Android Malware Detection Using Neural Networks with NEAT

A Dissertion submitted in the partial fulfilment for the award of

Degree of Master of Technology

in

Software Engineering

by

Shubham Jain

(2K15/SWE/17)

Under the Guidance of

Dr.Kapil Sharma

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

Bhawana Road, Delhi

Certificate

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

Bhawana Road, Delhi

It is certified that the work contained in this thesis entitled "**Android Malware Detection Using Neural Networks with NEAT**" by "**Shubham Jain**" is an authentic work which has been carried out under my supervision. The content embodied in this thesis has not been submitted elsewhere for the award of any degree to the best of my knowledge and belief.

Dr. Kapil Sharma

Head of Department

Department of Information Technology

Delhi Technological University

Bhawana Road, Delhi

Declaration

I hereby want to declare that the thesis entitled "**Android Malware Detection Using Neural Networks with NEAT**" which is being submitted to the **Delhi Technological University** , in the partial fulfilment of the requirements for the award of degree in **Master of Technology in Software Engineering** is an authentic work carried out by me. The material contained in the thesis has not been submitted to any institution or university for the award of any degree.

Shubham Jain

Department of Computer Engineering

Delhi Technological University

Bhawana Road, Delhi

Acknowledgement

I take this opportunity to express my deep sense of gratitude and respect towards my guide **Dr.Kapil Sharma** Department of Information Technology.

I am very much indebted for his generosity, expertise and guidance I have received from him while working on this project. Without his support and timely guidance the completion of the project would not be possible. In this respect I find myself blessed to have my guide. He have guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation.

I would like to express my gratitude to the university for providing us with the laboratories, infrastructure, testing facilities and environment which allowed us to work without any obstructions.

I would also like to thanks to the Almighty God with his blessings I had an opportunity and strength to do this wonderful project and studies, as well as to my parents who always support me and guide me in the right direct direction with their incredible experiences of life.

Shubham Jain

2K15/SWE/17

M.Tech Software Engineering

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogram **I**nterface
GPS	**G**lobal **P**ositing **S**ystem
APK	**A**ndroid **P**ackage **K**it
XML	e**X**tensible **M**arkup **L**anguage
SMO	**S**eqential **M**ining **O**ptimization
IMEI	**I**nternational **M**obile **E**quipment **I**dentity
ARM	**A**dvanced **R**ISC **M**achines
JVM	**J**ava **V**irtual **M**achine
SDK	**S**tandard **D**evelopment **K**it
GUI	**G**raphical **U**ser **I**nterface
IDE	**I**ntegrated **D**evelopmet **E**viroment
ANN	**A**rtficial **N**eural **N**etwork

Abstract

The Google's Android mobile platform is today's one of the most popular operating system of the smartphones in the market with the shipment of over 1 billion android device in the year 2016, so with the increasing popularity naturally cyber criminal has extended their vicious activities towards Android Operating system. Security researchers had reported the alarming increase in the Android malware detection in 2015. Everyday 700 new applications are released for android platform, so there is the need for a some way of an automated analysis to detect and isolate new malware instantly.

Google provides android as the linux based open source mobile operating system platform to the developer which allows them to take full advantage of operating system and to develop system level application but on the other side it is a suitable prone for some users to develop malicious application so that they can be inserted as a safe application in the Google Play Store[1] or over web for their vicious benefits.

The increasing popularity of this android platform is making it a primary target for privacy and security violations. Confidential and highly sensitive data such a text message, contacts, reminder data etc can be accessed through the application and can be leaked through maliciously crafted application. As well as hardware sensors such as GPS can also be privacy concerns by exploiting its data for tracking and monitoring of a person's location.
Android security model is based on the permission system, there are over 300 permissions that controls the various resources. Whenever a user tries to install an application the system ask the user to grant the permission of resouces needed by the application which it will be accessing.
In this work we use the permission and API calls from the android apps to be used as the features for machine learning methods such as decision tree, Support Vector Machines and Neural Networks. We learn these classifier to identify wether an application is malicious or benign. The inherent advantage of this that there no dynamical tracing of the system by execution of application rather it uses simple static analysis to find the functions involved with the application.

Secondly we have used the NEAT algorithm to develop and evolute the best topology of neural network and achieved the good detection rate of 95% with standard deviation of 0.299% with the ideal structure of neural networks which provides faster processing

during the detection. As well we have used the Androguard[2] and APK Tool [3] to extract the permission and API from the APK[4] package to use it as feature set to test wether the application is malware or benign and used the dataset from Kaggle[5] to train our classifiers.

To my Parents and little Sister

Chapter 1

Introduction

With an estimated market share over 73%,Android had become the most popular operating system for the smartphones and mobile computing devices[6].The popularity of smartphone and mobiles devices has risen significantly. Several other operating system are also available in the market today, with the Android and iOS being the most popular one[7] as shown in Figure 1.1.In order to provide the application for their operating system the companies had developed their centralised application market place like Google's Play and Apple's App Store. These market places allows the developer to develop application and upload on it and allows user to download it on their mobile platform beside this these market places analyse and check the application for the integrity that they are benign and safe.But Android platform allows also third party open installation (unsigned) of the application which are not analysed and checked by the Google's Play.
Over 50 billion of the total apps were downloaded since the first android operating system was released in the year 2008, with the increasing popularity and allowing third party insecure installation of the application cyber criminals had increased their vicious activities towards android platform. Mobile threats researchers reported an alarming increase in the Android malware from the year 2013 to 2104 and estimated that the known malicious applications of the android system is now in the range of 120,000 to 718,000[8]. In summers of 2102, the sophisticated *Euro grabber* attack shows that the mobile malware is a very attractive business as it steals about 36 billion Euro from the customers of the banks of Italy, Germany, Spain and Netherlands [9].

1.1 Motivation

Android's open source design which give user to install unsigned application, allows the users to install applications which are not on the Google's Play [1] or processed by it. With over 1.2 million apps available for the download from Google's official application market Google Play, and another 1 million of them spread from the third party app stores such as 9apps and other, According to an estimate there are over 20,500 new apps which are being released every single month. This requires app store administrators and the malware researchers to have access to the scalable and effective solution for quickly analysing new applications to identify and isolate them from malicious applications.
Google reacts to the increasing interest of the miscreants in Android platform by its program called Bouncer in Feb 2012, which is a service on the Google Play that checks for the malware and the other hazards, when an application that is submitted to the Google Play Store by the developer [10] . However, researchers has shown that Bouncer has the very low detection rate and can be bypassed easily [11]. A large number of the similar kind of research for the malware of Android has been proposed, but neither of them provide a effective and the comprehensive solution to gather the thorough understanding of the unknown and new applications. Blasing et al. limited their research to the system call analysis [12, 13], Rastogi and Spreitzenbarth track only the specific APIs invocations [14, 15], and Yan and Yin work is to use an emulator or the virtual enviroment [16].

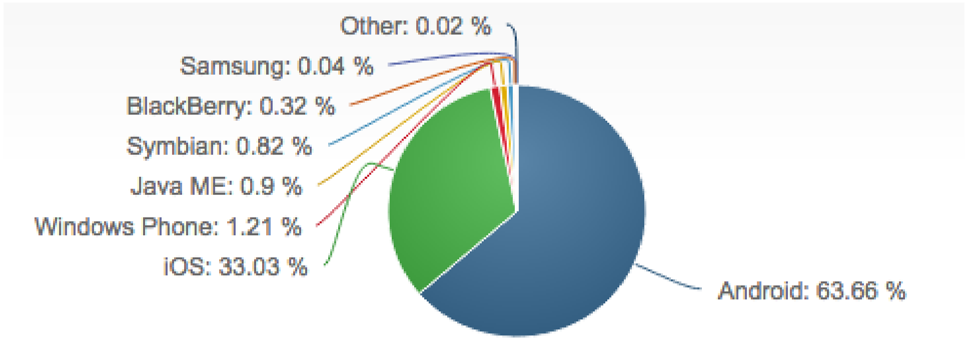


Figure 1.1: Mobile Operating System Market Share

1.2 Related Work

Samra and Yim [17] used the permission from the XML-files which is extracted from the apk files, and applied the unsupervised learning techniques such as K-means on it to group them in malicious and benign apps. This techniques achieved a fairly good

detection ratio with the average precision, recall and F-measure of 0.71, 0.71 and 0.71.
Salehi, Ghiasi and Sami [18] applied the mining techniques on the API function calls and their arguments by portable executable as features on classifiers such as Random Forests, J48 decision tree, HyperPipes, SMO and Naive Bayes and obtained a accuracy 92.1 %.
Sahs and Khan [19] extract the permission and control flow graphs from the android application files and use them as the features train one-class Support Vector Machine with different Kernels [20] to classify the application wether malicious or not.
Peiravian and Zhu [21] use the combination of the permission and API and use the machine learning methods.In their design they extract permission from the app's profile information and the API is extracted from packed apk [4] by using classes and packages. And used Support Vector Machine , Decision Tree and Bagging ensemble Machine learning classification technique.They had obtained accuracy of 96.88 %.

1.3 Problem Statement

A recent report in the field of application has shown that there are near about 2,987,387 Android applications with 12% low quality apps which are currently available on the android market [22]. The popularity of the Android operating system has lead spiking increase in the spread of the Android malware as shown in Figure 1.2 which demonstrates the Android malware growth Q1 2017 and expected growth in 2017. These malware are mainly distributed through the third party market and platforms such as 9Apps, but even the Google's Play[1] cannot promise that all of its available applications are threat free. Examples of Android malware include the Phishing Apps, Banking-Trojans, Spywares, Bot, Root Exploit, SMS Fraud's, Premium Dialer and Fake Installer. Download Trojans are the applications which download their malicious code after the installation of it, which means that these application will not be detected by Googles technology during publication or uploaded by the developer on Android market Google Play [1].
Most of the malware detection methods are based on the traditional signature based approaches in which they use a database of malware signature definitions, and compare application against this databases of the known malware signatures by extracting the sequence of bytes of code from the application. The major disadvantage of this detection method is that user is protect from those malware whose recent signature has been uploaded to the malware signature database, not from the new malware, zero day attack and not even from application which encode and change it code . A previous study of the malware patterns has concludes that the "Signature-based approaches never keep up

with the rate at which malware is created and evolved" [23] . In this thesis, our goal is to find out solution that can process an application, extract features and try to predict whether the application is Malware or Benign. Focus of our research is exploration of neural network in the domain of malware detection.

" Extract features from Android applications and use them to train Neural Networks and its variants for malware detection problem and evaluate its performance against other machine learning algorithms. "

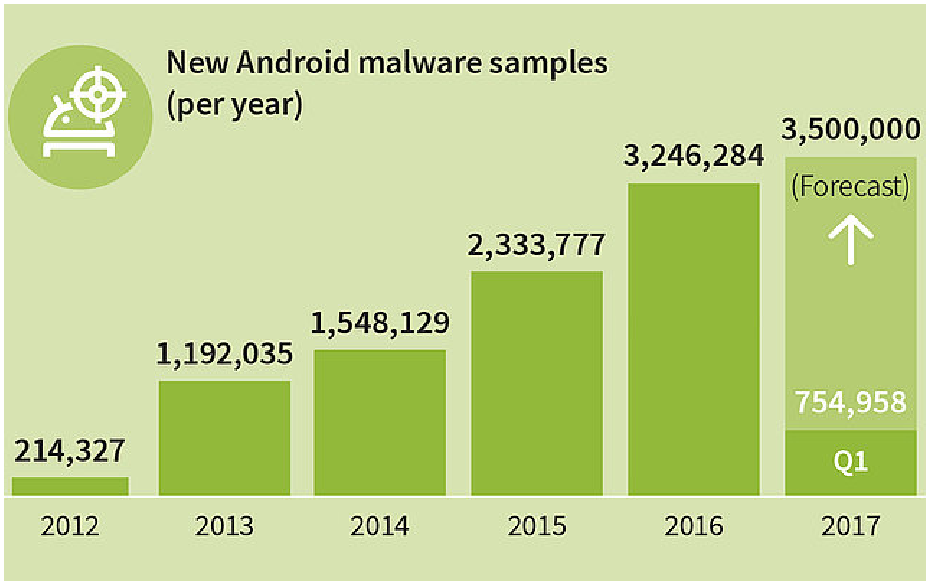


FIGURE 1.2: Android Malware Expected Growth

1.4 Scope of Work

Malware analysis is the challenging problem in the field of Computer Science. A. Moser et al. state that it is a NP-hard problem [24]. There are two major techniques for analysis namely **Static Analysis** and **Dynamic Analysis**. We have used the static analysis technique and merged it with the machine learning techniques to make our detection model more accurate and reliable by learning from the previous examples.
We extract the features such as the permission asked by the application and API calls by the application, use them for the learning algorithms. We had used the Decision tree, Support Vector Machines, Artificial Neural Networks [25] with NEAT [26] and Backpropogation training methods to train our model. For the implementation we

had used the Python with Scikit-learn module [27]. Hence our scope of works can be summarised as

- Static analysis of Android application to extract features such as permission and API using Androguard [2].
- Train the Artificial Neural Networks with NEAT and Backpropogartion algorithm.
- Compared their result with other machine learning techniques such as Decision tree, Support Vector Machines and evaluated their performance.

