	96.7%	3.30%	4.2%	96.30%	3.70%	97%
Kumar 2017 [3]	100%	97.9%	2.10%	0%	98.20%	-	98.9%

Paper Ref	**TPR**	**TNR**	**FPR**	**FNR**	**Accuracy**	**Error**	**AUC**
Yerima 2015[26]	97.3%	97.7%	2.30%	-	97.50%	2.50%	99.3%
Feizollah 2018 [5]	-	-	-	-	-	-	-
Amos 2013 [6]	93.3%	-	31.03%	-	-	-	-
Feizollah 2015 [7]	99.6%	-	0.40%	-	-	-	-
Dash 2016[10]	-	-	-	-	94%	-	-
Yuan 2014 [11]	-	-	-	-	96.50%	-	-
Feizollah 2017 [12]	0	0	0	0	0	0	0
Almin 2015 [13]	0	0	0	0	0	0	0
Idrees 2014 [14]	0	0	0	0	0	0	0
Karbab 2018 [15]	96.99%	0	0.06-2%	0	0	0	0
Canfora 2015 [16]	97%	0	3%	3%	0	0	97%
Hassen 2017 [17]	96%	-	0.01%	-	99.3%	-	99%
Kang 2015 [18]	-	-	-	-	98%	-	-
Saxe 2015 [19]	95.2%	-	0.1%	-	95%	-	0.99964
Yerima 2015 [4]	97.3%	97.7%	2.3%	-	97.5%	2.5%	99.3%
Peiravian 2013 [27]	-	-	-	-	96.88%	-	96.3%
Narouei 2013 [20]	98.5%	0	0	0	0	0	0
Zheng et all 2013 [21]	0	0	0	0	0	0	0
Sato 2013 [22]	90.0%	0	10.0%	0	0	0	0
Sanz 2013 [23]	94%	0	5%	0	94.83%	0	98%
Milosevic 2017 [24]	82.3%	0	17.6%	0	0	0	0
Okazaki 2002 [25]	0	0	0	0	0	0	0
Sekar 2002 [28]	97.2%	0	2.5%	0	97.6%	0	99.3%

In Canfora et al 2015 [16] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 97%, 3% fpr, fnr of 3%. They obtained 97% of auc. In .Hassen et al 2017 [17] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 96%, 3% fpr, fnr of 0.01%. They obtained 0.993 of accuracy, and 99% of auc. In Kang et al 2015 [18] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 98% of auc. In Saxe et al 2015 [19] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of

95.2%, 0.1% fpr They obtained 95% of accuracy. In Yerima et al 2015 [4] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 0.973, 2.3 % fpr, tnr of 97.7 %. They obtained 97.5% accuracy, with 2.50% error rate and auc was 99.3%. In Peiravian et al 2013 [27] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 96.88% accuracy, and auc was 96.3%. In Narouei et al 2013 [20] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 98.5% of tpr. In Zheng et al 2013 [21], Okazaki et al 2002 [36] they didn't provide the metrics results. In Sato et al 2013 [22] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 90.0% of tpr and 10 % of fpr. In Sanz et al 2013 [23] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 94% of tpr, 5% of fpr, 94.83% of accuracy, and 98% of auc. In Milosevic et al 2017 [24] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression. They obtained 82.3% of tpr and 17.6 % of fpr. In Sekar et al 2002 [28] they achieved following performance metrics, while performing machine learning algorithms for malware and benign program classification or regression, tpr of 97.2%, 2.5% fpr. They obtained 97.6% of accuracy and 99.3% of auc.

2.6 Experimental Setup

In this section, the experimental setup of different research papers is presented. Some common attributes were figured out on which different research paper conduct their experiments. These attribute are Frequency, RAM. Server , Desktop, Phone, Processor, and OS. In Arp, D. et al 2014 [8] they used desktop computer system with following specifications, Core 2 duo processor with 2.26 Gigahertz

(GHz) frequency and 4 Gigabyte (GB) of RAM. They used virtual machine for analysis of malware. In Burguera, et al 2011 [9], Yerima et al 2014 [2], Kumar et al 2017 [2], Yerima et al 2015[26] they didn't provide the experimental setup details. In Feizollah et al 2018 [5] they used desktop computer system with following specifications, Intel core i5 and 20GB of RAM using Windows 7 operating system. They used virtual machine for analysis of malware. In Amos et al 2013 [6] they used desktop computer system with following specifications, Intel Xeon 5645 and 36 GB Double Data Rate 3 (DDR3) of RAM using CentOS 6.3 operating system. They used virtual machine for analysis of malware. In Feizollah et al 2015 [7] they used desktop computer and phone for android malware analysis. They used virtual machine for analysis of malware. Further they didn't provide the system specifications. In Dash et al 2016[10],Yuan et al 2014 [11], they didn't provide the experimental setup details. In Feizollah et al 2017 [12] they used mobile with following specifications, 2 GB of RAM using Marshmallow, version 6.0.1 on Android operating system. In Almin et al 2015 [13] they used mobile using Jelly Bean, version 4.2.2 on Android operating system. In Idrees et al 2014 [14] they used desktop computer system with following specifications, Intel Core i3-3220 processor with 3.30 GHz frequency. They used virtual machine for analysis of malware. In Karbab et al 2018 [15] they used desktop computer system with following specifications, Intel E5-26301, T64001, ARM-A7 processor with 3.30 GHz frequency and RAM of 128GB/ 3 GB/ 1 GB. They used mobile and virtual machine for analysis of malware.

In Canfora et al 2015 [16] they used desktop computer system with following specifications, Intel Core i5 processor and RAM of 4GB. They used mobile and virtual machine for analysis of malware.

In Hassen et al 2017 [17] they used desktop computer system with following specifications, quad core processor with frequency 2.3 GHz and RAM of 8GB. They used virtual machine for analysis of malware.

In Kang et al 2015 [18] they used desktop computer system with following specifications, Intel Xeon X5660 and RAM of 4GB. They used virtual machine for analysis of malware.

TABLE 2.7: Experimental Setup

Paper Ref	Frequency	RAM	Server	Desktop	Phone	Processor	OS
Arp, D. 2014 [8]	2.26 GHz	4GB	-	1	-	Core 2 duo	-
Burguera, 2011 [9]	-	-	-	-	-	-	-
Yerima 2014 [2]	-	-	-	-	-	-	-
Kumar 2017 [3]	-	-	-	-	-	-	-
Yerima 2015[26]	-	-	-	-	-	-	-
Feizollah 2018 [5]	-	20 GB	-	1	-	Intel i5	Win 7
Amos 2013 [6]	-	36GB	DDR 3	1	1	Intel Xeon 5645	CentOS 6.3
Feizollah 2015 [7]	-	-	-	1	1	-	-
Dash 2016[10]	-	-	-	-	-	-	-
Yuan 2014 [11]	-	-	-	-	-	-	-
Feizollah 2017 [12]	-	2GB	0	0	1	Marshmallow, version 6.0.1	Android
Almin 2015 [13]	-	-	0	0	1	Jelly Bean, version 4.2.2	Android
Idrees 2014 [14]	3.30GHz	-	0	1	0	Intel Core i3-3220	-
Karbab 2018 [15]	-	128GB					
/3 GB							
/1 GB	1	1	1	Intel E5-26301, T64001, ARM-A7	-		
Canfora 2015 [16]	-	4GB	0	1	0	Intel Core i5	-
Hassen 2017 [17]	2.3 GHz	8GB	0	1	0	quad core	-
Kang 2015 [18]	-	4GB	0	1	0	Intel Xeon X5660	-
Saxe 2015 [19]	-	80GB	1	0	0	Amazon EC2	-
Yerima 2015 [4]	-	-	-	-	-	-	-

Paper Ref	Frequency	RAM	Server	Desktop	Phone	Processor	OS
Peiravian 2013 [27]	2.4GHz	2GB	0	1	0	Intel Core 2 Duo PC	-
Narouei 2013 [20]	2 GHz	8GB	0	1	0	Intel Core i7	Windows
Zheng et all 2013 [21]	0	0	1	1	0	two virtual machines	-
Sato 2013 [22]	0	0	0	0	0	0	-
Sanz 2013 [23]	0	0	0	0	0	0	-
Milosevic 2017 [24]	2.3 GHz	2GB	0	0	1	Quad-core Nexus 5	Android
Okazaki 2002 [25]	0	0	0	1	0	0	0
Sekar 2002 [28]	0	0	0	1	0	0	0

In Saxe et al 2015 [19] they used desktop computer system with following specifications, Amazon EC2 and RAM of 80GB. They used virtual machine for analysis of malware.

In Yerima et al 2015 [4] they didn't provide the experimental setup details.

In Peiravian et al 2013 [27] they used desktop computer system with following specifications, Intel Core 2 Duo PC processor with frequency 2.4 GHz and RAM of 2GB. They used virtual machine for analysis of malware.

In Narouei et al 2013 [20] they used desktop computer system with following specifications, Intel Core i7 processor with frequency 2.0 GHz and RAM of 2GB on windows operating system. They used virtual machine for analysis of malware.

In Zheng et al 2013 [21] they used desktop computer system with following specifications, two virtual machines. They used virtual machine and server systems for analysis of malware

In Sato et al 2013 [22], Sanz et al 2013 [23] they didn't provide the experimental setup details.

In Milosevic et al 2017 [24] they used Nexus 5 mobile with following specifications, quad core processor with frequency 2.3 GHz and RAM of 2GB. They used android operating system for analysis of malware.

In Okazaki et al 2002 [25], Sekar et al 2002 [28] they used desktop computer systems. Further they didn't provide the system details. They used virtual machine for analysis of malware.

2.7 Discussion

In order to produce a good antimalware tool, a few studies, such as [29], [30], and [2], have been found to adjust this detection technique by combining whether Signature-based with Anomaly-based detection method or Anomaly-based with Specification-based detection method. Researchers such as [30], described a detailed analysis methodology for generating a new signature based detection on code level of the malware or benign file. In this thesis, a malware detection taxonomy is demonstrated to be effective by comparing it to the present malware detection technique: Table 2.1 shows competence criteria for signature-based detection, anomaly-based detection, and specification-based detection. All of these strategies are unable to reduce false alert due to their inability to reduce either FNR or FPR alert. The use of static and dynamic features, as shown in Tables 2.2 and 2.3, suggests that there is still potential for improvement in terms of lowering false alarms. In this thesis, based on the machine learning analyses, ensemble based techniques [2, 3, 12, 14, 16] have been proven more effective in classifying the data, as compared to other techniques. We have also compared the experimental setups utilized by various researchers, in order to perform the methodologies in their research.

Chapter 3

Methodology

This chapter will be explaining the working methodology of thesis. The working methodology is divided into three phases. In phase 1 the preprocessing of the dataset has been performed. Correlation technique has been used to preprocess the data. As a result of correlation the features were reduced from 80 to 57 that will be explained in section 3.1.2. In phase 2 the feature selection technique is discussed. Genetic algorithm has been used to reduce the feature set of dataset. The features were reduced from 57 to 35 as a result of Genetic algorithm when elitism rate was 0.30 and mutation rate was 1 with a fitness function value of 90.04%. In phase 3 final reduced dataset was compiled and that dataset was then used to evaluate the classification of android malware families. The evaluation was performed for three categories i.e binary, malware category and malware family.