1.5 Organization of Thesis

Rest of our work can be summarised as below :-
Chapter 2 discuss about the android architecture and types of malware with their characteristics.
Chapter 3 discuss application of the machine learning techniques used in the android malware detection.
Chapter 4 explains the new variants artificial neural networks i.e NEAT and its working and application.
Chapter 5 explains about the implementation the NEAT, comparison, experiments and results.
Chapter 6 is about conclusion and paves way for future work.

Chapter 2

Literature Review

This chapter will give a short description of the fundamental concepts and the terminology related to the Android Operating System Architecture, Android application structure, Linux system calls, the components of the Android applications and the building blocks framework such as *activities, the services, receivers and the intents* of the Android applications. Finally about the malware what is it, its various types and how it take advantage of the android platform.

2.1 Android System Architecture

The software stack of the Android platform is shown in Figure 2.1. The violet items are the libraries and components which are native code(C/C++), green items in the Java language components which are executed and interpreted by the *Dalvik Virtual Machine* [28].The last red bottom layer represent the components of Linux kernel and execute in the kernel space of the Android operating system. In the following subsection, we will discuss about the different abstraction layer of the android system architecture. For the more detail overview of it, we refer to the earlier studies [28].

2.1.1 Linux kernel

Android use a special version of the Linux's Kernel written by the Linus Torvolds with the special additions to cope with the various requirement of the embedded system. It include wakelocks i.e it is the mechanisms which indicate that the application want that the screen should keep on displaying and does not goes off, the Binder IPC driver, a memory management system more insistent in preserving memory, and various other features which are very important for the mobile embedded platform having low hardware resources.

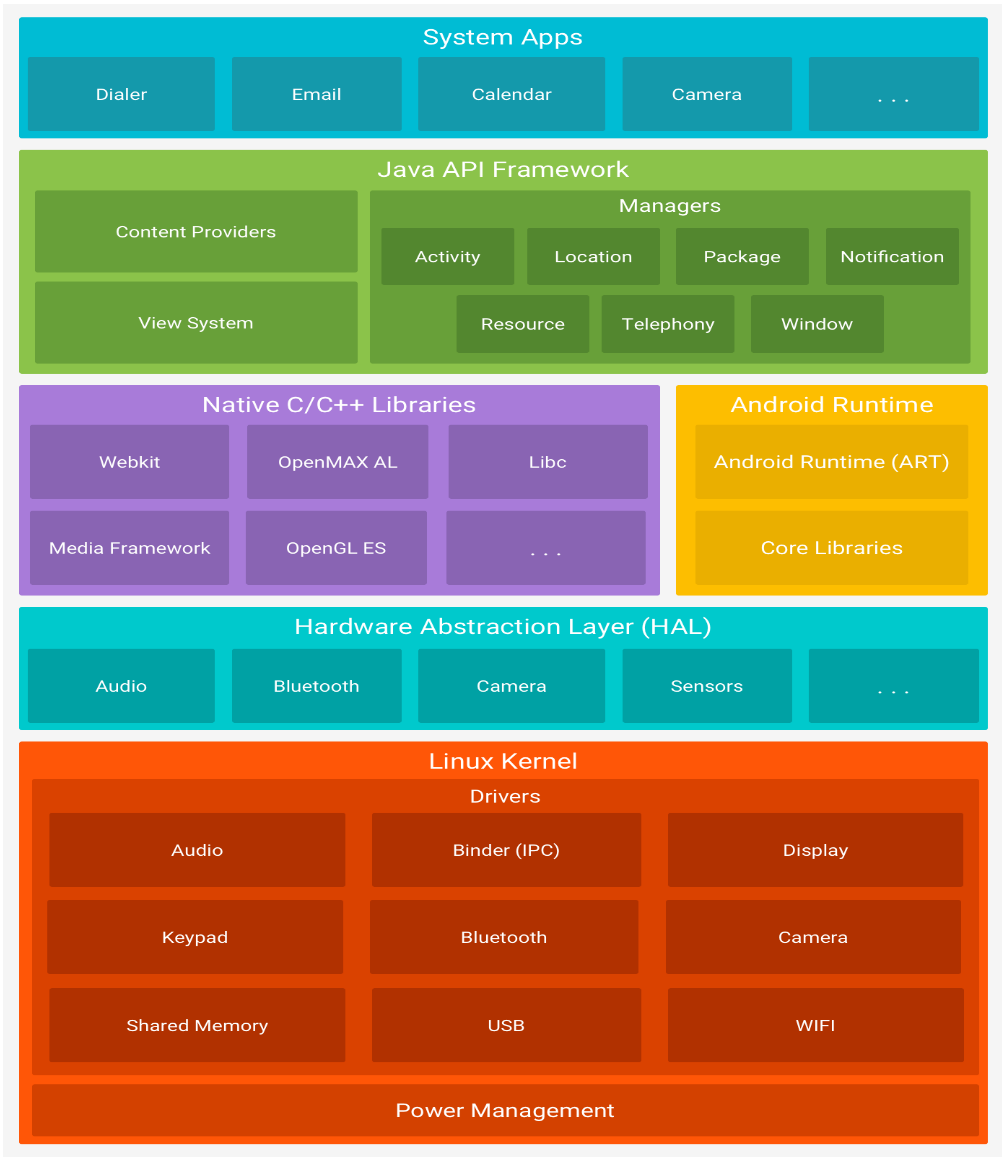


FIGURE 2.1: Android Platform Low Level System Architecture

2.1.2 Libraries

Application Framework contains set of the native libraries which are written in C/C++ and these libraries are also available to the Android Runtime by the Libraries Component. These are generally open source external libraries such as OpenSSL , WebKit and bzip2 with only very minor modifications. The essential C/C++ libraries codename as Bionic were derived from the BSD's libc and were written again for to support the

ARM hardware instruction set and implementation of the pthreads based on Linux futexes("fast userspace mutex") by Android.

2.1.3 Android runtime

The Component in the middleware of the android system is called Android Runtime which consists of the Dalvik Virtual Machine (DVM)[29] and had the set of the Core Libraries. Dalvik Virtual Machine is like the JVM which is responsible for interpretation and execution of the applications that are written in the Java programming language and discussed in detail in Section 2.2. The core libraries lay the foundation for the implementation of the general purpose API for the applications that are executed by Dalvik Virtual Machine. Android distinguishes between the two categories of the core libraries.

1. Dalvik Virtual Machine specified libraries.
2. Java or Java driven interoperability libraries.

The first one allow in modifying or processing Virtual Machine specific information and is mainly used when instructions or bytecodes are need to be loaded on the memory. The second one provides the friendly environment for the Java programmers and is derived from the Apaches Harmony. It implements the most of the popular Java language packages like java.lang and java.util.

2.1.4 Application framework

The Application Framework of the android platform provides a building blocks of high level or the abstraction to the applications in form of various packages of android. Most of these components in this layer are implemented as the applications which runs in the background as the background processes on the mobile device. Other components are responsible for the management of basic phone functionality like receiving phone calls, text messages or monitoring power usage etc. A couple of components which are more important are such as:

1. **Activity Manager** : The Activity Manager is process, which keeps the track applications which are currently active. It kills the background processes if the

device is out of memory. It also had the capability for detection of the non-responsive applications i.e when an application does not give response to the input event in 5-10 seconds (like a key pressed or screen is touch). It thus prompts an dialog for the Application Not Responding.

2. **Content Providers**: These acts as the primary building blocks for the android application. They can share the data between the various multiple applications. Address book data, for example, need to be accessed by the different applications and thus it is stored in the content provider.

3. **Telephony Manager**: The Telephony Manager provides the access to the telephony services's information of the device such as the phone's unique device identification number (IMEI), the cell current location or serial number. It is also responsible for management of the phone calls.

4. **Location Manager**: The Location Manager provide the system location services access to the android application which allow the applications to receive updates of device's geographical location periodically by the help of device's GPS sensor.

2.1.5 Applications

Applications are responsible for the interaction between device and user, and are built upon the top of the Application Framework. It is very unlikely that the average user of the android ever have to deal with the components are not in this layer. Pre-installed applications which come with bundled android system offer numerous basic tasks to the user like to perform like surfing web, reading text, making phone calls etc., but user is free to install the third party applications to use other features like (e.g., gaming, watching videos, reading news, navigation through GPS, reading books, etc.).

2.2 Dalvik Virtual Machine

The Dalvik VM [29], is like a Java Virtual Machine specially designed and modified for memory optimisation and reduce energy consumption in the embedded systems like smartphones, tablets and smartTVs. It was designed and created by the Dan Bornstein, with collaboration and contribution with Google engineers. This virtual machine is optimized to requires a low level of the memory usage and enables multiple virtual machine instances which can run simultaneously with little additional load on processor.

The Dalvik VM uses register-based architecture[30] explained in Section 2.2.3, which is faster and more efficient than the stack-based architecture[31] used in other virtual machines.

Every application in the android system runs its own process, with its very own instance of the Dalvik Virtual Machine inside the secure environment, called Sandbox [15]. The Dalvik Virtual Machine executes files which is in the Dalvik VM executable format (DEX Format) shown in Figure 2.2, which is the optimized version of the Java code file for the systems with constrained memory and slow processor speeds.

2.2.1 Hardware Constarints

The Android platform is especially designed to operate on the mobile device and thus it has to come over the challenging hardware restrictions which when compared to the regular personal computer operating systems environment, the mobile phones are small in the size and powered by the small source generally a battery. Due to this, the initial mobile devices contained a very slow CPU relatively and had very little amount of the memory left once the system is booted. Despite of these ancient and low specifications, the Android operating system does rely on the modern Operating System principles: each application is supposed to run by creating it own process and has the memory space of its own which means that each application on android should run in its own virtual machine.

It is argued that the security requirements will not be satisfied with these limited hardware constraints with the use of the existing Java virtual machines which is ported to the android system. To overcome this, Android had used the Dalvik Virtual Machine. A special instance of the Dalvik Virtual Machine is started during the boot time which later acts as the parent of all the future Virtual Machines. This Virtual Machine is called Zygote process which preloads and pre-initializes all the system classes and the core libraries. Once it is started it listens on fork() command and local socket, whenever a new application execution is requested instead of creating a new Virtual Machine from the scratch it uses the fork() command to create a virtual machine with the copied structure, thus it reduces memory footprint for running the applications and increases the speedup time by sharing memory pages which contain the preloaded system classes.

Furthermore, opposed to the Stack-based virtual machines a mechanism which can be ported to the any platform of the Dalvik Virtual Machine is Register-based virtual machine and is designed to run on the ARM processors specifically. This allows the Virtual Machines developers to add more speedup optimisations.

2.2.2 Bytecode

The bytecode is interpreted by the Dalvik Virtual Machine therefore it is called DEX bytecode or Dalvik Executable Code.Figure 2.2 shows that the DEX code is obtained by conversion of the Java language bytecode using the DX tool. One of the main difference between the Java language bytecode and DEX file format is that all of code is repacked into single output file (classes.dex), and duplicate functions, strings and code blocks are removed. Naturally, this results in the more use of the pointers within the DEX bytecode itself than in Java .class files. In general, .dex files are about 6% smaller than the compressed .jar files.

During the Android application installation, classes.dex file which is included with the application is optimised and verified by the android operating system. Verification is done in order to make sure that the program must behave normally by reducing the runtime bugs. Optimisation involves inlining of special methods, static linking and pruning empty methods.

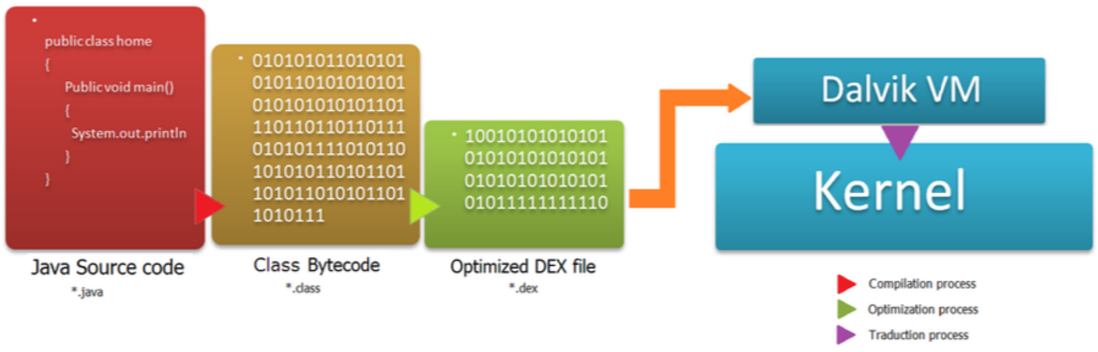


FIGURE 2.2: DEX file creation

2.2.3 Register-based Architecture

Virtual machine developers have always been in the argument of implementing a virtual machines with stack-based architecture [31] rather than register-based architecture [30]. The very simple implementation of the stack-based architecture leads the developers to prefer it use. Obviously, this simple implementation of the stack-based machine comes with performance cost. Executables for stack-based architecture are compressed than executables for register-based architecture due to more use of the POP and PUSH operations. Which leads to higher memory consumption, leading to the worse performance of the virtual machine. Register based architecture generally requires an average of 48% fewer executed virtual machine instructions than the Stack based architecture, which considerably improves performance of the device. While on the other hand, the register code used by register-based architecture is larger than stack-based architecture code. Although, the processing load generated by the Register-based architecture is still lower than that of the Stack-based architecture. Taking into account the fact that the Dalvik Virtual Machine runs smoothly on the embedded devices with constrained memory and processing power, the use of the register-based architecture is the much more appropriate choice.

2.3 Android Application

The android applications are coded in the Java programming language. Android uses Java's programming IDE and Android's Software Development Kit (SDK), such as Android Studio [32], to create an Android application installation APK file by the compilation Java code. These APK files can then be later install on Android devices by the Android Debug Bridge tool called ADB or or from the Google's Play. Figure 2.3 shows the basic structure of an APK file.

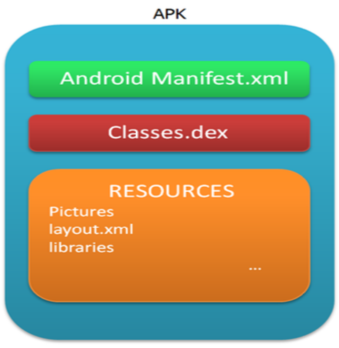


FIGURE 2.3: Dex file creation

An APK file is composed of the three main groups: AndroidManifest.xml, Classes.dex and Resources, which are packaged into a single file.

- AndroidManifest.xml: The Android manifest file contains the most important information of the Android application. It describes the application features such as the package name, application name and the permissions used by the application as well the minimum version of the Android Operating System which is required by the application to run.

- Classes.Dex: This file is output of compilation of source code in the Android Java language. It contains optimised Dex Bytecode of Android application and runs on the Dalvik Virtual Machine.
- Resources: This group contains the pictures, media, libraries and the layout files used by android application.

One of the most important elements of creating an APK file is the compilation of the Java source code. The process of generating the APK file is described in figure 2.4. The files undergoes to a series of transformations during the process of the creation of the Android APK file. The transformations comprise of the compilation process required to the generate APK files that will run on the Android devices.

The first step of process of the creation an Android application is to create an Android project, in which Java source code, Android manifest and resource files will be generated by Android Studio.

The next step is to program and configure the code to suit the purpose and to compile the project. The Java's compiler in the SDK programming environment will generate the class files from the Java's source code and the aapt will transform the AndroidManifest.xml and resource files into the adequate format so that they can be interpreted by the Dalvik Virtual Machine. The generated class files cannot be interpreted by the Dalvik Virtual Machine so in order to convert these class files into Dex files, Android SDK provides a tool called DX. This tool converts class files into the Dex format. When all the files are compiled, the final task of aapt is to compile and generate the Android APK file.

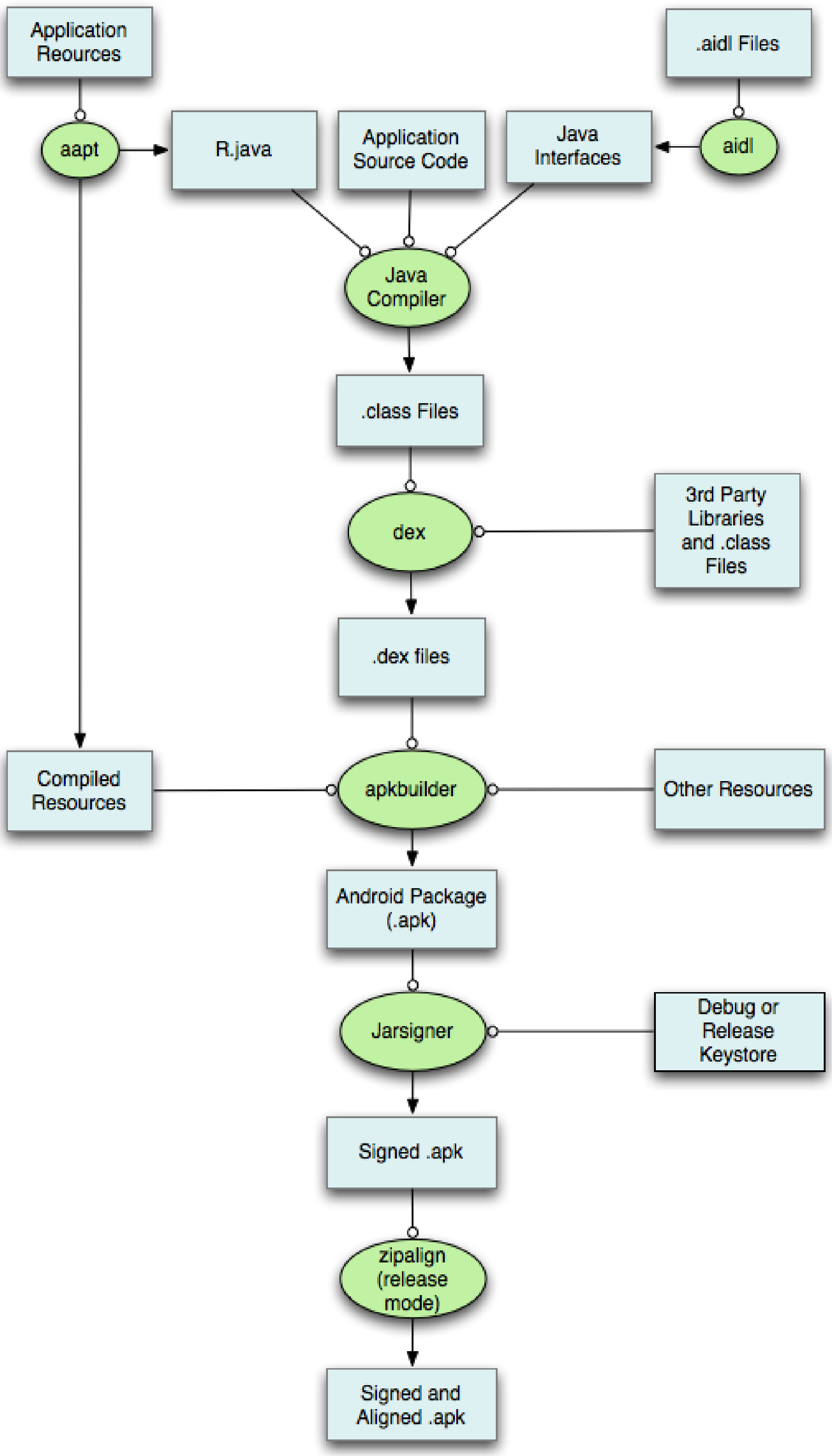


FIGURE 2.4: Android APK build process

2.3.1 Application components

We discuss the core application components thats are required to build android applications.

Activities
It represents the single screen with the particular interface for the user. Applications can have any number of the activities for the different purpose. Like a web browser, for an instance, may have one of the activity to shows the list of history items while another activity to manage the setting of the web browser. Each activity is independent of the other activity and can be started by the other applications. An e-mail application, for example, can start the web browser activity to preview of received hyperlink.

Services
Services are the components which run to perform the long running operations while running in background and do not have a interface for the user. The email application, for example, will have a socket service which is responsible for downloading emails from the server in the background simultaneously the user is interacting with the different application. Services can be started by the other components of applications like activity or by the broadcast receiver.

Content Providers
Content Providers provide access of the data between multiple applications of the android applications. They use to manage shared set containing the application data. Reminder Information, for example, is stored inside the content provider and other application can access it when required.

Broadcast Receivers
It listens to the particular broadcast announcements from the system and react on these. Generally most of the broadcast are from the system and it receives them and announce them. They don't have a user interface and act as the gateway to send information to the other components of the android system. For example if battery is low then system send the broadcast and then the broadcast receiver will announce it and the activity will capture it and alert the user by displaying a alert box on the screen.

2.3.2 Distribution

Android users can install applications either it is from google or its from other developers or via the Google Play. Google Play is an authenticated online distribution platform of application of Google from which user can download the free and paid application from the developer. Google has build an in-house fully automated antivirus system called Google Bouncer which scan the uploaded application and checker wether they may not contain malicious code .

Users can install the application from the the other sources than the Google Play, for this installation the user must enable the allow installation from the unknown source option in the settings. By this user can install the external APK files downloaded from the web as well as the third party markets. The third party markets like 9apps, Mikandi's app store, Xiaomi app store give user specialised offers.

2.4 Malware

A malware or the application which is malicious could be referred as the application which contains the malicious code which does not follow the ethnicity can compromise the device operation like the computer will not be able to boot, steal users data, bypass the control of the access and cause damage to the host. Normal application also called benign application does not perform any operation which could be dangerous to the system or to the user data and follow ethnicity. We will now discuss the various types of malware which could be found on the android operating system.

2.4.1 Spywares

The most common type of the android malware application is the spyware, the reason for its more occurrence is that the android platform is generally on mobile phone and the large amount of the person private data and activity could be found on the mobile phones. So, therefore attackers target the mobile phones with the spyware. It just simply transfer the user information to the attacker. Spyware are also commercially available to provide the user to spy on other's phones.
Spywares are installed on the victim phone either manually or by sending a link of attractive application to the target user and when spyware is active it just simply start to send the information like call records, voice mails, mails, passwords, screenshot to the third party.Example of the Commercial Spyware is CarrierIQ [33], it logs everything that is done on the mobile phone including call logs, web browsing and send those log over HTTPS protocol.

2.4.2 Graywares

Grayware is similar to the spyware but the difference is that the user itself installed that application and thought that the application legitimate. As these application provide some basic functions but secretly the send the user data like user address book, browsing history to the server and this information is used for the marketing purposes like the Filebrowser application which provide the basic file explorer service to the user but also send the collect user data secretly.

2.4.3 Fraudwares

Fraudware application gets installed on the user phone by tricking the user by some hyperlink, which user thinks some legitimate application and then they will do frauds such like doing a premium sms or the premium calls, although they informs user about the charge but the information is hidden or not mind by the user.

2.4.4 Trojans

Trojans are the malicious application which as usual in the desktop operating system will perform the dangerous activities like modify the file system, downloading the other malware, altering the system setting, making the device to act as a zombie by performing the DDos attack to the server, making multiple copies of it in the file system. The attack vector are unavailable to the attackers due to the sandboxing model therefore malicious code is merged with the some genuine app and then that application is distributed by the third party markets.

2.4.5 Root exploits

Root exploits are used to gain the root access also called sudo, super user access to the android filesystem, these are like the two side edged sword which allows user to have the full control to the devices but at the same time the same level of control is available to the application which are running on the system. Root access if available to the malware application could completely compromise the system, the malware can have the hardware access, boot access and file system access. Advance malware comes in with the root exploit to attack the user device, if there is security flaw in the system then this exploit may be successful and the malware will be running with the root privileges. An application with root access can install, modify and replace the applications. The DroidKungFu Trojan [34] is an example of it .It installs backdoor on the system which runs with root access and could send data or provide the shell access to the attacker.

2.4.6 Bots

Bots is the new trend emerged in the field of the mobile malware. It communicate with the send and receive message with the command and control centre usually the attackers servers. Bot here means that the user mobile device is acting like a Bot which is taking the command with the send and receive instruction from the server. With these commands the malware can install specific application, do the DDos[35] attack to particular server, execute some arbitrary process etc. Attackers obfuscate their code with the encoding or encryption techniques to hide the information which could help in bypassing the detection of the malware. Some examples of Bot malware are DroidDream and jSMSHider[36]. The recent version of DroidKungFu Trojan [34], can also create the bots.

2.4.7 Malware with Privilege Escalation Exploits

These are the part of the application usually the malicious application which use the the vulnerabilities or the security flaws in the System which runs the application in the sandbox or the secure environment. Every Android application use to run in the security sandbox, however, if malware is successful to get the root privileges, then it is able to do the actions which are not generally allowed to the application to perform, like the file system access. The malware DroidDream[37] which contains two exploits, first one is Exploid and second is RageAgainstTheCage[38] which is used to take the advantage of the vulnerability present in the android kernel and get the root privileges and then install an application that allows the malware to install the additional applications without the knowledge of the user. jSMSHider, was signed with a compromised key that allows the installation of the applications without intervention of the user on any mobile device which contains the firmware builds that is also signed with that compromised key.

2.4.8 Types of Malware Penetration Techniques

Repackaging is techniques which is most commonly used in malicious application installation, The developer normally downloads a legitimate application and dissemble it and add malicious code of their own in it and then re-assemble and upload to the application market. **Updating** instead of adding the code the developer include places a update component which will download the malicious code at the run time. **Downloading** app will make user lucrative and attract them to download other malicious applications.