3.1 Phase 1-PreProcessing

The dataset named 'CICAndMal2017' used is the labeled dataset with 80 dynamic features of 42 malware families. Lashkari.et.al [31] gathered about 10,854 samples from various sources (4,354 malware and 6,500 benign) and they also gathered over 6,000 Benign apps from the Google Play store in 2015, 2016, and 2017. The actual malware behavior will trigger once installed on real smart phones that's why over 5,000 of the gathered samples (426 malware and 5,065 benign) were installed on

real smart devices to get the network traces and eventually extracted 80 unique dynamic features.

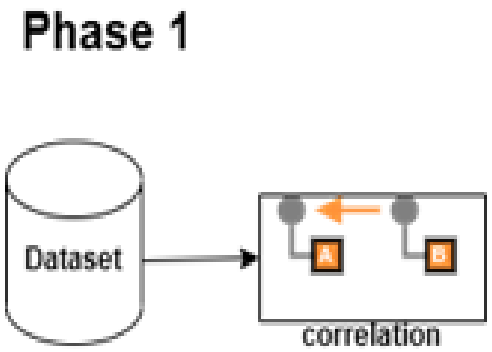

FIGURE 3.1: Phase I - Preprocessing

3.1.1 Dataset

The dataset used for feature selection and classification is collected by Lashkari.et.al [31]. The dataset is completely labeled and includes network traffic, logs, API/SYS calls, phone statistics, and memory dumps of 42 malware families. The table 3.1 below shows the description of each feature extracted in this dataset. Table 3.2 shows the malware families names and category C labeled accordingly.

TABLE 3.1: Feature Description of Dataset

Feature name	**Description**
Source_Port	
Destination_Port	
Protocol	
Flow_Duration	Flow duration
Total_Fwd_Packets	Total packets in the forward direction
Total Backward Packets	Total packets in the backward direction
Total Length of Fwd Packets	Total size of packet in forward direction
Fwd Packet Length Max	Maximum size of packet in forward direction

Feature name	**Description**
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Mean	Average size of packet in forward direction
Fwd Packet Length Std	Standard deviation size of packet in forward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Flow Bytes/s	flow byte rate that is number of packets transferred per second
Flow Packets/s	flow packets rate that is number of packets transferred per second
Flow IAT Mean	Mean time between two flows
Flow IAT Std	Standard deviation time two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT Total	Total time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Total	Total time between two packets sent in the backward direction

Feature name	**Description**
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Bwd_Packets/s	Number of backward packets per second
Min Packet Length	Minimum length of a flow
Max Packet Length	Maximum length of a flow
Packet Length Mean	Mean length of a flow
FIN Flag Count	Number of packets with FIN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
Fwd Avg Bulk Rate	Average number of bulk rate in the forward direction

Feature name	**Description**
Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction
Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
Init_Win_bytes_backward	# of bytes sent in initial window in the backward direction
act_data_pkt_fwd	# of packets with at least 1 byte of TCP data payload in the forward direction
min_seg_size_forward	Minimum segment size observed in the forward direction
Active Mean	Mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Mean	Mean time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active

TABLE 3.2: Malware Families Names

Main Catgeory	**Adware**	**Ransomware**	**Scareware**	**SMS Malware**
Families	Dowgin	Charger	AndroidDefender	BeanBot
	Ewind	Jisut	AndroidSpy	Biige
	Feiwo	Koler	AV for Android	FakeInst
	Gooligan	LockerPin	AVpass	FakeMart
	Kemoge	Simplocker	FakeApp	FakeNotify
	koodous	Pletor	FakeAppAL	Jifake
	Mobidash	PornDroid	FakeAV	Mazarbot
	Selfmite	RansomBO	FakeJobOffer	Nandrobox
	Shuanet	Svpeng	FakeTaoBao	Plankton
	Youmi	WannaLocker	Penetho	SMSsniffer
			VirusShield	Zsone

3.1.2 Correlation

For feature reduction, correlation is applied. The goal is to minimize the dataset's dimensionality by removing strongly associated features. It is essentially a measure of how closely two variables, X and Y, are connected to one another. Correlation, in other terms, is an indicator of the magnitude of a linear relationship between two variables. The correlation coefficient 'r' might be anything between –1 and 1. Correlation is a quantity that has no dimensions. The units of measurement of X and Y have no bearing on correlation. If the correlation is higher than 0, the two variables are considered to be positively correlated, meaning that as X rises, Y rises as well. A perfect positive correlation has a r = 1. If the correlation is less than zero, it suggests that the two variables are not related. Figure 3.2 below describes the process how correlation is being used in order to get our reduced feature set.

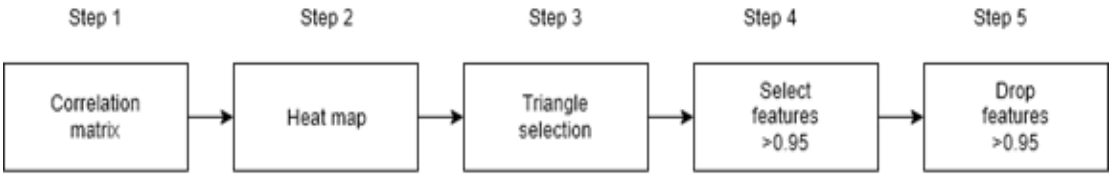

FIGURE 3.2: Process of Finding the Correlation

Below mentioned are the steps that describes how features greater the 0.95 have been selected and at the end we get the dataset in which all features have less than 0.95 correlation.

Step 1: Finding correlation matrix.

Step 2: Calculating the heat map. Figure 3.3 below shows the Heap map.

Step 3: Then select upper or lower triangle.

Step 4: Then select features with correlation more than 0.95.

Step 5: Drop the selected features having correlation more than 0.95.

The total number of features extracted by Lashkari.et.al [31] is 80. After using correlation the total number of features was reduced to 57.

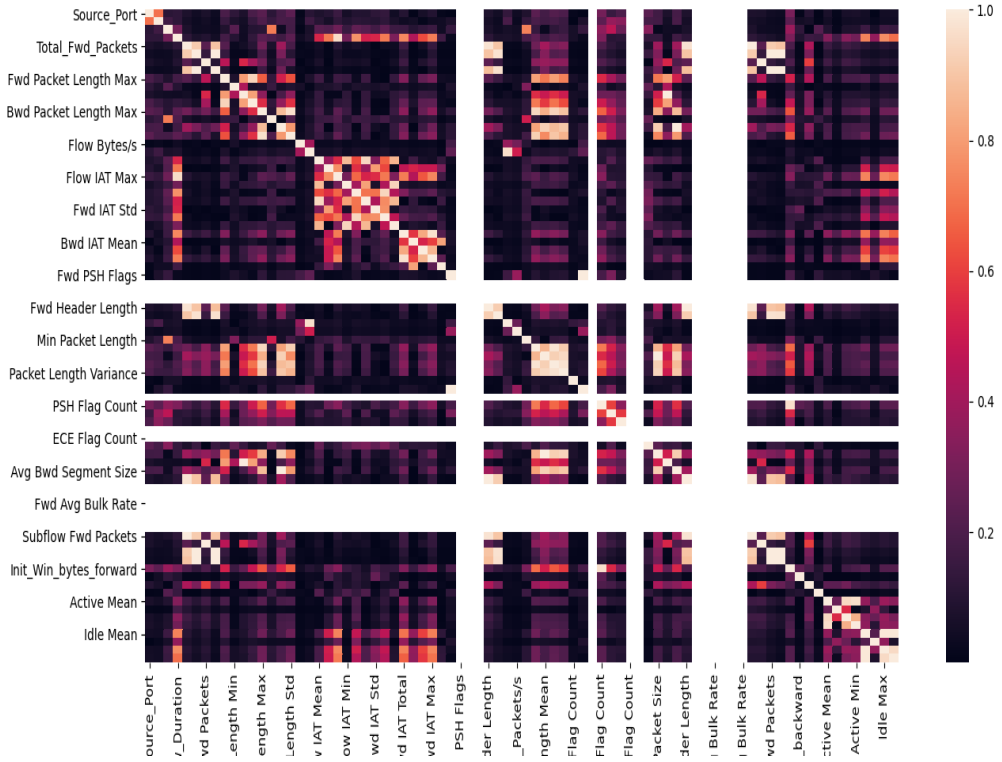

FIGURE 3.3: Heatmap

The figure 3.4 below shows the snap shot of the implementation in python.

```
def corelation():
    # Create correlation matrix
    corr_matrix = data_file.corr().abs()
    #heat map
    sns.heatmap(corr_matrix)
    # Select upper triangle of correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

    # Find features with correlation greater than 0.95
    to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]

    # Drop features
    data_file.drop(to_drop, axis=1, inplace=True)
    data_file.to_csv("corelated_original_updated_dataset.csv", index=False)
```

FIGURE 3.4: Correlation Code Snapshot

3.2 Phase 2-Feature Selection using Genetic Algorithm

Feature selection is the process of finding the most relevant input for the models. This technique helps eliminate the redundant, unwanted and irrelevant features

which affect the overall accuracy of the model. The most advanced method for feature selection is genetic algorithms. Genetic algorithm is the method based on neural mechanisms and biological evolution for function optimization.

Genetic algorithms revolve around evolving the fittest generation. In nature, organisms' genes tend to evolve through generations to improve their ability to adapt to their surroundings. The genetic algorithm is a predictive optimization method based on natural evolution procedures. Genetic algorithms act on a population of individuals to improve approximations over time for each generation, the algorithm generates a new population by selecting individuals based on their fitness. Following that, operators borrowed from natural genetics are used to recombine these individuals which are selection, crossover and mutation.

To start the evolution process the first generation has to be initialized and rest of the generations will be evolved automatically based on what operators have been used for selection , crossover and mutation. Termination check determines whether the generation has converged to provide the fittest chromosome of that generation, if not then the process will continued until the desired solution achieved. Figure 3.5 below shows the block diagram of the feature selection process through Genetic Algorithm.

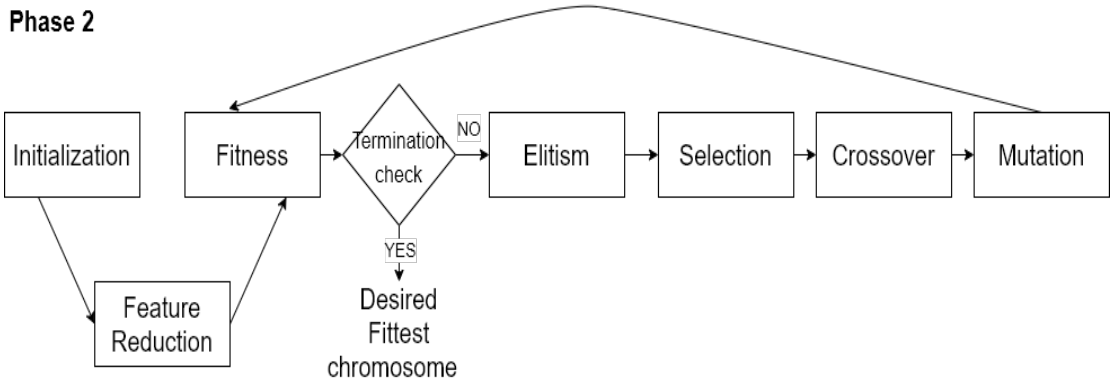

FIGURE 3.5: Phase 2 - Feature Selection Using Genetic Algorithm

3.2.1 Initialization

The procedure starts with a group of individuals known as a Population. Each individual is a potential solution to the problem you're trying to solve. Genes are

a set of factors (variables) that characterize an individual. A Chromosome is made up of a string of genes (solution).The set of genes of an individual is represented by a string in terms of an alphabet in a genetic algorithm. Binary values are generally utilized (string of 1s and 0s). The genes in a chromosome are said to be encoded. The below described are the steps that took to initialize the population and figure 3.6 shows the block diagram and flow of these steps.

Step A: 20 chromosomes were created which in fact are the 20 empty arrays using python.
Step B: Randomly generate natural numbers between 0 and 1.
Step C: Assign random value to each gene in the chromosomes the value between 1 and 0. The genes are the indexes of the array which are assigned randomly either 0 or 1 value.
Step D: Now we have 20 chromosomes with 57 genes each.

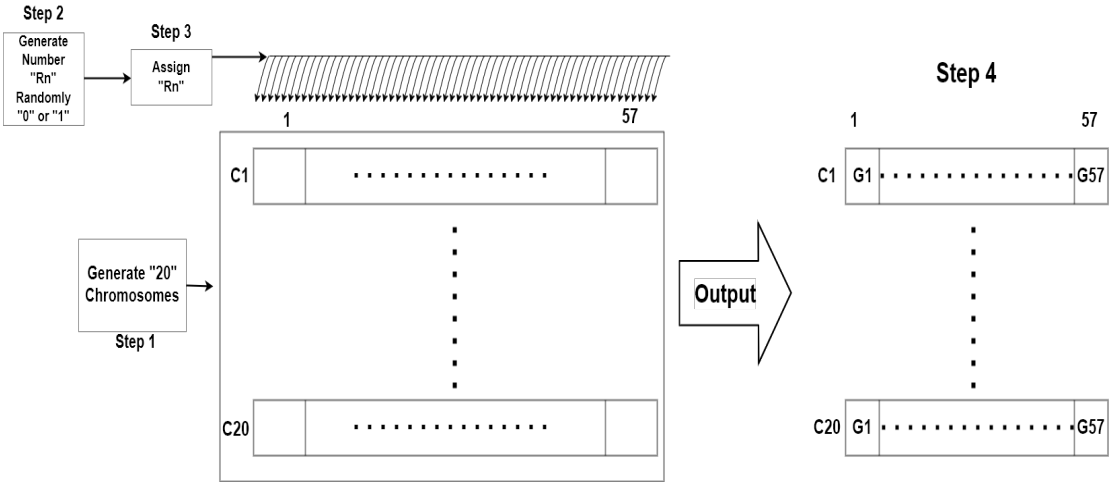

FIGURE 3.6: Initialization

3.2.2 Feature Reduction

The chromosomes that have been initialized in section 3.2.1 will then be compared with dataset . The value of each gene is compared to the column number of the dataset. So if the value is 1 then that column is taken else dropped. As a result 20 different datasets have been generated corresponding to each chromosome. The process of feature reduction is shown in figure 3.7.

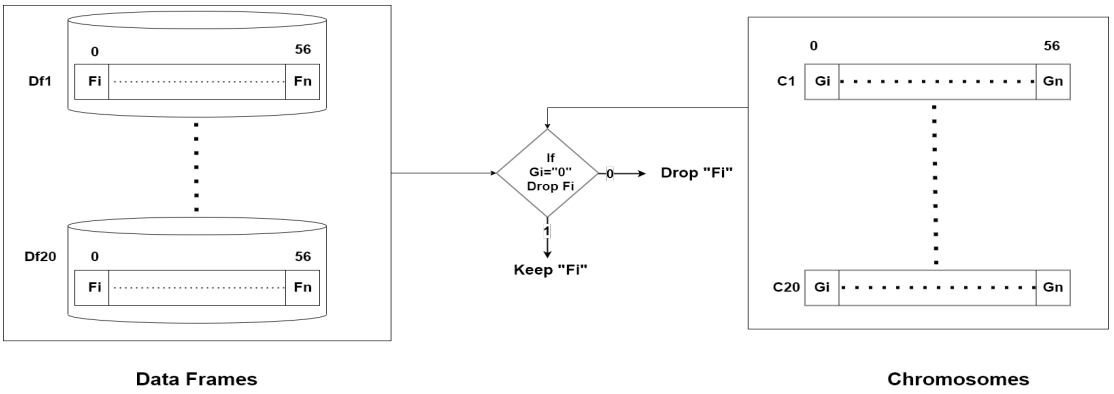

FIGURE 3.7: Feature Reduction

3.2.3 Fitness

The 20 different datasets corresponding to each chromosome was then used to evaluate and calculate the fitness of each chromosome of a particular generation. Figure 3.8 shows the process of calculating fitness. Below are the steps involved in fitness calculation.

Step A: Fed the dataset generated in section 3.2.2 to the classifier to calculate the accuracy of that chromosome.
Step B: 70 % training and 30 % testing is the train and test split used.
Step C: Used random forest classifier for evaluation with 42 random states and number of estimators 100.
Step D: Train the classifier and then test it to evaluate the accuracy.
Step E: As a result each chromosome was assigned a fitness value.

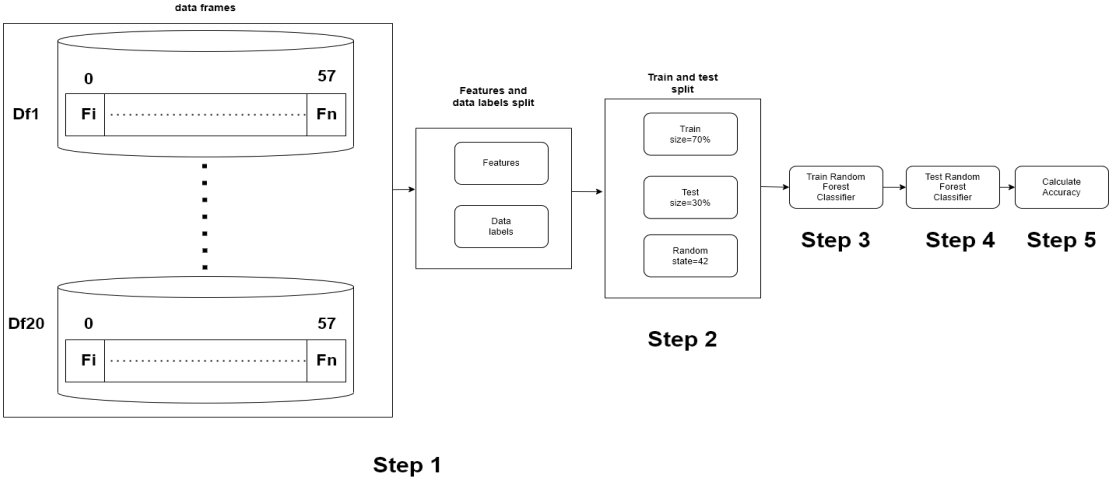

FIGURE 3.8: Fitness

3.2.3.1 Fitness Function

The below mentioned fitness function was used to calculate the fitness of each chromosome. A true positive (TP) is when the model classifies the positive class properly. A true negative (TN), on the other hand, is a result in which the model correctly classify the negative class. A false positive (FP) occurs when the model forecasts the positive class inaccurately. A false negative (FN) is an outcome in which the model forecasts the negative class inaccurately.

$$Fitness function = (TP + TN)/(TP + TN + FP + FN)$$

3.2.4 Termination Check

Termination check determines if there is a need for evolving next generation or not. Figure 3.9 shows the Termination Check process.

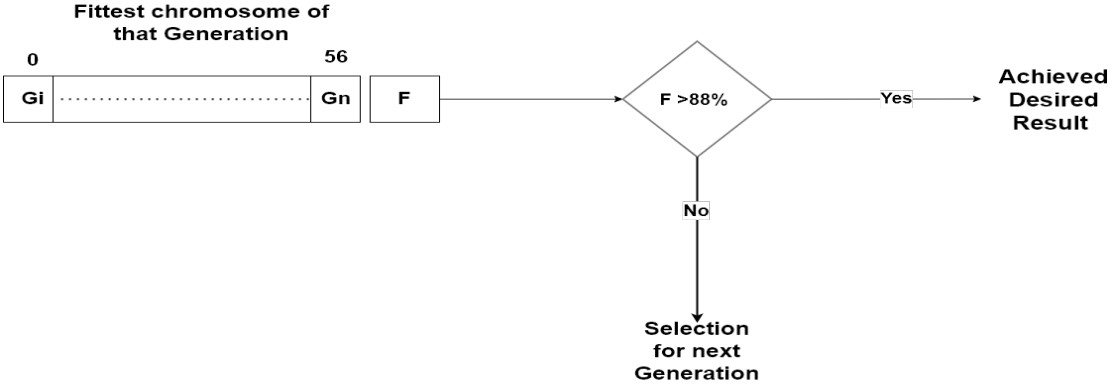

FIGURE 3.9: Termination Check

3.2.5 Elitism

Elitism is the strategy of allowing the best chromosome of current generation to carry on to the next generation unaltered. With the help of this strategy the solution quality does not degraded form one generation to another. The fittest chromosomes according to the different elitism rates as shown in Table 3.3 below will be chosen without any alteration to the next generation. Figure 3.10 shows the elitism process in detail.