2.4.9 Malware Datasets

The access to the known or analysed Android malware is mainly provided through the Contagio Mobile platform and Android Malware Genome Project [39]. The malgenome is the project which was the result of the work done by the Zhou and Jiang [39] and contains over 2000 Android malware samples, classified in 57 malware families and were gathered in the interval from Aug 2010 to Oct 2011. As well the Contagiodump also offers an upload the dropbox service to share mobile malware samples among the security and malware researchers and hosts 144 items presently.

Chapter 3

Machine Learning in Android Malware Detection

In this chapter we discuss about the application of the machine learning classification algorithms in the android malware detection. We discuss about the android file structure and procedure to extract features such as permission and API calls from the android applications. Then use these features to train the different models and discuss about them, such as Decision trees, Support Vector Machines and Neural Networks. Finally we present the different methods how to evaluate the performance of these models in classification.

3.1 Android Application Structure

The structure of Android application is of uttermost importance in order to do the static analysis of the android malware application, this will help in the preliminary understanding of the android application and gain the knowledge about the android application, through which we can extract the important facts from these application which could be used as the feature for the machine learning algorithm and help in development of the model for the detection of the malware. The structure of the android application is as follows:

APK: As discussed in Section 2.3 apk is the android application installation package file. Every android application is compiled and packed into the single .apk file which contains the all of the application code in the .dex format, the resources which are image and graphic and other media used by the application and the android manifest file in the

.xml format named as AndroidManifest.xml and these apk are uploaded to the Google Play or third party market or shared on the web.

Androidmanifest.xml is the one of the important file in the android application package, it defines the structure and the layout of the android application. Whenever the application is launched the android system look for the manifest file and start reading it and this file provide the roadmap to the application to make sure that the application will function properly in the android system.

It is considered that the Android system will not allow the application the access to the any resource, permission and the features which are not mentioned in the Manifest file. So the basic characteristic of the AndroidManifest file with the android mechanism of security is as follows:

3.1.1 Android Security Mechanism

The model of the Android Security is dependent on the permission based mechanism which includes about over 324 permissions that controls the access to the different resources on the android system. Therefore android application requires the certain permission to access the resource on the android system to execute. The important step in the installation of the application on a mobile device is to allow the permission request by that application.Before an application is installed the system read the permission which are request by the application and asks the user to confirm these setting show in figure 3.1 . Although these permission make sure that the application does not misuses the resource of the system but the user often have the rare knowledge to determine that the particular application is harmful or not. For example, requesting the wifi and cellular data service seem to be normal to the user but the malware can use to steal the bandwidth or other information. So therefore it is very difficult for the user to the determine by the request of the permission that the application is malicious or not.

At the level of the system, Google announce that it had implement a security mechanism called Bouncer which scan and analyse the application which are uploaded to their market Goole Play. The open design of the Android Operating System allows the installation of the application which are from the unknown sources. Nevertheless, the permission request by the application is the low level defence in the protection of the protection of the user from the harmful applications. In this way user can denies a application which ask for the address book access.

Google categorise the android permission request into the following levels:

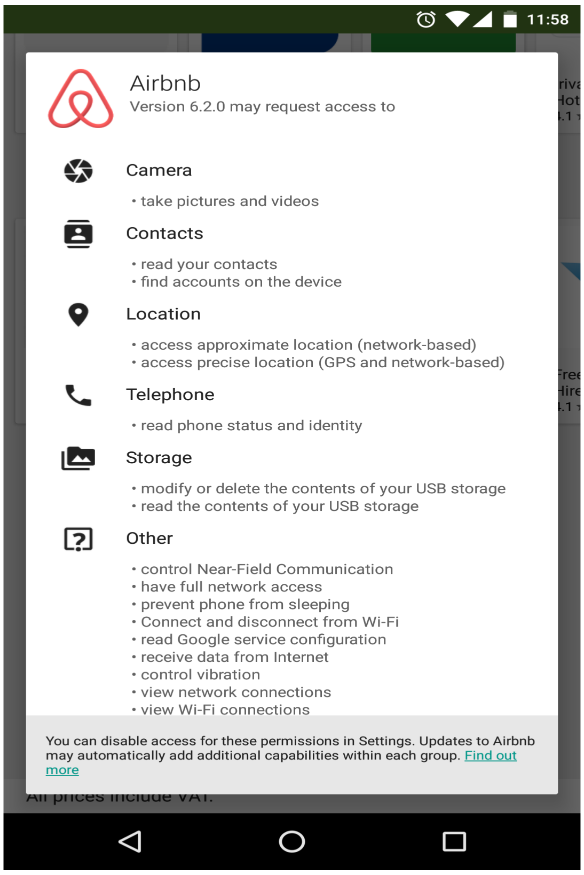
Normal Permissions: includes the lower level permissions that are the access to the API calls which are not harmful. The system automatically allows the application have the access with the user intervention. like the android.permission.SET_ALARM.

Dangerous Permissions: it is the access to the API calls which are potentially harmful which give the access to the user private data such as the android.permission. WRITE _CONTACTS or android.permission.ACCESS_MOCK_LOCATION.

Signature Permissions: is the most protected access, the system which declared permission had a certificate and the application which wants those permission should have the same certificate like android.permission.ACCESS_DRM_CERT.

System Permissions: Only system application are allowed to have these permission.

A simple idea to determine wether an application is malicious or not is to see that application is requesting the permissions which are of dangerous or system level. Although application is following an authorised method for requesting the permission to have the access to the components of the android system, there is no clear evidence by which we can say that the particular application is harmful or not. It should be considered that the permission which are shown at the time of the installation are the *requested* one not the *required* one. The requested permission are declared by the application developer itself. If the request application are not the super or equal set of required permission than application will not have access to that resource.



Airbnb
Version 6.2.0 may request access to

Camera
• take pictures and videos

Contacts
• read your contacts
• find accounts on the device

Location
• access approximate location (network-based)
• access precise location (GPS and network-based)

Telephone
• read phone status and identity

Storage
• modify or delete the contents of your USB storage
• read the contents of your USB storage

Other
• control Near-Field Communication
• have full network access
• prevent phone from sleeping
• Connect and disconnect from Wi-Fi
• read Google service configuration
• receive data from Internet
• control vibration
• view network connections
• view Wi-Fi connections

You can disable access for these permissions in Settings. Updates to Airbnb may automatically add additional capabilities within each group. Find out more

FIGURE 3.1: Android App asking for permissions

3.1.2 Android Permission Setting

AndroidManifest.xml file is present in the root directory of the APK package of the android application. This manifest .xml contains the important information about the application for the android system and as well for the android user. Android operating system process retrives the information about the application from this .xml file before it run the application's code.

The manifest file contains the permission which the android application must have granted from the user to access the protect parts of the API and interact with the other application of the android system. Structure for the android's components access is defined in this manifest file, Apktool[3] is the reverse engineering tool for the apk files and it generates the AndroidManifest .xml file, once the Manifest file is generated the android permission can be extracted from it.

Permission request are having the high relevance to the malware or the benign application, Figure 3.2 shows the comparison between the top set of the permission requested by the both malware and benign application.

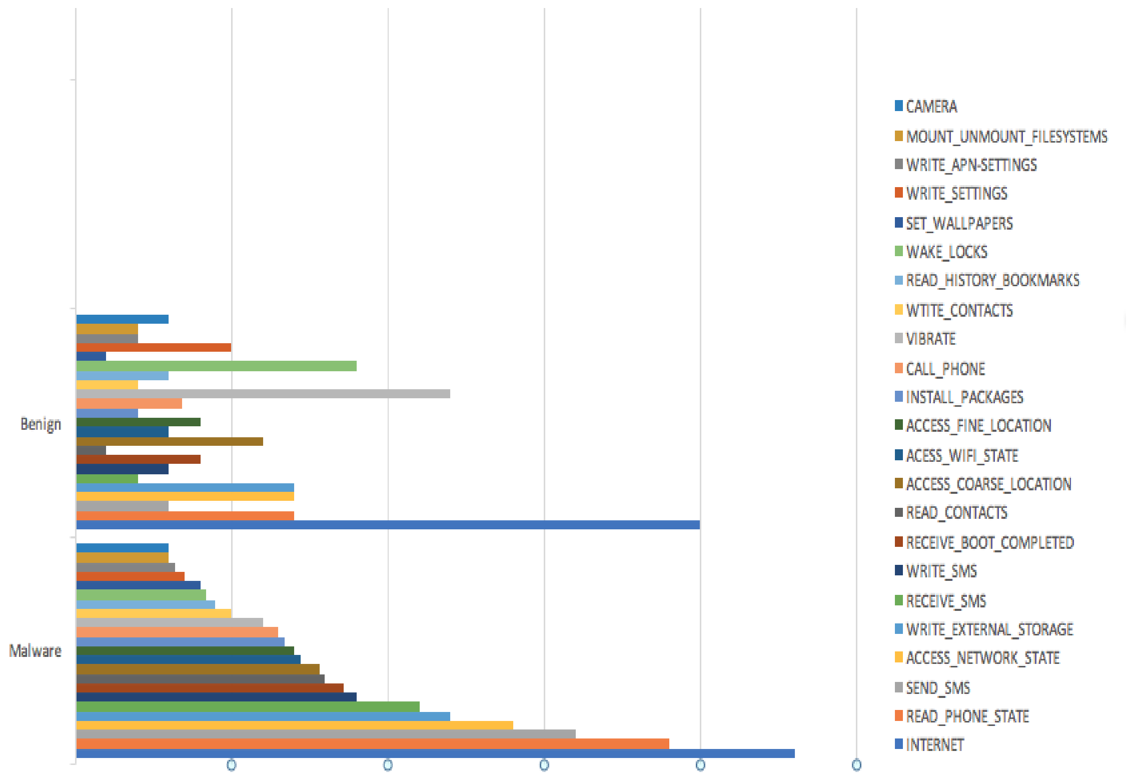


FIGURE 3.2: Most permission requested by the android applications

From the comparison, INTERNET, READ_PHONE_STATE , ACCESS_NETWORK _STATE and WRITE_EXTERNAL_STORAGE are the most frequently requested permission by the malware application, as it obvious that the malicious application will send the data from the internet or download the package from the internet, and read phone state to know the status of that the user is not currently using the system, if user is using the system that malware generally pause it malicious activity as user may feel suspicious that there may have a malware in their system. Hence for our dataset we will use each permission as the feature for the classification model.

Every application will be represented as the binary vector, namely P, where P_i=1 if the particular application has the i_{th} permission request and 0 if the permission is not request by the application or permission is present in the AndroidManifest.xml file and Table 3.1 shows different permissions and description of it .

Table 3.1: Android Permission and Description

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission. CHANGE_NETWORK_STATE	dangerous	change network connectivity	Allows an application to change the state of network connectivity.
android.permission. DISABLE_KEYGUARD	dangerous	disable key lock	Allows an application to disable the key lock and any associated password security. A legitimate example of this is the phone disabling the key lock when receiving an incoming phone call, then re-enabling the key lock when the call is finished.
android.permission. KILL_BACKGROUND_PROCESSES	normal	kill background processes	Allows an application to kill background processes of other applications, even if memory is not low.
com.android.launcher.permission. UNINSTALL_SHORTCUT	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. READ_LOGS	dangerous	read sensitive log data	Allows an application to read from the system's various log files. This allows it to discover general information about what you are doing with the phone, potentially including personal or private information.
android.permission. ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.
android.permission. INTERNET	dangerous	full Internet access	Allows an application to create network sockets.
android.intent.action. BOOT_COMPLETED	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. WAKE_LOCK	dangerous	prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.
com.android.launcher.permission. INSTALL_SHORTCUT	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission. GET_TASKS	dangerous	retrieve running applications	Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.
android.permission. DELETE_PACKAGES	SignatureOrSystem	delete applications	Allows an application to delete Android packages. Malicious applications can use this to delete important applications.
android.permission. WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete SD card contents	Allows an application to write to the SD card.
android.permission. GET_PACKAGE_SIZE	normal	measure application storage space	Allows an application to retrieve its code, data and cache sizes
android.permission. READ_EXTERNAL_STORAGE	dangerous	read SD card contents	Allows an application to read from SD Card.
android.permission. RECEIVE_BOOT_COMPLETED	normal	automatically start at boot	Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission. INSTALL_PACKAGES	SignatureOrSystem	directly install applications	Allows an application to install new or updated Android packages. Malicious applications can use this to add new applications with arbitrarily powerful permissions.
android.permission. ACCESS_MTK_MMHW	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. WRITE_SETTINGS	dangerous	modify global system settings	Allows an application to modify the system's settings data. Malicious applications can corrupt your system's configuration.
android.permission. READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
android.permission. MOUNT_UNMOUNT_FILESYSTEMS	dangerous	mount and unmount file systems	Allows the application to mount and unmount file systems for removable storage.
android.permission. VIBRATE	normal	control vibrator	Allows the application to control the vibrator.
android.permission.SYSTEM_OVERLAY _WINDOW	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. SYSTEM_ALERT_WINDOW	dangerous	display system-level alerts	Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.
android.permission. CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.ACCESS_WAKE_LOCK	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. ACCESS_DOWNLOAD_MANAGER	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. CHANGE_WIFI_STATE	dangerous	change Wi-Fi status	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
android.permission. PACKAGE_USAGE_STATS	signature	update component usage statistics	Allows the modification of collected component usage statistics. Not for use by common applications.
com.android.permission. UNINSTALL_SHORTCUT	dangerous	Unknown permission from android reference	Unknown permission from android reference
android.permission. RESTART_PACKAGES	normal	kill background processes	Allows an application to kill background processes of other applications, even if memory is not low.
android.permission. GET_ACCOUNTS	normal	discover known accounts	Allows an application to access the list of accounts known by the phone.