TABLE 3.3: Feature Description

Elitism Rate	No. of Fittest Chromosome Chosen
0.1	2
0.2	4
0.3	6

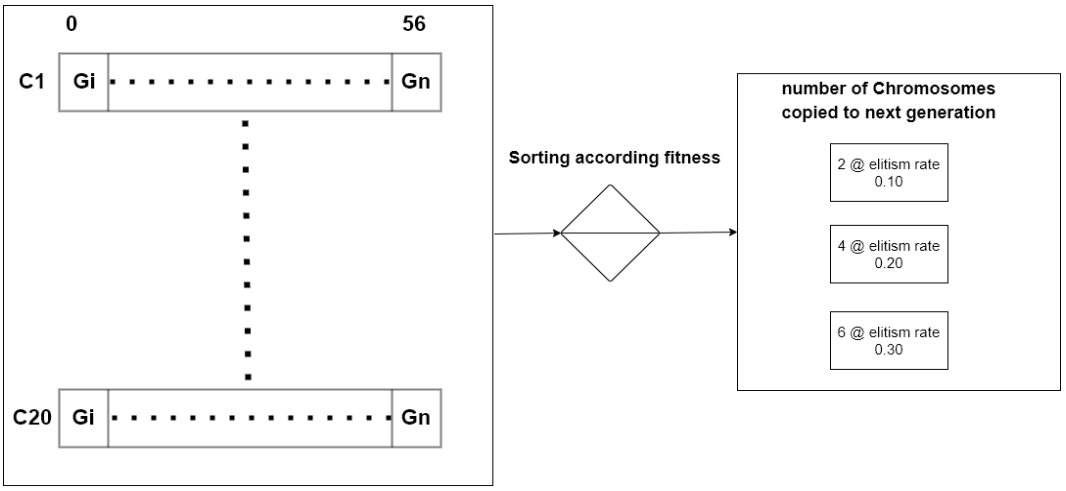

FIGURE 3.10: Elitism

The Figure 3.11 below shows the effect of elitism and mutation rate on fitness values.

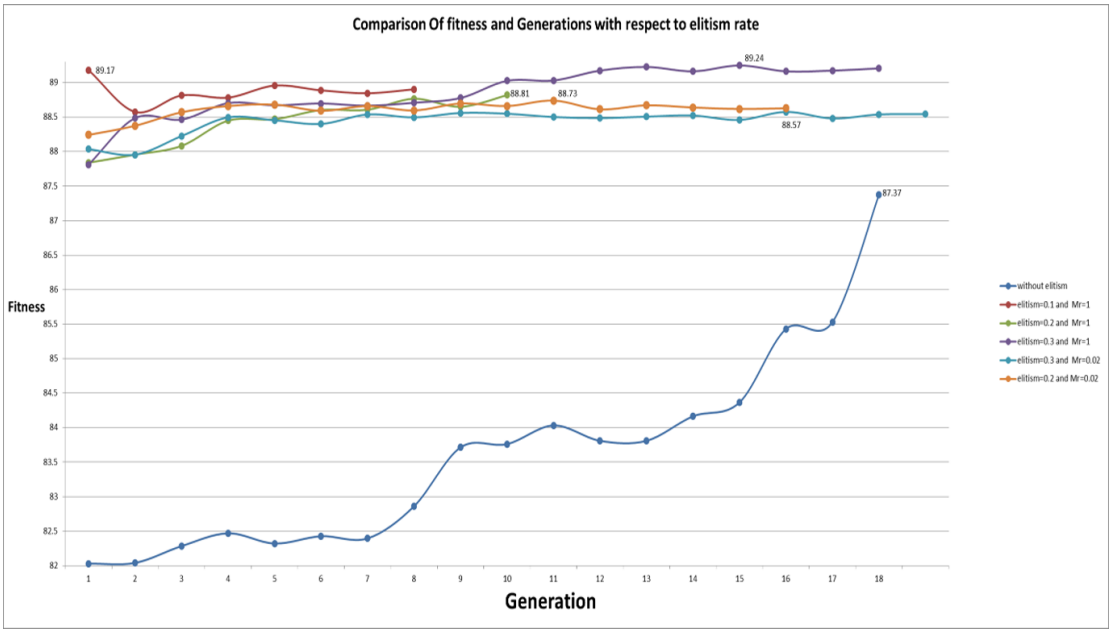

FIGURE 3.11: Comparison of Fitness and Generations with Respect to Elitism Rate

3.2.6 Selection

The goal of the selection phase is to find the fittest individuals and let them to pass their genes along to future generations. Based on their fitness scores, two pairs of people (parents) are chosen. Individuals who are physically fit have a better probability of being chosen for reproduction.

In a Genetic Algorithm, Tournament Selection is a selection approach for having fittest candidates from the particular generation. After that, the chosen candidates are passed on to the next generation. We choose k-individuals and run a competition among them in a K-way tournament selection. Only the fittest candidate is chosen and handed on to the next generation from among the selected candidates. Many such tournaments are held in this manner, and we have our final pick of candidates who will advance to the next generation. Figure 3.12 shows the selection process and below are the steps. Figure 3.13 shows the code snapshot for selection process.

Step A: Used K-Tournament selection.

Step B: Choose randomly K chromosomes as shown in Table 3.4 below.

TABLE 3.4: Values of K According to Elitism Rate

Elitism Rate	**Value of K**
0.1	8
0.2	6
0.3	4

Step C: Sort those K chromosomes according to fitness value.

Step D: The chromosome with highest fitness value among those K Chromosomes will be chosen as the parent.

Step E: We repeated the process two times in order to find out two parents.

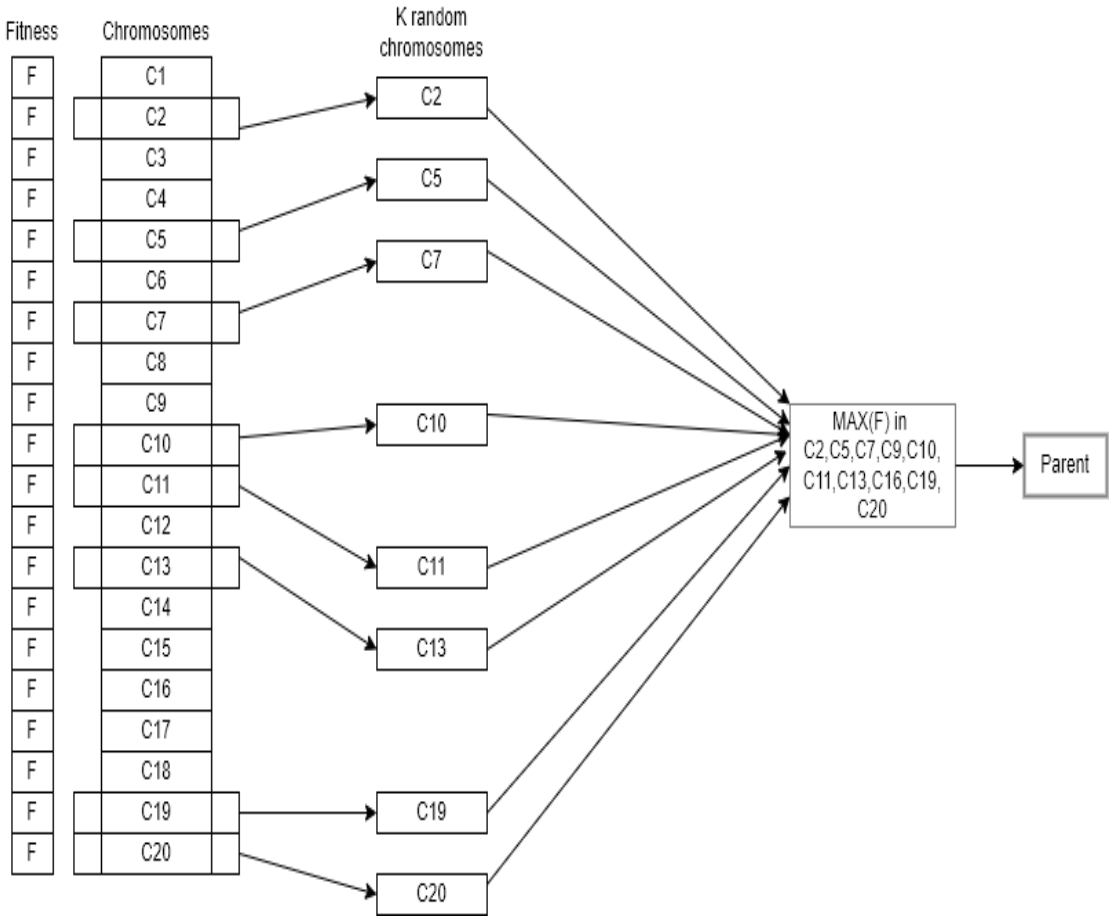

FIGURE 3.12: K Tournament Selection

```
def selection():
    parent1=[]
    parent2=[]
    for i in range(5):

        a=randint(1,10)
        csv_file = csv.reader(open('finalproject.csv', "r"), delimiter=",")
        rows = list(csv_file)
        parent1.append(rows[a])

        b=randint(11,20)
        csv_file = csv.reader(open('finalproject.csv', "r"), delimiter=",")
        rows = list(csv_file)
        parent2.append(rows[b])

    sorted_parent1 = sorted(parent1, key=operator.itemgetter(2))
    sorted_parent2 = sorted(parent2, key=operator.itemgetter(2))
    parentA=sorted_parent1[0][1]
    parentA_fitness = sorted_parent1[0][2]
    parentB=sorted_parent2[0][1]
    parentB_fitness = sorted_parent2[0][2]
    data = pd.read_csv("finalproject.csv")
    data.at[0, 'chromosome'] = parentA
    data.at[1, 'chromosome'] = parentB
    data.to_csv("finalproject.csv", index=False)
    print("parentA="+ parentA+","+"parentA_fitness="+parentA_fitness)
    print("parentB="+ parentB+","+"parentB_fitness="+parentB_fitness)
    for i in range(2, 20):
        data.at[i, 'fitness'] = 0
    data.to_csv("finalproject.csv", index=False)
```

FIGURE 3.13: Selection code snapshot

3.2.7 Crossover

Crossover also called recombination is a genetic operator used to combine the genes of two chromosomes to form new offspring. Used two point crossover to create new offspring in python figure 3.14 shows the code snapshot.

```
def crossover():
    data = pd.read_csv("finalproject.csv")
    a = list(data._get_value(0, "chromosome"))
    a = remove(a)
    b = list(data._get_value(1, "chromosome"))
    b = remove(b)
    x=[2,4,6,8,10,12,14,16,18]
    y=[3,5,7,9,11,13,15,17,19]
    for i in range(9):
        f1 = randint(1, 61)
        f2 = randint(f1, 61)
        aa=a[:f1]+b[f1:f2]+a[f2:]
        bb=b[:f1]+a[f1:f2]+b[f2:]
        data.at[x[i], 'chromosome'] = aa
        data.at[y[i], 'chromosome'] = bb
        data.to_csv("finalproject.csv", index=False)
```

FIGURE 3.14: Crossover Code Snapshot

The below mentioned is the formula for string concatenation.

$$Newparenta = a[: r1] + b[r1 : r2] + a[r2 :]$$

$$Newparentb = b[: r1] + a[r1 : r2] + b[r2 :]$$

Figure 3.15 shows the crossover process.

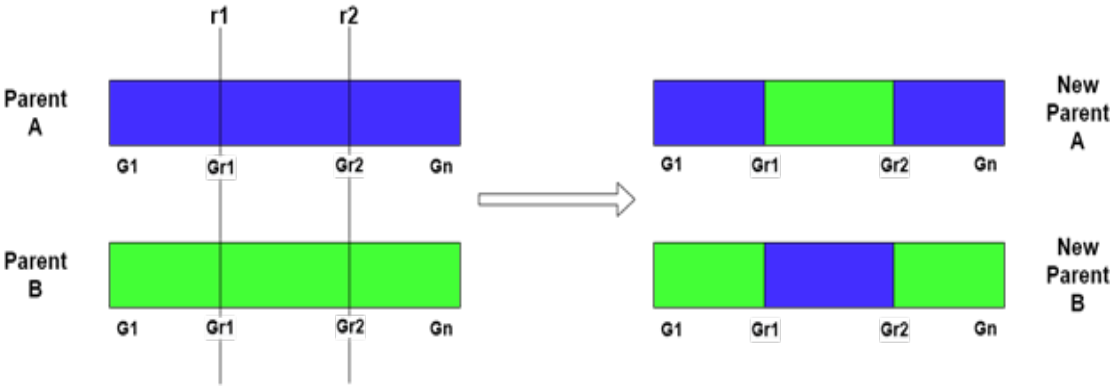

FIGURE 3.15: Crossover

3.2.8 Mutation

In simple terms, a mutation is a minor random change in the chromosome that results in a new solution. It's used to keep and introduce genetic variation into the population. The part of the GA linked to the "exploration" of the search space is mutation. The swap mutation operator has been used in this thesis. Swap mutation involves picking two locations on the chromosome at random and swapping their values. Permutation-based encodings are prone to this. Figure 3.16 shows the process we used to apply mutation and figure 3.17 shows the code snapshot which shows implementation of mutation in python.

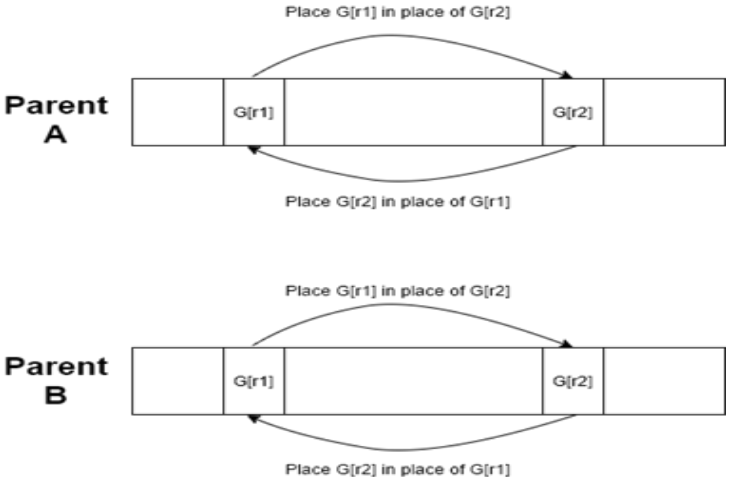

FIGURE 3.16: Mutation

```
def mutation():
    data = pd.read_csv("finalproject.csv")
    for i in range(20):
        a = randint(0, 60)
        b = randint(0, 60)
        m=0
        v1 = list(data. get_value(i, "chromosome"))
        #removing unwanted characters from list
        v1=remove(v1)

        m = v1[a]
        v1[a] = v1[b]
        v1[b] = m
        data.at[i, 'chromosome'] = v1
        data.to_csv("finalproject.csv", index=False)
```

FIGURE 3.17: Mutation Code Snapshot

Six different experiments were performed to see the effect of mutation rate and elitism rate on fitness. Figure 3.18 below shows Fitness variations with respect to mutation and elitism rate. The best fitness value (89.24%) was observed as a result when Mutation rate was equal to 1 and elitism rate was equal to 0.3.

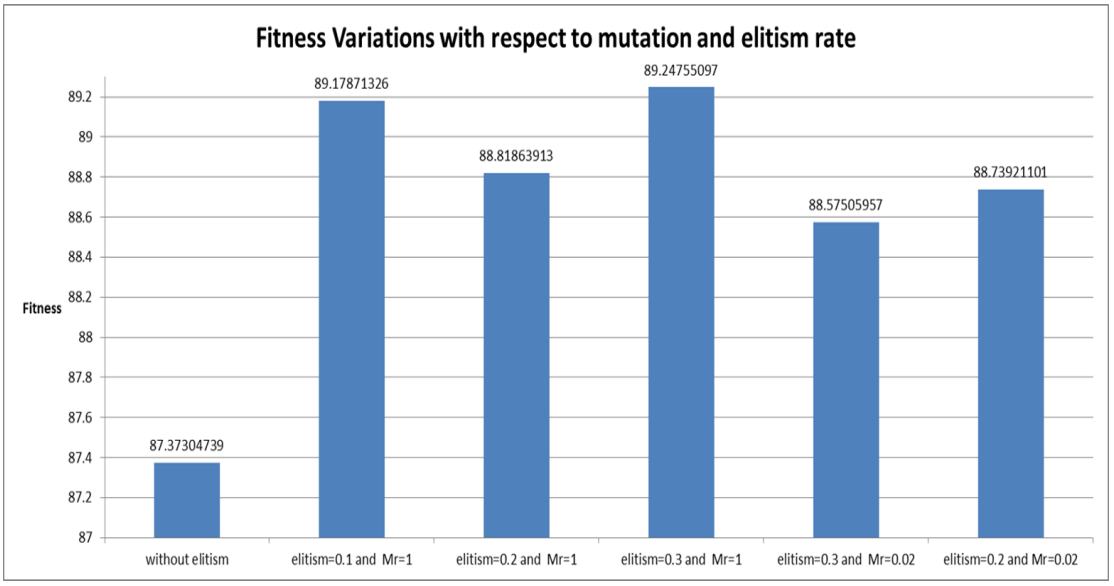

FIGURE 3.18: Mutation Code Snapshot

3.2.9 Fittest Chromosome

As a result of feature selection the table 3.5 shows the fittest chromosome.

TABLE 3.5: Fittest Chromosome

Chromosome Information	Values
Fittest Chromosome	['1', '1', '0', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '1', '1', '0', '0', '1', '1', '0']
Fitness	89.24755097
No. of features	35
Elitism Rate	0.3
Mutation Rate	1
Generation	15th

3.3 Phase 3-Evaluation

3.3.1 Final Reduced Dataset for Evaluation

The fittest chromosome that was found in section 3.2.9 was then used to generate final reduced dataset for evaluation. This dataset has 35 features. The table 3.6 below shows the names of all 30 features of the final reduced dataset.

TABLE 3.6: Names of Fittest Features

Sr. No	Feature List	Sr. No	Features List
1	Source_Port	19	Fwd URG Flags
2	Destination_Port	20	Bwd URG Flags
3	Total_Fwd_Packets	21	Min Packet Length
4	Total Length of Fwd Packets	22	Packet Length Mean
5	Fwd Packet Length Min	23	FIN Flag Count
6	Fwd Packet Length Mean	24	RST Flag Count
7	Fwd Packet Length Std	25	ACK Flag Count
8	Bwd Packet Length Max	26	URG Flag Count
9	Bwd Packet Length Mean	27	CWE Flag Count
10	Flow IAT Mean	28	Down/Up Ratio
11	Flow IAT Std	29	Fwd Avg Bytes/Bulk
12	Flow IAT Min	30	Fwd Avg Bulk Rate
13	Fwd IAT Mean	31	Bwd Avg Bytes/Bulk
14	Fwd IAT Min	32	Bwd Avg Bulk Rate
15	Bwd IAT Mean	33	Init_Win_bytes_backward
16	Fwd PSH Flags	34	act_data_pkt_fwd
17	min_seg_size_forward	35	Idle Mean
18	Active Std		

The above mentioned features were extracted using python. The fittest chromosome was fed to a for loop in which each gene value was checked, if the value of that gene was equal to "0" that feature from the original dataset was dropped and

if the value of that gene was equal to "1" that feature taken as to make part of final reduced dataset .The snapshot of code is shown in Figure 3.19 below.

```
def drop_columns_acc_to_features():
    a=[0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1]
    data_file = pd.read_csv('complete_dataset.csv')
    drop = []
    for j in range(len(a)):
        if a[j] == 0:
            drop.append(j)
        elif a[j] == 1:
            pass

    data_file = data_file.drop(data_file.columns[[drop]], axis=1)
    data_file.to_csv("feature_selected_dataset.csv", index=False)
```

FIGURE 3.19: Code Snapshot for Final Reduced Dataset for Evaluation

3.3.2 Category Wise Evaluation

This section involves compiling a final reduced dataset, which was then used to evaluate three Algorithms (KNN, DT, and RF) for classification of Android malware detection. The analysis included three categories: binary, malware category, and malware family. Three metrics (Precision, Recall, and F1 Measure) were used for evaluation with 10 fold cross validation. Figure 3.20 shows this process in detail.

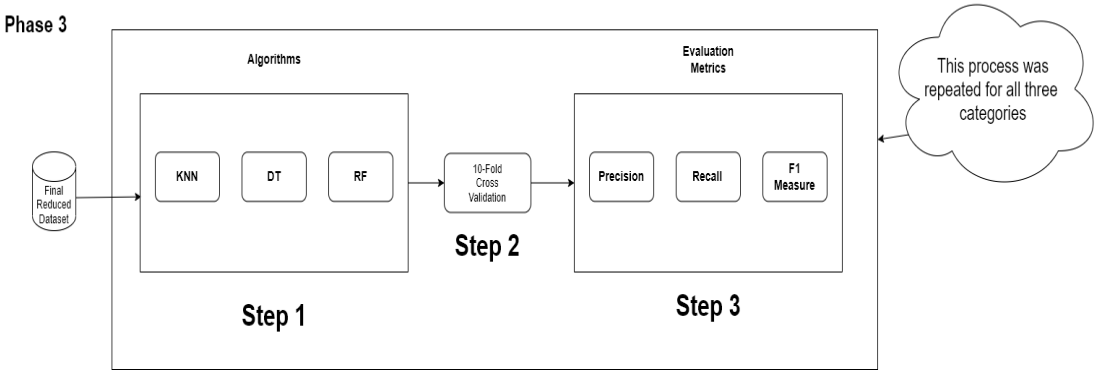

FIGURE 3.20: Phase 3 - Evaluation

Chapter 4

Results

As a result of feature selection through genetic algorithm the numbers of features were reduced from 80 total features to 35 features. The accuracy of the fittest chromosome as a result of applying genetic algorithm is 89.24%.

Table 4.1 below shows the values of fitness values of fittest parents of each generation with respect to different elitism rate and mutation rate. Figures from 4.1 to 4.6 shows 6 experimentation results for finding fittest chromosome.