3.1.3 Android API calls

Android platform provides a variety of the API of the framework which allows the android application to interact with the android system and to perform the certain functions. API of the frameworks consists of the set of packages and the classes.

Every application use the large numbers of the APIs, therefore the malware application also uses a variety of the APIs of the android platform, thus these can be also merged with feature for the classification model to add more dimension to the training set. Using the MobSF [40] framework to extract the API and similarly like the permission we can represent it as the binary vector namely as A, where A_i =1 represent that the particular i^{th} API is called by the application.

The combined API and permission are used as the features for the classification model and the training is done using the dataset of the malware and benign application, we had used the 440 samples of malware and benign application in this work.

Table 3.2: Android APIs and Description

API	**Description**
android.content.Content()	used for retrieving resource data associated with an application.
android.content.Intent()	an action to be performed
android.app.Activity()	initialise create pause the activity
android.view.View()	change the view of the activity layout
android.os.Handler()	handle the document or the media
android.os.Bundle()	get the version of the bundle of the OS
android.graphics.Bitmap()	get the pixel bitmap of the graphic or image
android.content.res.Resources()	get the resouces of the application
android.graphics.Canvas()	access the canvas to draw on the screen

3.2 Extracting the features using Static Analysis from the Android Application

We had discussed that what are the different features which can be used for the classification models from the android application. Now we will discuss how we will extract these features with the help of the tools such as the Androguard [2] and APK tool [3] with the technique called static analysis. Static analysis in which we analyse the strings and Bytecode and structure of the particular application with its any kind of execution. Figure 3.3 that how the APK is processed to extract the API and Permission to be used as features.

3.2.1 Androguard

Androguard [2] is the command line interactive oriented tool written in the python language which perform the static analysis of the android applications. It can do the disassembly of the apps and access their various components such as permissions, receivers, activities, classnames. It also contains tools such DEX to Jar, smali, API extractor. Some of the commands of the Androguard are

```
a,d,dx = AnalyzeAPK(”malware.apk”,decompiler=”dad”)

a.get_permissions()

a.get_packages()

a.get_appname()

a.get_mainactivity()

a.get_activities()

a.get_classes_names()

d.get_class.source()

a.get_recievers()
```

3.2.2 APK Tool

APK Tool is tool for reverse engineering application of the android platform which is written in Java programming language. It uses the Smali [41] which uses to convert the Dalvik Virtual Machine [29] DEX code into jar code which can be used for the analysis and the reverse engineering the application. It produce the extracted folder of the .apk file and AndroidManifest .xml file and can also reverse the class and function in the jar format which can be used to modifies the application and add the new code before repackaging it. It can **Decompile** and **Recompile** the application using the Smali to convert the DEX to jar and then again jar to the DEX.

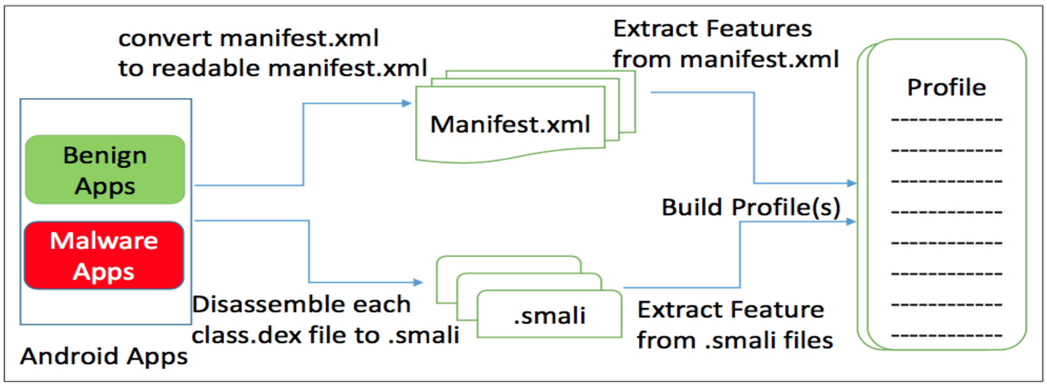


Figure 3.3: Steps in Feature Extraction from Android APK

3.3 Machine learning Techniques Used

We now had the features and data for the classification models extracted with the static analysis using Androguard [2] or APK Tool [3]. We will discuss the classification machine learning technique which we had used for the classification of the android malware and benign application and then compare these with the Neural Network NEAT variant.

3.3.1 Decision Trees

Classification can be done asking the series of the question about the attributes of the features dataset. Suppose we do the classification of the mammals and the non-mammals. To determine wether the specie is mammal or non mammal the series of question are asked like *is the specie warm blooded or cold blooded?*, If it is cold blooded then definitely the specie is not mammal. *Do the female of the specie give birth ?*, Those species who give birth to the animals are definitely mammals. Each time the question is asked and then follow up is made till one reaches to the class label of record. When all these question are label to a directed tree then it becomes a Decision tree[42].

Decision tree represent those question and does the classification by traversing through the tree where every branch represent the question related to the attribute of the data set. Decision tree is the directed tree which contains the root, internal nodes and leaf nodes. All nodes other than the leaf nodes are called decision nodes. The internal node splits into the two or more subspaces with respect to the certain function applied on the input attributes values figure 3.4 show a decision tree. Various terminologies related to the decision tree are :

Root node its represent the entire population of the dataset and further divided into two or more sets.

Decision node nodes divide into sub tree, and contains the criteria for selecting the follow branch.

Leaf node Represent the class label.

Splitting is the process of dividing a node into sub-nodes.

Pruning removal of the sub nodes from the decision tree.

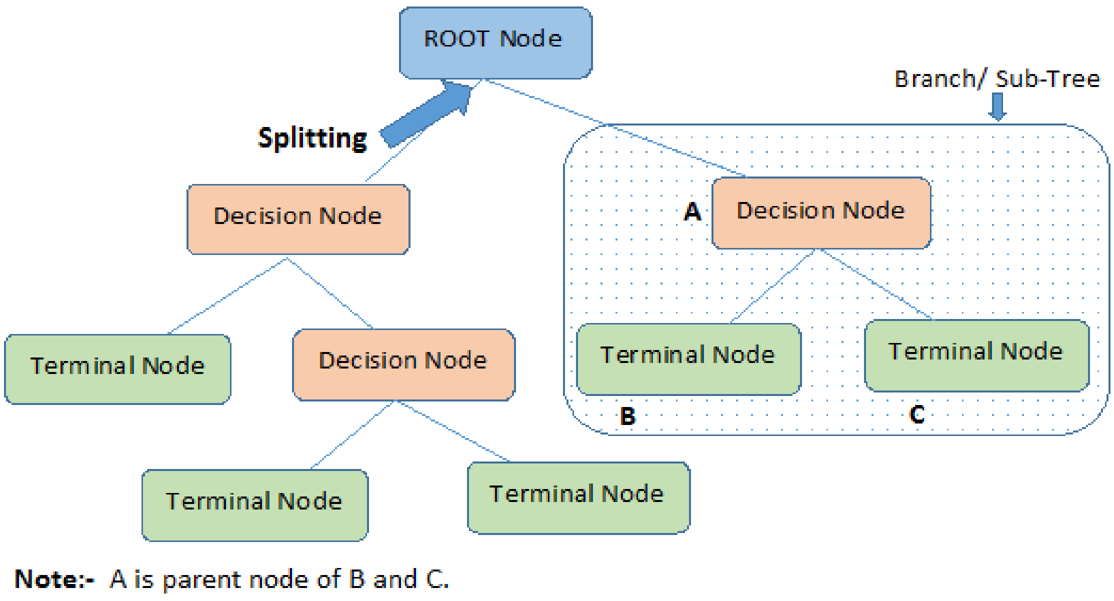


FIGURE 3.4: Decision tree with its components

How to decide when to split?

Accuracy of the decision tree is heavily dependent on the split of the tree. There are multiple ways by which the splitting of the tree can be done, the creation of the sub-nodes increases the homogeneity of the output sub-node. Decision tree select the split which has the highest homogeneity. Following are examples of measures for the split as quoted by the Quinlan [42].

Gini Index

Gini impurity measure can be used as hyperparameter for the splitting of the tree into the sub-nodes. Before any split on the particular attribute the Gini impurity is measured by the help of the given formula and the Gini impurity is measured and the attribute is select which as had the high score of the Gini impurity.

$$G_i = 1 - \sum_{k=1}^{n} P_{i,k}^2$$

where G_i stands for the Gini score and i is the particular node and n stands for the number of nodes.$P_{i,k}$ is the ratio class k with all the classes.

Information Gain and Entropy

Entropy can be also be used at the measure for the checking the split for the particular attribute, similarly like the Gini index the entropy or the information gain the attribute having maximum entropy is used for the splitting. Entropy H_i can be measured as

$$H_i = - \sum_{\substack{k=1 \\ P_{i,k} \neq 0}}^{n} log(P_{i,k})$$

where n stands for the number of nodes. $P_{i,k}$ is the ratio class k with all the classes.

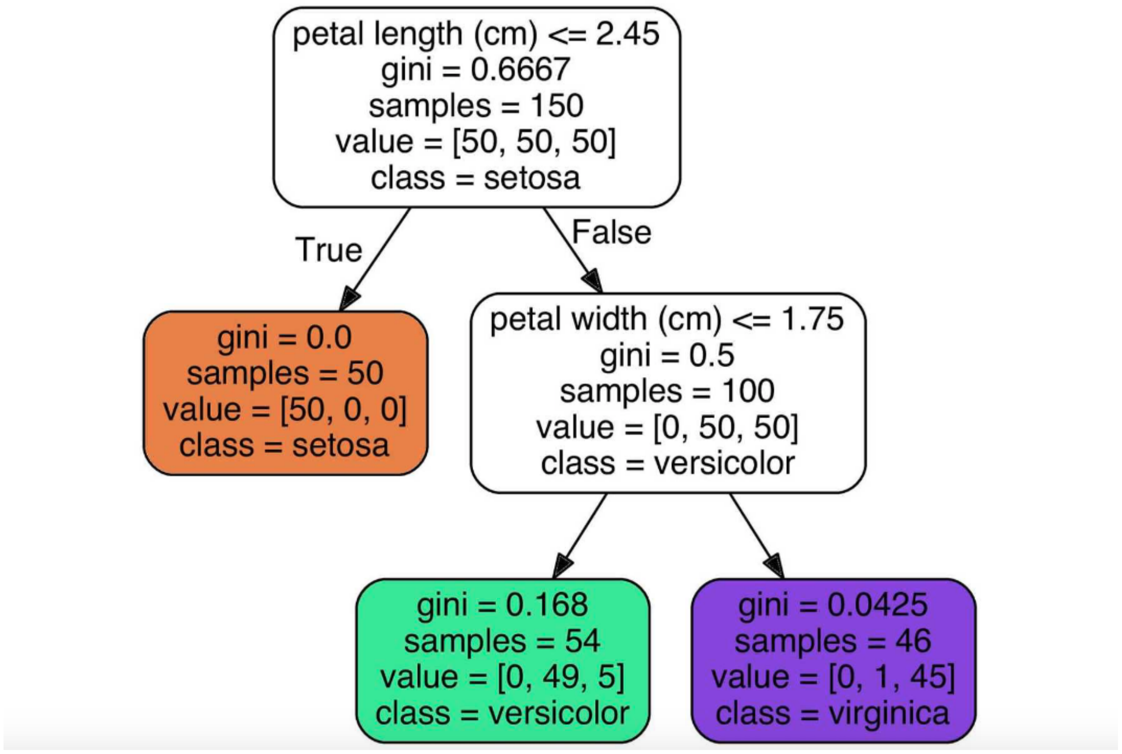


Figure 3.5: Iris Decision Tree

3.3.2 Support Vector Machine

Support Vector Machines is the algorithm or the classification techniques by which the hyperplane can be obtain which can distinguish between the n dimensional data set. It is intended to maximise the margin between them which are separated by the derived hyperplane which represent the two different classes [43]. SVM is also called large margin classifier because of it property to separate the two classes with the maximum distance. There are two types of the SVM namely the
Linear Support Vector Machine which generate the linear hyperplane to separate the two classes

$$\alpha_1 w_1 + \alpha_2 w_2 + \alpha_3 w_3 .... \alpha_n w_n = +1/-1 \tag{3.1}$$

$$\tilde{w} = \sum_i \alpha \tilde{s_i} \tag{3.2}$$

the above equation represent the linear SVM and the weight for the each parameter or attribute of the equation.

$$f(x) = \sigma(\sum_i \alpha_i \Phi(s_i).\Phi(x)) \tag{3.3}$$

equation 3.3 represent the hyperplane of the classifier where α_i represent the attributes and x is the input to the equation.
Non-linear Support Vector Machine generate the polynomial hyperplane to classify. Equation 3.4 and 3.5 represent the polynomial and multiple attribute equation for the Support Vector Machine.

$$\alpha_1 \alpha_2 w_1 + \alpha_1 \alpha_2 w_2 = +1 \tag{3.4}$$

$$\alpha_1^2 w_1 + \alpha_1^2 w_2 = +1 \tag{3.5}$$

Support Vector Machines although are very power and can separate the linear and quadratic separable data. Figure 3.6 show the linear SVM on left side but on the right side the data is not linear separable so the non-linear hyperplane is constructed or the Kernel is used.