TABLE 4.1: Fitness Values of Fittest Parent of each Generation

Without Elitism	**Elitism=0.1 Mr=1**	**Elitism=0.2 Mr=1**	**Elitism=0.3 Mr=1**	**Elitism=0.3 Mr=0.02**	**Elitism=0.2 Mr=0.02**
87.37304739	89.17871326	88.81863913	89.24755097	88.57505957	88.73921101
85.53031506	88.95631454	88.76568705	89.22637014	88.55917395	88.69684935
85.42970612	88.90336246	88.64919248	89.2051893	88.54858353	88.68096373
84.36536934	88.88747683	88.60683082	89.17341806	88.54328832	88.67037331
84.16415144	88.84511517	88.60153561	89.17341806	88.53799312	88.6597829
84.03177125	88.81334392	88.46915541	89.16282764	88.53799312	88.6597829
83.80937252	88.78157268	88.44797458	89.16282764	88.52210749	88.6597829
83.80937252	88.57505957	88.08260524	89.03044745	88.50622187	88.63860207
83.76171565		87.95552025	89.02515224	88.50092666	88.62801165
83.71405878		87.83902568	88.77627747	88.49563145	88.61742123
82.86153032			88.70743977	88.49563145	88.61212603
82.46968494			88.70214456	88.48504104	88.5962404
82.42732327			88.69684935	88.47974583	88.59094519
82.39555203			88.67037331	88.458565	88.56976436
82.32141912			88.6650781	88.45326979	88.37384167
82.28435266			88.48504104	88.40031771	88.24146148
82.0407731			88.46386021	88.22557585	
82.03018268			87.81254964	88.03494837	
				87.95552025	

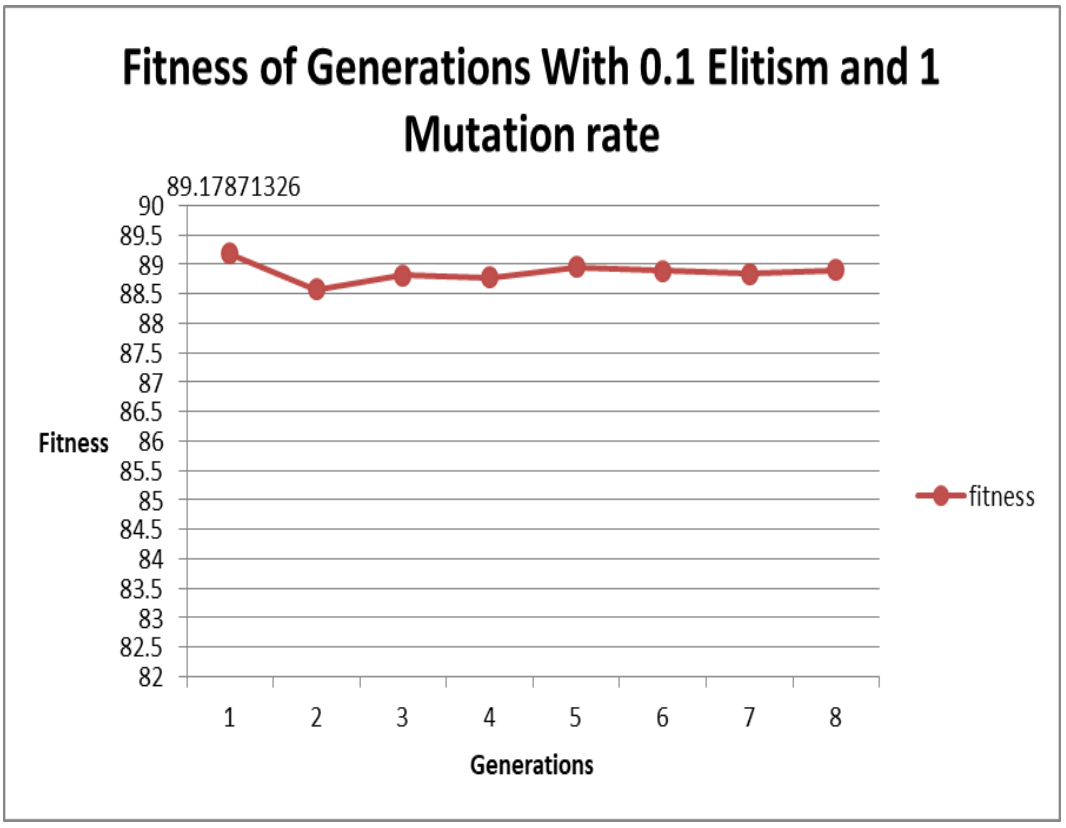

FIGURE 4.1: Fitness of Generations With 0.1 Elitism and 1 Mutation rate

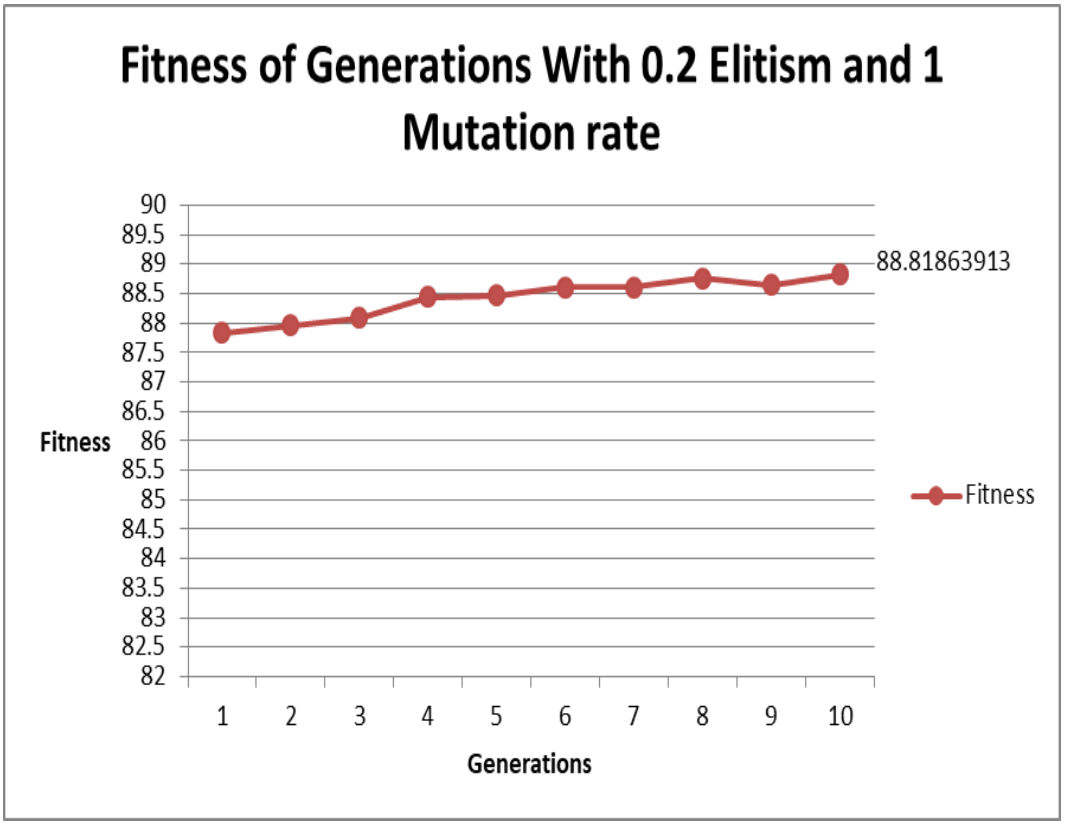

FIGURE 4.2: Fitness of Generations With 0.2 Elitism and 1 Mutation rate

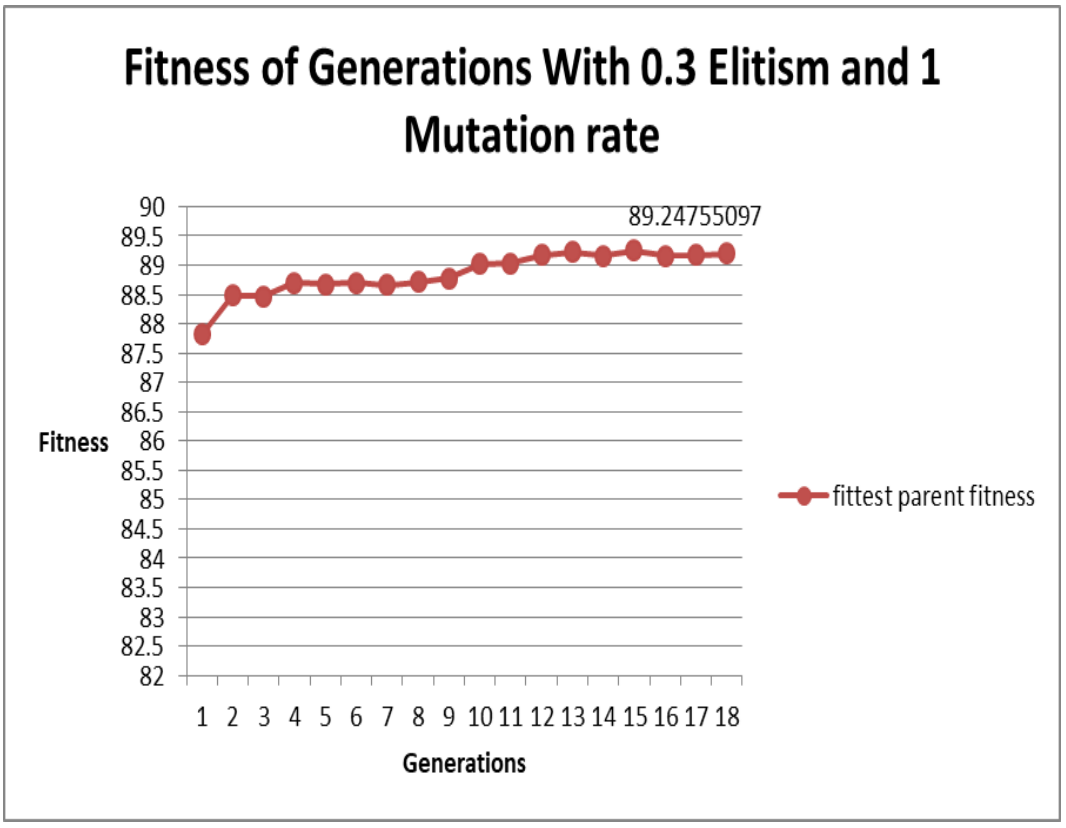

FIGURE 4.3: Fitness of Generations With 0.3 Elitism and 1 Mutation rate

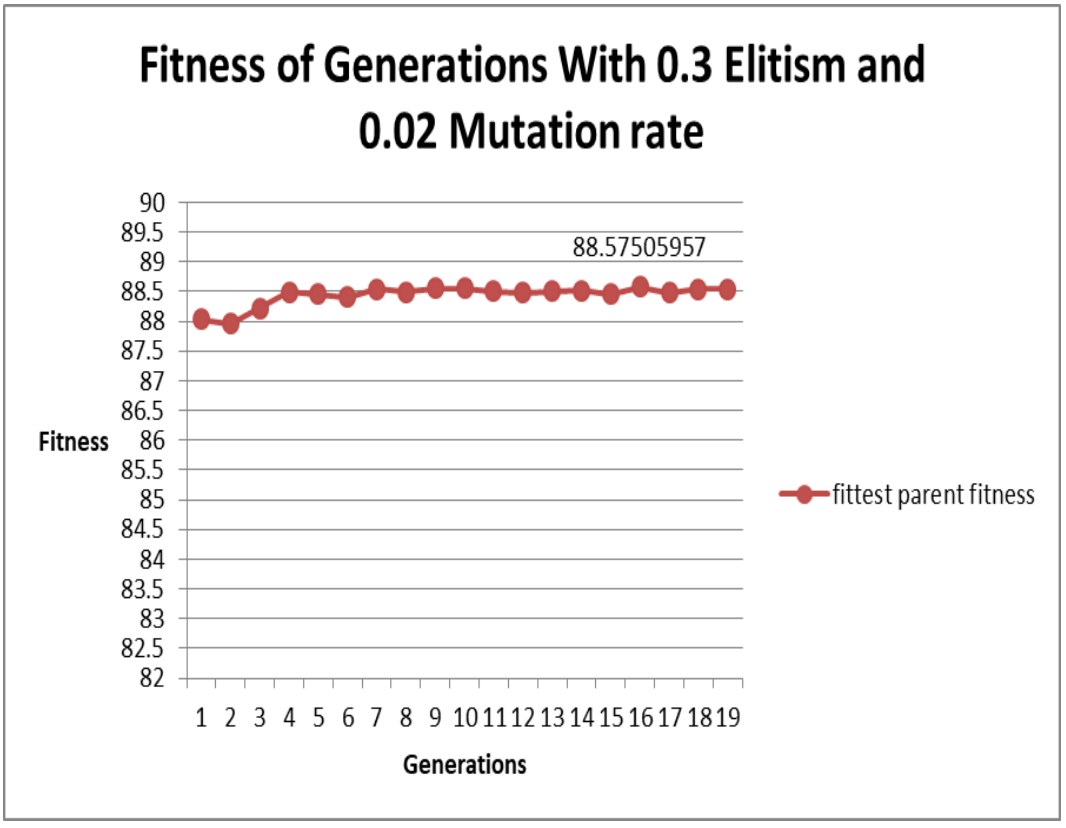

FIGURE 4.4: Fitness of Generations With 0.3 Elitism and 0.02 Mutation rate

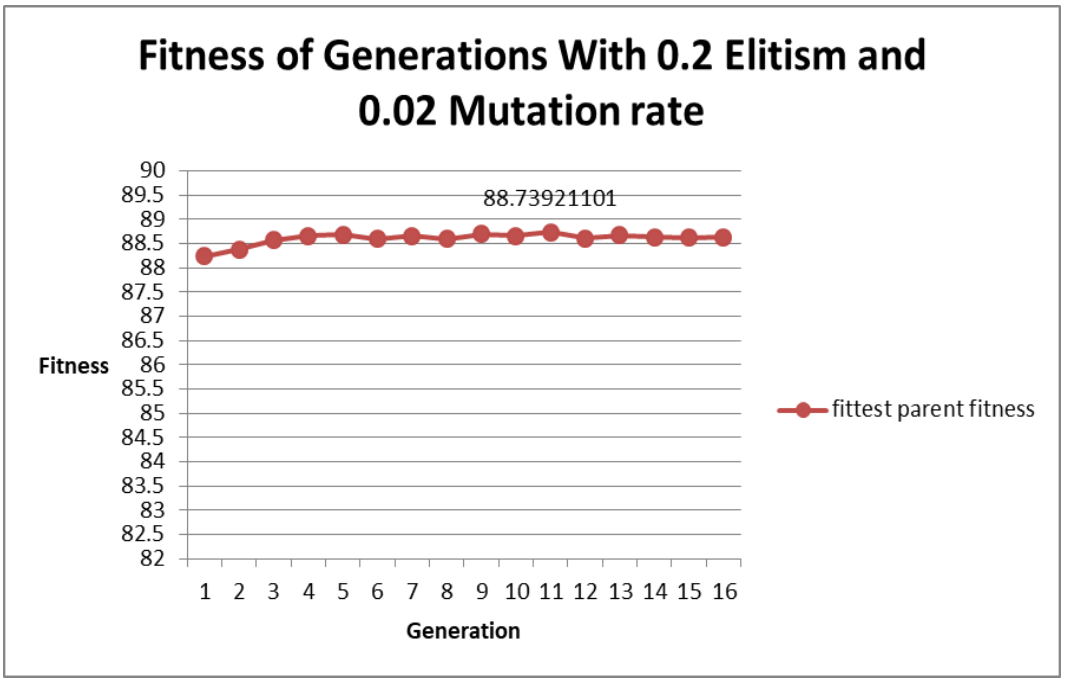

FIGURE 4.5: Fitness of Generations With 0.2 Elitism and 0.02 Mutation rate

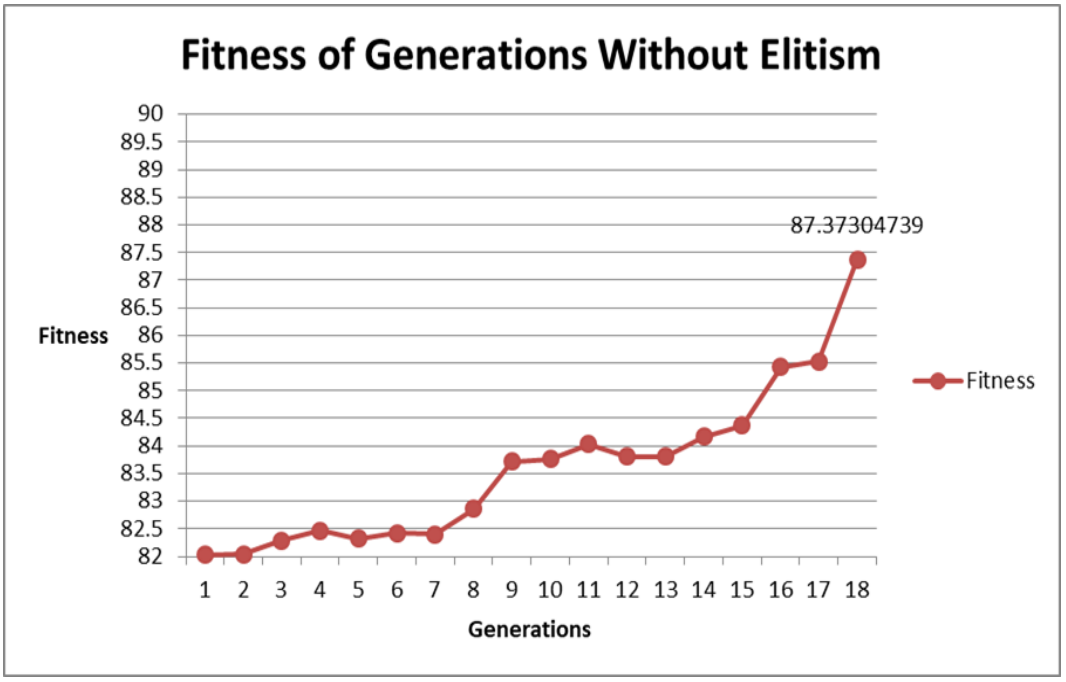

FIGURE 4.6: Fitness of Generations Without Elitism

The final reduced dataset that was compiled as explained in section 3.3.1 was used for final evaluation. The evaluation was done for three categories of the malware data. Category A corresponds to binary label which means the labels was "malware" and "benign". Category B corresponds to malware category, there

TABLE 4.2: Evaluation Matrix of Proposed Solution

Decision Tree			**KNN**			**Random Forest**		
			Category A					
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
89.6	91.7	90.7	85.5	88.8	87.1	93.3	90	91.6
			Category B					
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
52.8	53.3	53	53.2	53.1	53.2	52	53	53
			Category C					
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
43.6	45.5	44.2	41.8	41.7	41.8	37	42	39

are four types of malware that were used for classification, those four categories are Adware, Ransomware, Scareware and SMS malware. Category C corresponds to malware families that were used for classification. The classifier that was used for classification was random forest , Decision tree and K-Nearest neighbors and evaluation metrics are Precision , Recall and F1 measure. Table **??** below shows evaluation metrics results.

4.1 Result Comparison

The comparison of results with base paper was performed. Three classifiers namely Decision tree, K nearest neighbors and random forest were evaluated for calculating precision and recall for Category A,B and C and compared with base paper. The Table 4.3 below shows the comparison of our own evaluation metrics results with base paper.

TABLE 4.3: Comparisons of Results With Base paper

Categories	**Classifiers**	**Evaluation Matrix**		
Category A	**Decision Tree**	Precision	89.6	85.1
		Recall	91.7	88
	KNN	Precision	85.5	85.4
		Recall	88.8	88.1
	Random Forest	Precision	93.3	85.8
		Recall	90	88.3
Category B	**Decision Tree**	Precision	52.8	47.8

Categories	Classifiers	Evaluation Matrix		
Category C	KNN	Recall	53.3	45.9
		Precision	53.2	49.5
	Random Forest	Recall	53.1	48
		Precision	52	49.9
	Decision Tree	Recall	53	48.5
		Precision	43.6	26.66
	KNN	Recall	45.5	20.06
		Precision	41.8	27.24
	Random Forest	Recall	41.7	23.74
		Precision	37	27.5
		Recall	42	25.5

The Figures 4.7,4.8 and 4.9 shows that the results achieved as a result of this thesis are improved then the base paper results.