Kernel

Kernel are use to convert the input feature space into the space which become the linearly separable. It is kind of similarity function which maps the data from the one feature space to another feature space with the aim that the transformed space become the linearly separable. Figure 3.7 represent the transformation using the kernel.

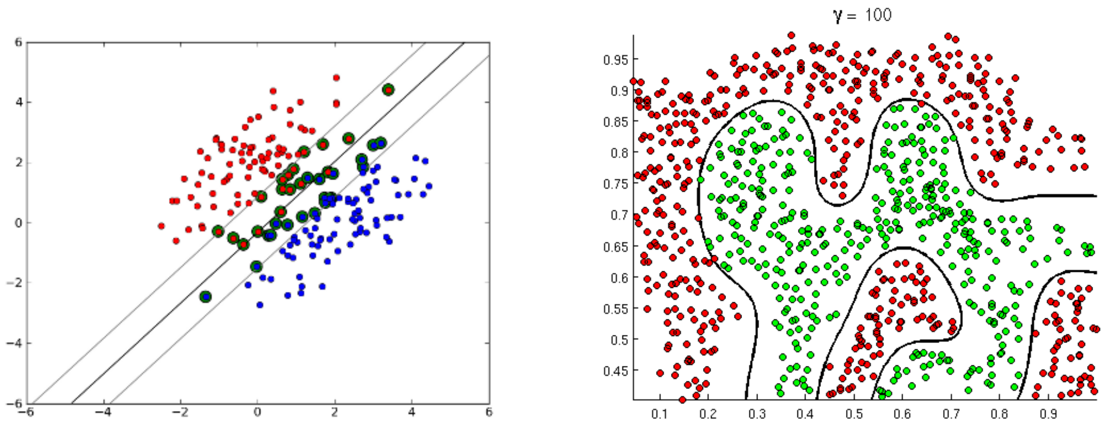


FIGURE 3.6: Linear SVM and its hyperplane on left and non linear separable space separated using the polynomial kernel on right

The Radial Basis Function model is

$$f(x) = \sum_{i=1}^{n} \alpha_i g(x - x_i) \tag{3.6}$$

The output is a linear combination of non-linear functions of the input. The non-linearity is a function of distance only.

An Radial Basis Function or the Gaussian is the solution to the following interpolation problem:

$$\min \sum_{i=1}^{n} L(f(x_i), y_i) + \lambda f_H. \tag{3.7}$$

$$\min \ f_H \quad \text{st.} \quad f(x_i) = y_i, \quad i = 1, 2, ..., n. \tag{3.8}$$

This is similar to a kernel density, except that the coefficients α_i are not restricted to a convex combination, and the basis function g does not have to be a density. following are the different kernel function for Support Vector Machine

Gaussian Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}} \tag{3.9}$$

Polynomial

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i\mathbf{x}_j)^p \tag{3.10}$$

Sigmoid

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\eta\mathbf{x}_i\mathbf{x}_j - \delta)^p \tag{3.11}$$

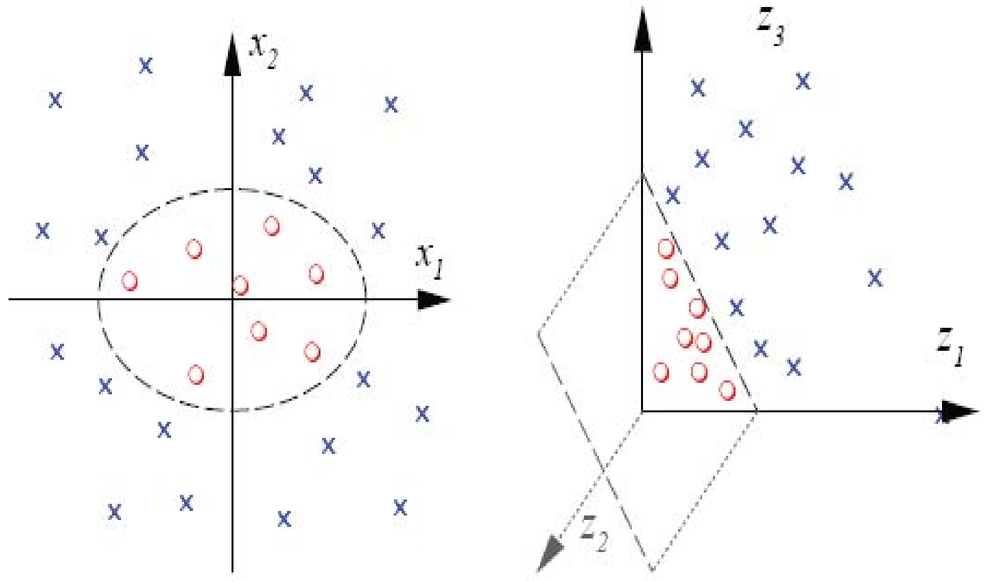


FIGURE 3.7: Transformation of the feature training data

3.3.3 Artificial Neural Networks

Artificial Neural Networks is the type of the machine learning models which are derived from the human brain's neuron. They were first introduced by Warren McCulloh who was a neurophysiologist and Walter Pits who was a mathematician [44]. ANN are inspired by the biological neuron and work in the similar fashion. Figure 3.8 label A shows a biological neuron which is consists of centre and long tail. It had branching extensions called *dendrites*, which receives the signal from the other neurons, a long tail like structure called *axon* which had the terminal and send the signal to the other neurons and the intersection of the two neurons is called *synapse*.

Similarly like the biological neuron the ANN has the structure in which it receives the input as $x_1, x_2...x_n$ and applied the activation function and produce the output y_i which later act as the input of the other neurons.

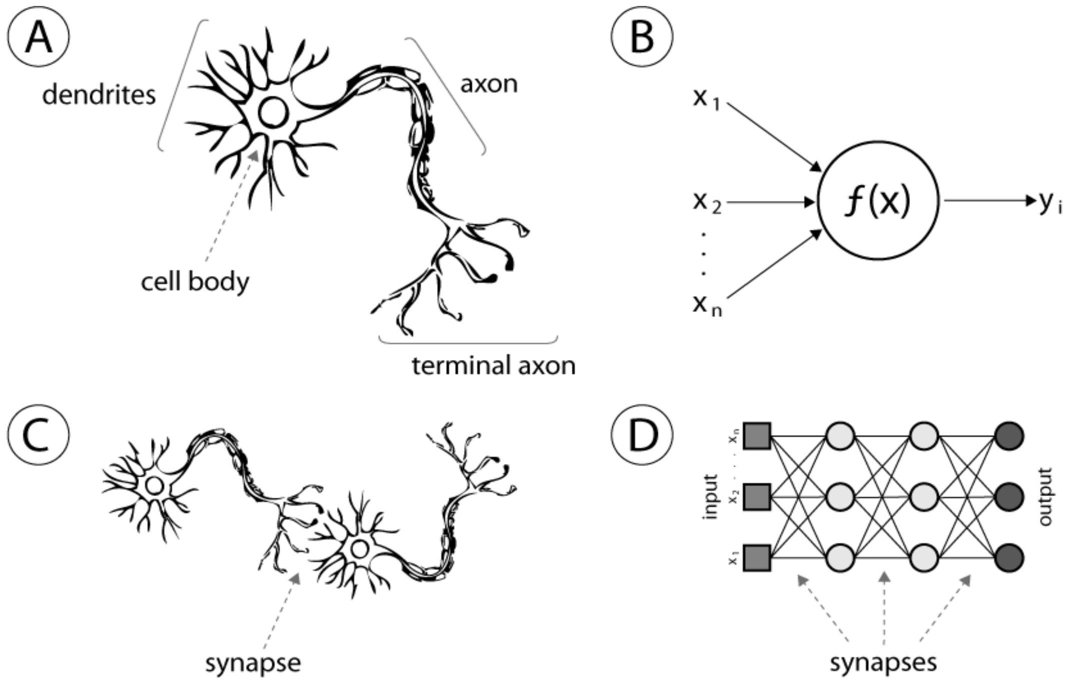


FIGURE 3.8: Derivation of ANN from a biological Neuron

Perceptons
The perceptron was the earliest and the most simplest ANN architecture. It is the basic building block of the ANN which was introduced by the Frank Rosenblatt in 1957. It take the input $x_1, \ldots, x_n$ and then produce the single output, weight introduced at the each input as $w_1, \ldots, w_n$ which express the importance of each of the input, more the weight more importance to the particular input is given, then output which is given as 0 or 1 by input the $(z = w_1x_1 + w_2x_2 + \ldots + w_nx_n)$ into the function and the learning is done by adjusting the weight w so the accuracy of the prediction is increased

$$y(x_1, \ldots, x_n) = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b) \tag{3.12}$$

$$output = \begin{cases} 1, & \text{if } z < 0. \\ 0, & \text{if } z \geq 0 \ . \end{cases} \tag{3.13}$$

the output of the perceptron is only 0 or 1, making it not possible to work on the classification of the multiple categories or classes. Hence the solution to this problem is Multi-Layer Perceptrons, in which the output of one perceptron act as the input to the other perception and a structure is generated which comprises of *input layer*, *hidden layer* and *output layer*, but the complex algorithm have been proposed which has the most optimised structure of the neural network(e.g. NEAT Neuro Evolution through Augmenting Topologies). Artificial Neural Networks are the layers of perceptions which work simultaneously to do classification.

Sigmoid neuron

The main limitation of perceptrons is that there are very difficult to tune, because minimum changes in the weights and bias of any single perceptron can cause the output to change drastically by completely flip, from 0 to 1 or vice versa. And if we have a network of perceptrons, a single flip can completely change the behaviour of the rest of the network.This problem was solved by the introduction of the sigmoid neuron. Exactly as the perceptron, a sigmoid neuron has inputs $(x_1, \ldots, x_n)$ and it also has weights for each input and a bias, but the output can be a real number. The sigmoid function is given in equation 3.14 and it is notable that the sigmoid function is having a property that $f'(z) = f(z)(1 - f(z))$ which easy calculation of adjusted in backpropagation algorithm.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.14}$$

Loss Function

To measure the performance of the neural network it is defined a function, typically named cost or loss function which given a prediction or set of predictions and a label or a set of labels measures the discrepancy between the algorithms prediction and the correct label. There are various cost functions but the most common and simple in neural networks is the mean squared error (MSE) which can be given by equation 3.15.

$$E = \frac{1}{m} \sum_{i=1}^{m} \|o_i - t_i\|^2 \tag{3.15}$$

where:

- o_i actual outputs
- t_i desired outputs
- m number of training examples

The goal in training neural networks is to find weights and biases that minimises some cost/loss function. For that, it is used an algorithm called gradient descent.

Gradient Descent Algorithm

Gradient descent algorithm is an algorithm for minimising the loss function. It is used to find the local minimum of the loss function. The gradient can be defined as follow

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n} \right]$$

this gradient is used to find the minimum of the loss function, the outline of the algorithm can be give as

1. Start with the random initialisation of the weight and bias in the ANN. It is important to initialise all the value randomly, as if they are initialise to particular value then they all will end up in the learning in the same way again and again. Random initialisation breaks the symmetry of the learning.

2. Keep iterating to update the parameters W,b as follows until it hopefully ends up at a minimum.The following equation represent the calculation of adjusted weights.

$$\Delta\vec{w} = -\alpha\nabla E[\vec{w}] \tag{3.16}$$

i.e.,

$$\Delta w_i = -\alpha\frac{\partial E}{\partial w_i} \tag{3.17}$$

$$\Delta b_i = -\alpha\frac{\partial E}{\partial b_i} \tag{3.18}$$

where α is the learning rate and $W^l_{i,j}$ and $b^l_{i,j}$ denote each weight and bias in a particular layer l in the Neural Network, respectively.

The derivative of the overall loss function can be computed as:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

The learning rate is used to control how big a step is taken downhill with gradient descent. Selecting the correct learning rate is critical. On one hand, if α is too small, gradient descent can be slow. On the other hand, if α is too large, gradient descent can overstep the minimum and even diverge.

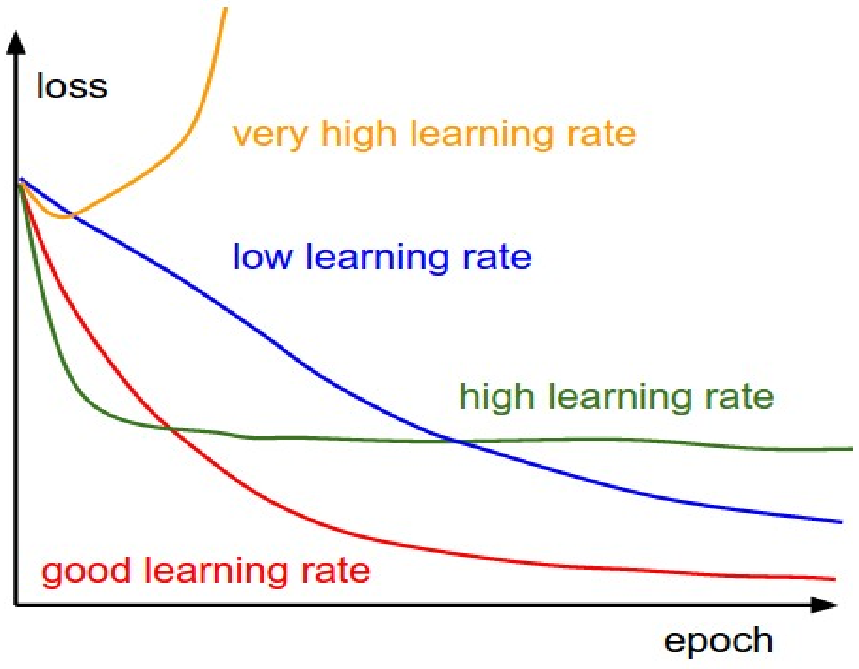


FIGURE 3.9: Effects of different learning rates

Backpropagation
The key step is to compute all those partial derivatives presented before. Therefore, to compute efficiently backpropagation algorithm. Steps involve in the backpropagation are

1. For each training example, Do
2. Input the training example to the network and compute the network outputs
3. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

4. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

5. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \alpha \delta_j x_{i,j}$$

Regularization

Weight of the neural network can grow uncontrollably so in order to regularise those weight and extra term is added called regularization term shown in equation 3.19 that opposes weight growth. It also avoid the overspecialisation means to divert the output towards only a particular input by increasing its weight very tremendously

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} - \alpha w_i \tag{3.19}$$

3.4 Performance Measure of Different Techniques

There are many different techniques by which the performance of the classification model could be measured. In most of the techniques the trained model is taken and the test set created either by the cross fold or by newly selected test set and then the prediction is done on the trained model and then the output are recored to measure the performance such as *precision*, *recall*, *accuracy* and *f-score* which are defined by the true positive which means the input which are true or belong to the class are correctly predicted by model, true negative means the input which does not belong to the class is correctly predict out of that class , false positive means the input does not belong to that class but classifier model incorrectly predict that it belong to that class and false negative means that the input sample which does belong to that class is predict not belonging to that class.

3.4.1 Precision

$$PRECISION = \frac{TRUE\ POSITIVE}{TRUE\ POSITIVE + FALSE\ POSITIVE} \tag{3.20}$$

3.4.2 Recall

$$RECALL = \frac{TRUE\ POSITIVE}{TRUE\ POSITIVE + FALSE\ NEGATIVE} \tag{3.21}$$

3.4.3 Accuracy

$$ACCURACY = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.22}$$

where $TP\ is\ TRUE\ POSITIVE$, $TN\ is\ TRUE\ NEGATIVE$, $FP\ is\ FALSE\ POSITIVE$ and $FN\ is\ FALSE\ NEGATIVE$ respectively.

3.4.4 F1-score

$$F1 - SCORE = 2\frac{PRECISION * RECALL}{PRECISION + RECALL} \tag{3.23}$$

3.4.5 ROC Curve

Receiver Operating Characteristic curve is the curve which is plot against true positive rate or Sensitivity as Y axis and false positive rate 100-specificity as X axis for the different points of test set result. The more the area under the curve more the classification is better. The "steepness" with respect to the Y equal to X axis plays a significant role in performance of the classifier. Figure 3.10 shows the ROC curves with the perfect classifier which is always 1, good classifier which above the random guessing line and the worse classifier which is below the random guessing.

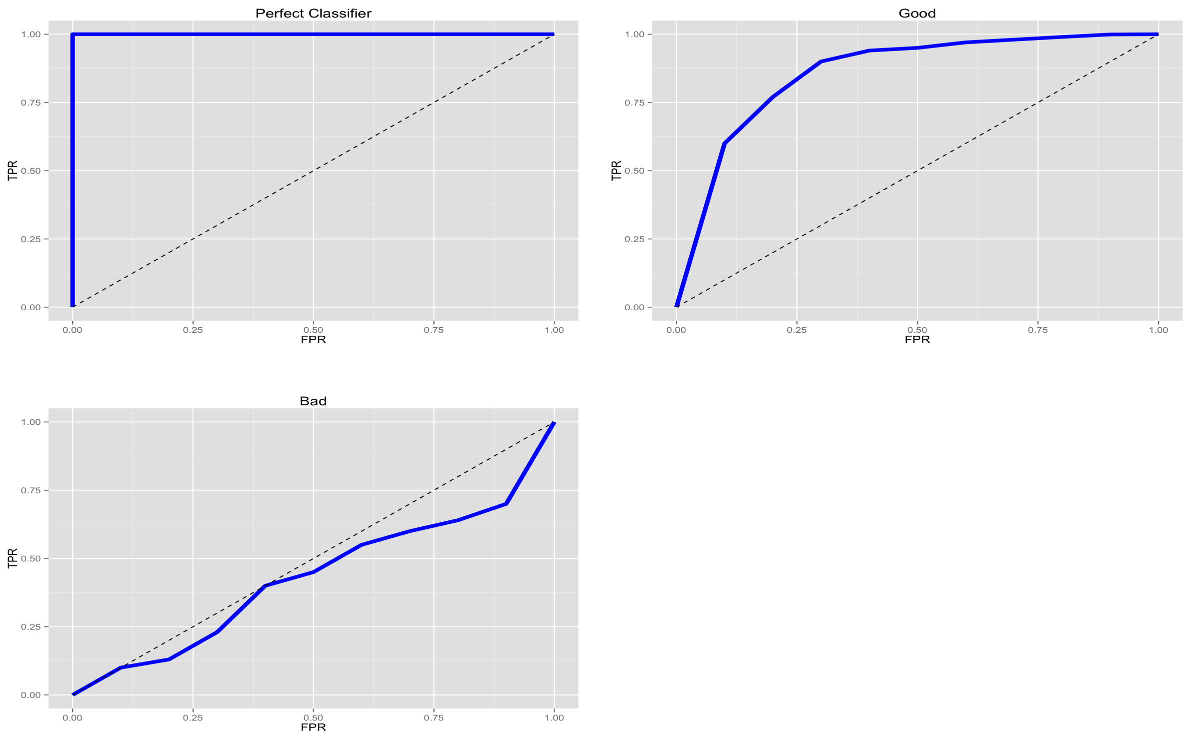


FIGURE 3.10: ROC curves

Chapter 4

Artificial Neural Networks with NEAT

Although Artificial Neural Network can classifies the highly complex input vector and the accuracy of the classification increases with the increase in the number of the layers and the number of the neurons, the complexity of the neural and the dilemma for choosing the structure of the neural network and Zhang et al. [45] argued that the speed and accuracy of the learning is greatly affected by the network complexity and proposed the approach to select the optimal neural network by genetic algorithm [46].

NeuroEvolution of Augmenting Topologies(NEAT) is designed to take the benefit of the minimisation of the dimensionality of the search space for the weight of the connection in the neural network by Genetic Algorithm. If the structure of the neural network is minimised then the speed of learning is increased significantly.

4.1 Background

Angeline et al.[47] suggest the neuroevolution approach for the recuurent neural networks, Dasgupta et al. design the application specific neural networks using the genetic algorithm [48], Fullmer et al.[49] uses the marker genetic encoding creating a finite state automata, Gruau et al.[50] compared the cellular and direct encoding of genetic algorithm for neural networks, Lee et al.[51] uses the link list structure for the neural network's structure, Mandisher did work on representation and evolution of neural network in better way[52], Maneizzo[53] uses the genetic evolution for topology and weight selection of neural networks and Yao ate al.[54] design the neural network using evolution by genetic algorithm. The work can explained in detail in different section of encoding of network.

4.1.1 TWEANN Encoding

TWEANN encoding use the efficient genetic representation of the nodes. It can represent both the direct encoding and indirect encoding schemes, in direct encoding every connection and the node which will appear in the phenotype is specified in the genome, while in indirect encoding which is the compact representation of the phenotype as the every connection and node is not specified in the genome rather they can be derived from it.

4.1.2 Binary Encoding

Like the traditional string representation in the genetic algorithm similarly the direct encodings and indirect encoding are represented as the bit string, Dasgupta at el. [48] uses the bit string to represent the connection matrix of the network of the ANN. But this representation is become huge string when the number of nodes are very large and the crossover to the string may or may not represent the correct structure of the ANN.

4.1.3 Graph Encoding

Graph structure is use to represent the structure of the ANN [55], Subgraph is used to be passed to another during the crossover to make the breeding of the new generation of the ANN structure. Graph can be represented as the two by two matrix and the crossover could not be easily done on that to produce the new graph .

4.1.4 Nominating

As the crossover of the different network could lead to loss of functinality, Yao et al. [54] had proposed the there will no crossover between the different network rather than new links and nodes are address to the network structure of the ANN.

4.1.5 Indirect Encoding

Gruau et.al [50] *Cellular Encoding* is used in the indirect encoding scheme in which genomes can be written with the help of the graph transformation language where transformation are motivated by the cell divisions and generations, similarly the new generation are produced.

4.2 NeuroEvolution of Augmenting Topologies

Aritifical neural networks is a type of supervised classification technique in which the learner is present which teaches the ANN to adjust it weights but the Evolution of the neural network structure with the help of the genetic algorithm is a type of reinforcement learning in which model i.e neural network structure and weights are created and destroyed like the generation of species, and the best model(generation) is selected as the classification model.

The NEAT algorithm uses the genetic algorithm searching through the space for the network weight and structures.It has the following properties:

1. Genetic representation of the structure such that it can be easily processed and the genetic operators like mutate and crossover can applied easily.
2. Protect the innovation (means the new structure or the node added to the topology or to the structure of the ANN) from repeating again and again by marking them as the innovation number.
3. Minimise the topology throughout the training without using complex function like gradient descent and backpropagation.

4.2.1 Genetic Encoding

The topology or the structure can be represented as shown in the figure 4.1. The information of the network can be represented by the genome which contains the connection genes and node genes. The connection gene represent the connection between the different node by input coming from the node as *InNode* and output to the node as *OutNode*, the weight of the link as *Weight*, is it *Enabled* and the *innovation number*. Every connection gene has the corresponding *innovation number* which is used to track the history of the genes evolution.