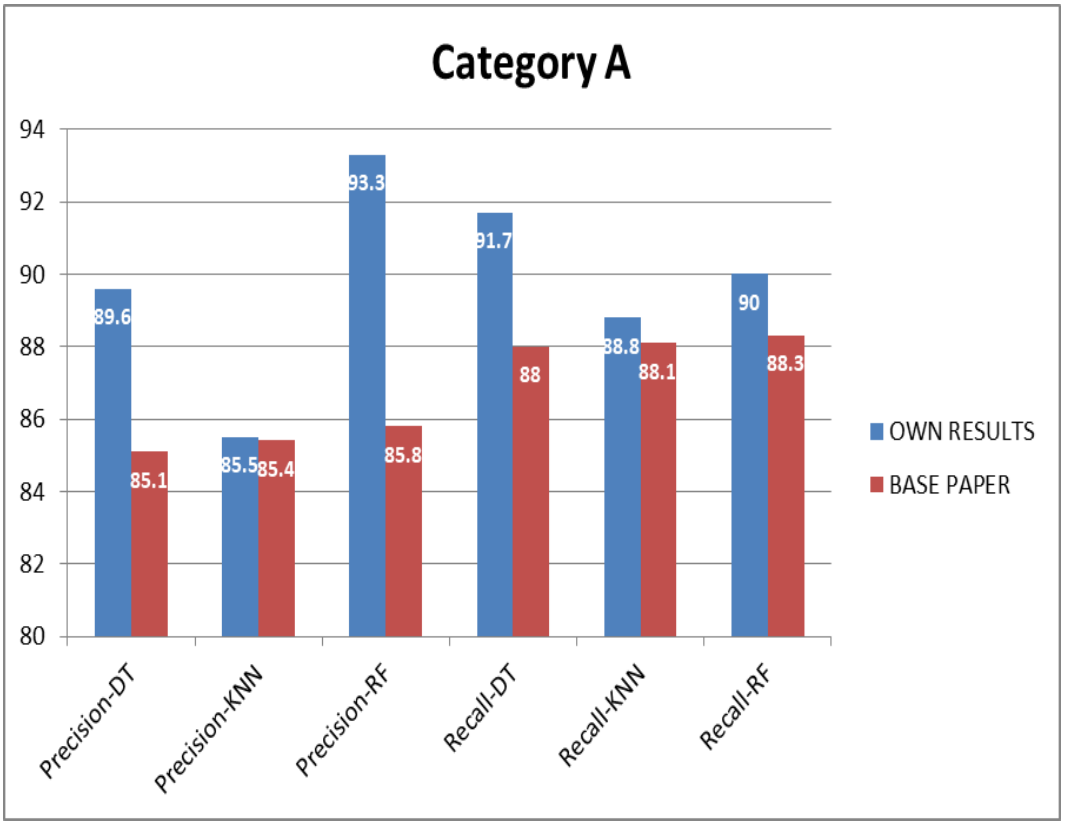

FIGURE 4.7: Comparisons of results with base paper for category A

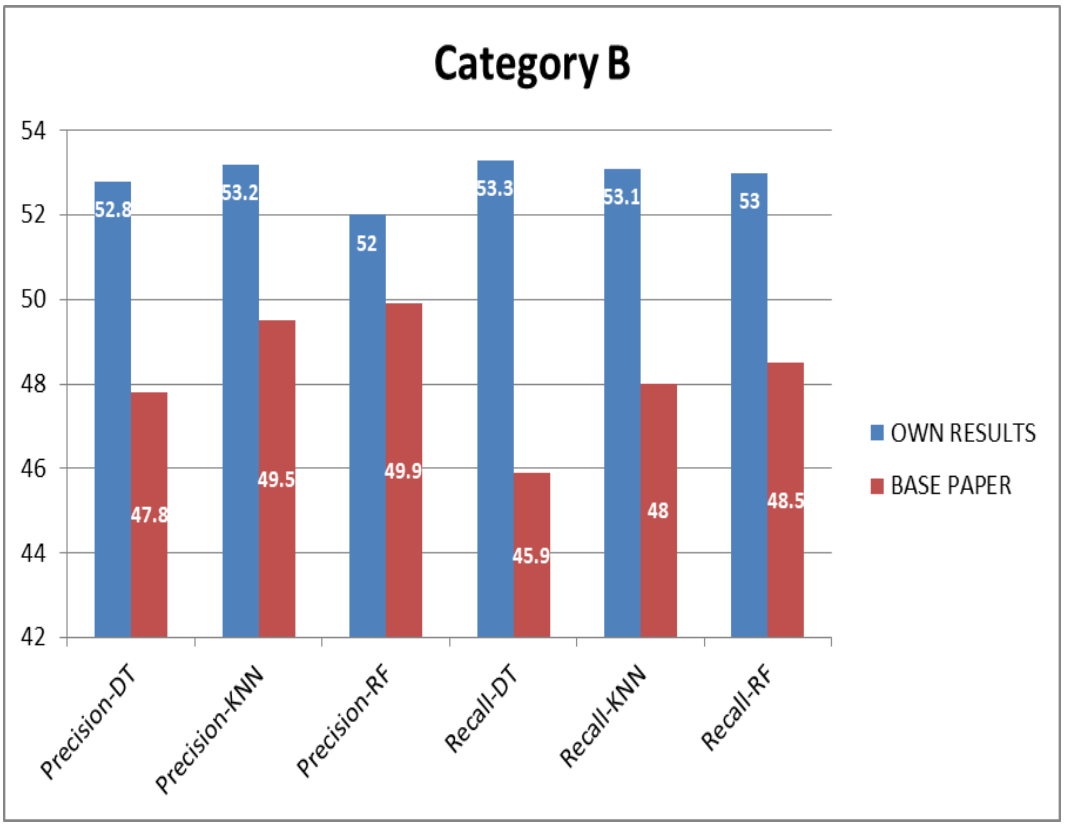

FIGURE 4.8: Comparisons of results with base paper for category B

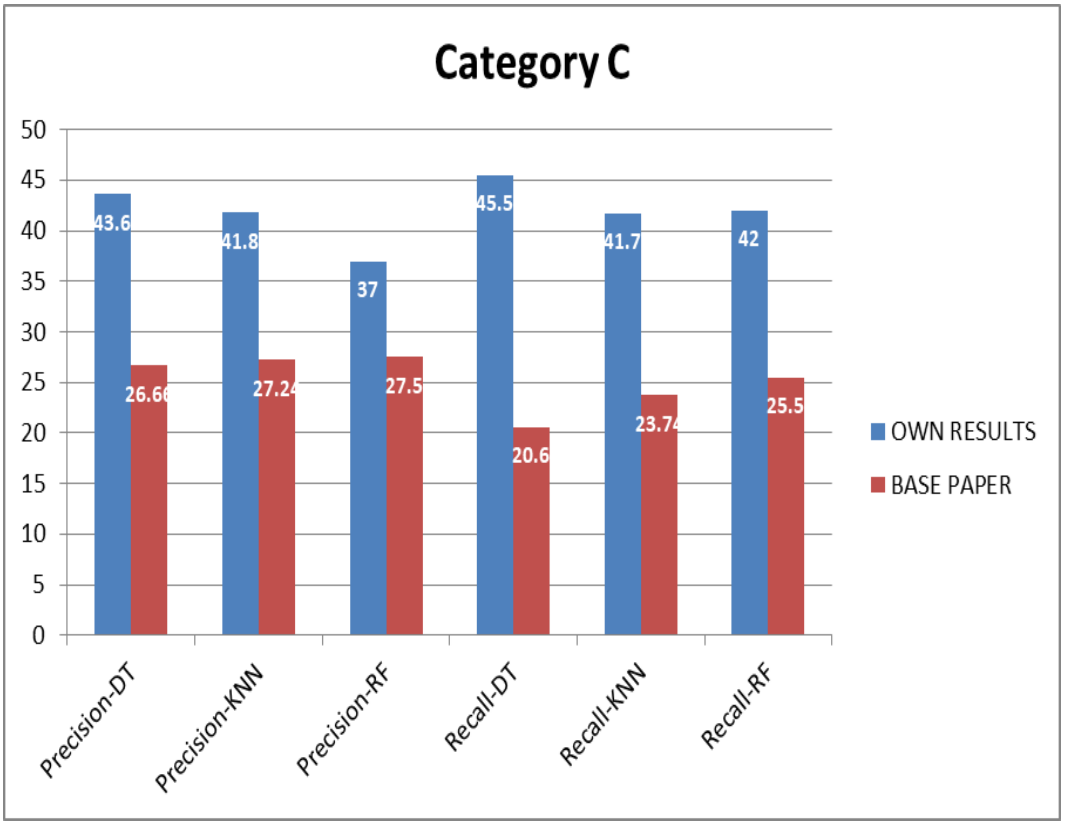

FIGURE 4.9: Comparisons of results with base paper for category C

Chapter 5

Conclusion

Feature selection plays vital role in eliminating the redundant data from the dataset. This thesis work revolves around the same goal to reduce the dimensionality of dataset to be effective for achieving high accuracy. Total number of feature set of original dataset were 80 which were first brought down to 57 using correlation preprocessing method and then Genetic algorithm came into play to further reduce the feature set from 57 to 35. The Genetic Algorithm was initialized by 20 chromosomes and then different experiments were conducted to find the fittest chromosome. Fittest chromosome was achieved in 15 generation when mutation rate was 1 and elitism rate was 0.3 and fitness value was 89.24%. The classifications resulted with average precision and average recall of 89.43% and at 90.14% for category A, 52.6% and 53.1 for category B and 40.8 and 43.06 for Category C.

Bibliography

[1] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas, "Android malware detection: A survey," in *International conference on applied informatics*. Springer, 2018, pp. 255–266.

[2] 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*. IEEE, 2014, pp. 37–42.

[3] S. Kumar, A. Viinikainen, and T. Hamalainen, "Evaluation of ensemble machine learning methods in mobile threat detection," in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2017, pp. 261–268.

[4] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015.

[5] A. Feizollah, N. B. Anuar, and R. Salleh, "Evaluation of network traffic analysis using fuzzy c-means clustering algorithm in mobile malware detection," *Advanced Science Letters*, vol. 24, no. 2, pp. 929–932, 2018.

[6] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *2013 9th international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2013, pp. 1666–1671.

[7] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, "Comparative evaluation of ensemble learning and supervised learning in android malwares using network-based analysis," in *Advanced Computer and Communication Engineering Technology*. Springer, 2015, pp. 1025–1035.

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[9] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[10] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 252–261.

[11] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 371–372.

[12] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121–134, 2017.

[13] S. B. Almin and M. Chatterjee, "A novel approach to detect android malware," *Procedia Computer Science*, vol. 45, pp. 407–417, 2015.

[14] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2014, pp. 354–358.

[15] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[16] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, 2015, pp. 13–20.

[17] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 239–248.

[18] H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 479174, 2015.

[19] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE, 2015, pp. 11–20.

[20] M. Narouei, M. Ahmadi, G. Giacinto, H. Takabi, and A. Sami, "Dllminer: structural mining for malware detection," *Security and Communication Networks*, vol. 8, no. 18, pp. 3311–3322, 2015.

[21] M. Zheng, M. Sun, and J. C. Lui, "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2013, pp. 163–171.

[22] R. Sato, D. Chiba, and S. Goto, "Detecting android malware by analyzing manifest files," *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, no. 23-31, p. 17, 2013.

[23] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez Marañón, "Mama: manifest analysis for malware detection in android," *Cybernetics and Systems*, vol. 44, no. 6-7, pp. 469–488, 2013.

[24] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine learning aided android malware classification," *Computers & Electrical Engineering*, vol. 61, pp. 266–274, 2017.

[25] Y. Okazaki, I. Sato, and S. Goto, "A new intrusion detection method based on process profiling," in *Proceedings 2002 Symposium on Applications and the Internet (SAINT 2002)*. IEEE, 2002, pp. 82–90.

[26] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015.

[27] 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*. IEEE, 2013, pp. 300–305.

[28] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection: a new approach for detecting network intrusions," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 265–274.

[29] C. Ko, M. Ruschitzka, and K. Levitt, "Execution monitoring of security-critical programs in distributed systems: A specification-based approach," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*. IEEE, 1997, pp. 175–187.

[30] C. Ko, G. Fink, and K. Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring," in *Tenth Annual Computer Security Applications Conference*. IEEE, 1994, pp. 134–144.

[31] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware

datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2018, pp. 1–7.