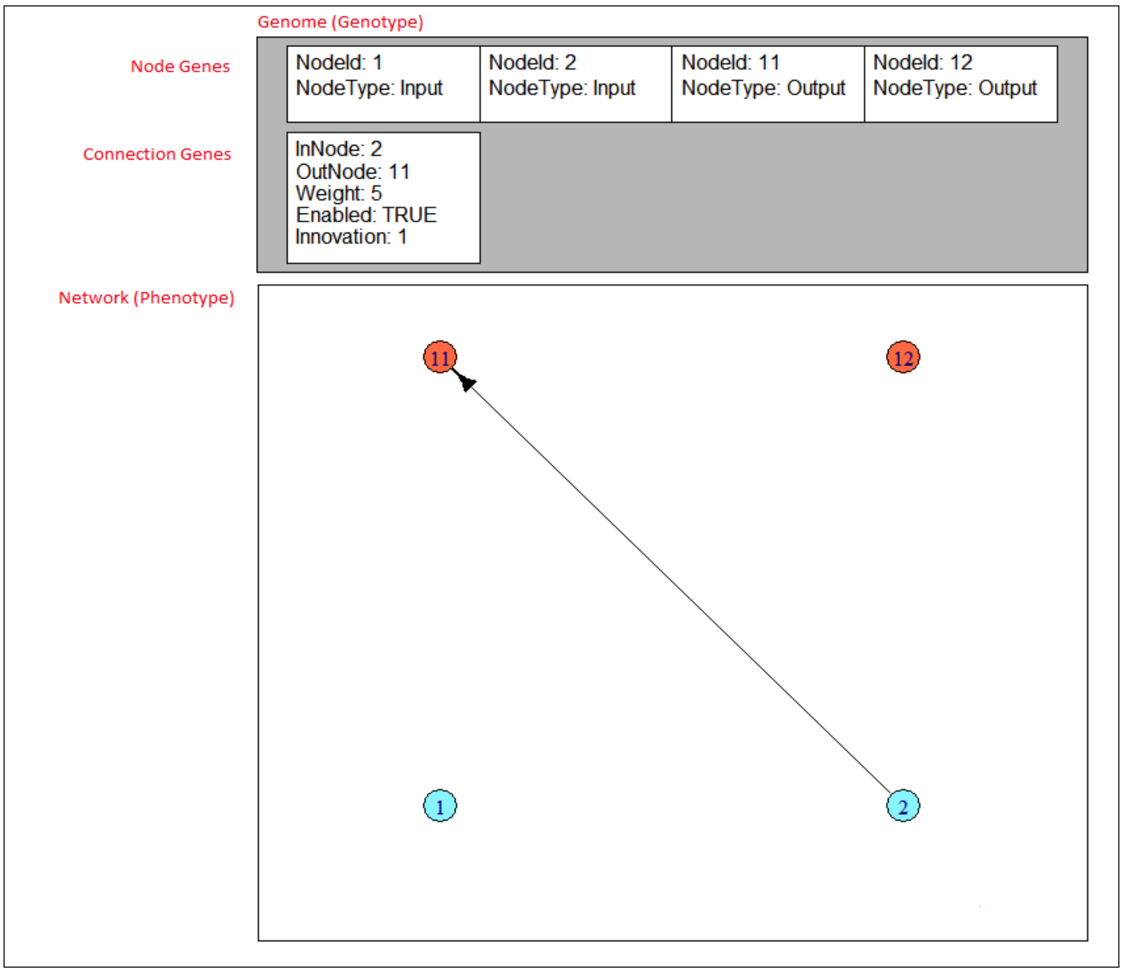


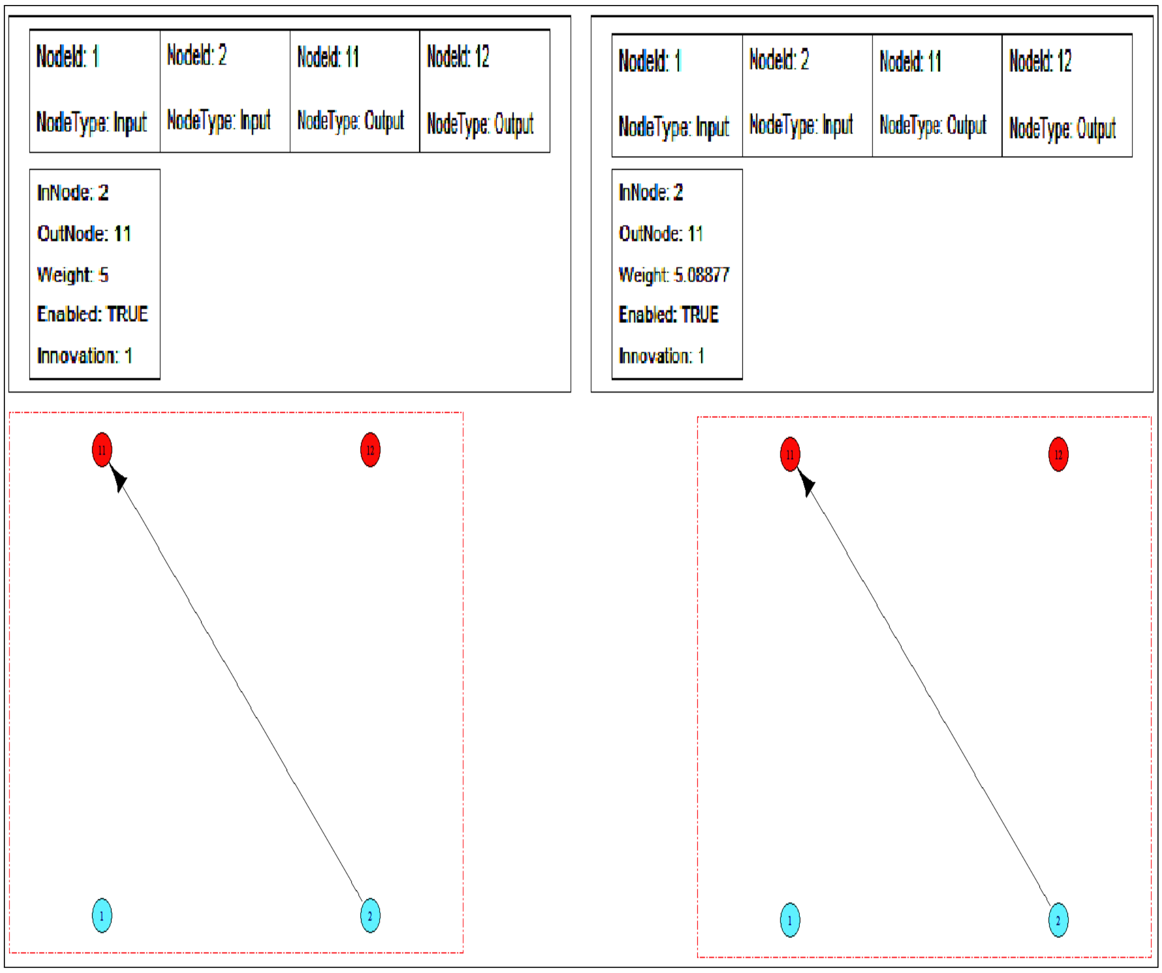
FIGURE 4.1: Genome of the topology of Neural Network by NEAT

4.2.2 Mutation

The NEAT algorithm has the structure dependent mutation which is similar to the genetic algorithm like in genetic algorithm the particular bit of the string is changed similarly here the connection genome is altered.

Point Mutate

Point mutate is the mutation in which the weight of the connection genome is changed by the some factor. Like in genetic algorithm the mutation binary bit is selected randomly similarly the connection genome is selected randomly for the mutation.



NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2
OutNode: 11
Weight: 5
Enabled: TRUE
Innovation: 1

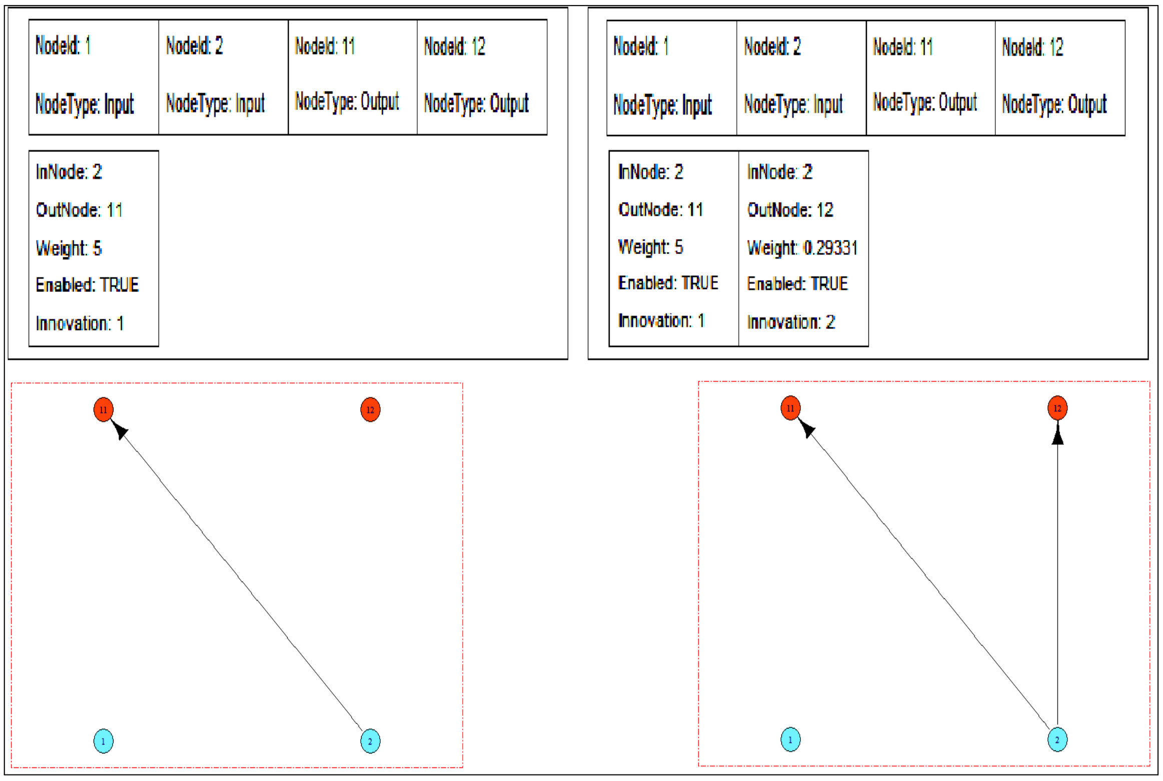
NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2
OutNode: 11
Weight: 5.08877
Enabled: TRUE
Innovation: 1

FIGURE 4.2: Point Mutate by NEAT

Link Mutate

In this mutation the new connection genome is produced, the new connection is tracked to the history by the searching it in the previous connection genome and then innovation number is allotted to it as shown in figure 4.3. The list of the innovation is kept and every time the genome is produced it is check in this list wether the genome is produced earlier or not, if not then new number is allocated to it and is added to the list, if produced earlier then the genome is discarded and again new genome is produced.



NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: TRUE Innovation: 1

NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: TRUE Innovation: 1	InNode: 2 OutNode: 12 Weight: 0.29331 Enabled: TRUE Innovation: 2

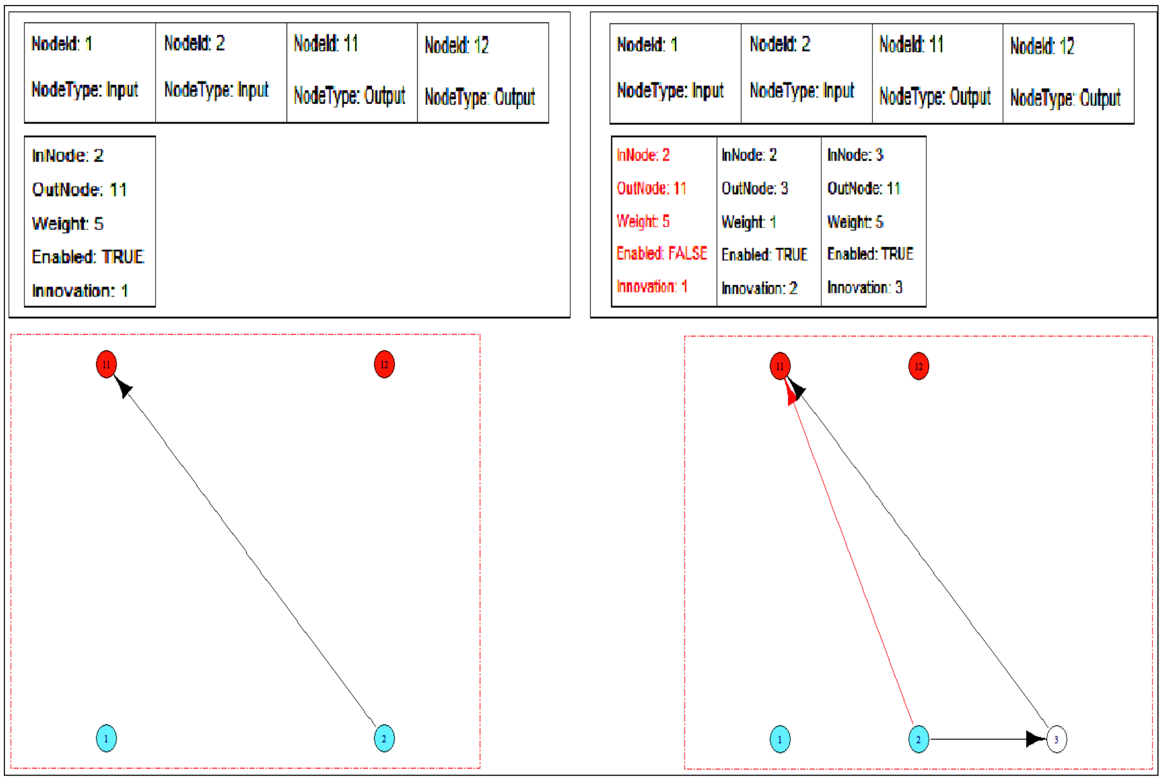
FIGURE 4.3: Link Mutate by NEAT

Node Mutate

In the Node mutate shown in figure 4.4 the new node genome is added or the randomly selected previous node genome is deleted and the equivalent connection genome of the node are added or deleted.

Enable/Disable Mutate

In this mutation the randomly selected connection genome is enabled or disabled. The connection genome is selected randomly and it is enabled or disabled by changing the value for *Enabled* in the list of connection genomes.



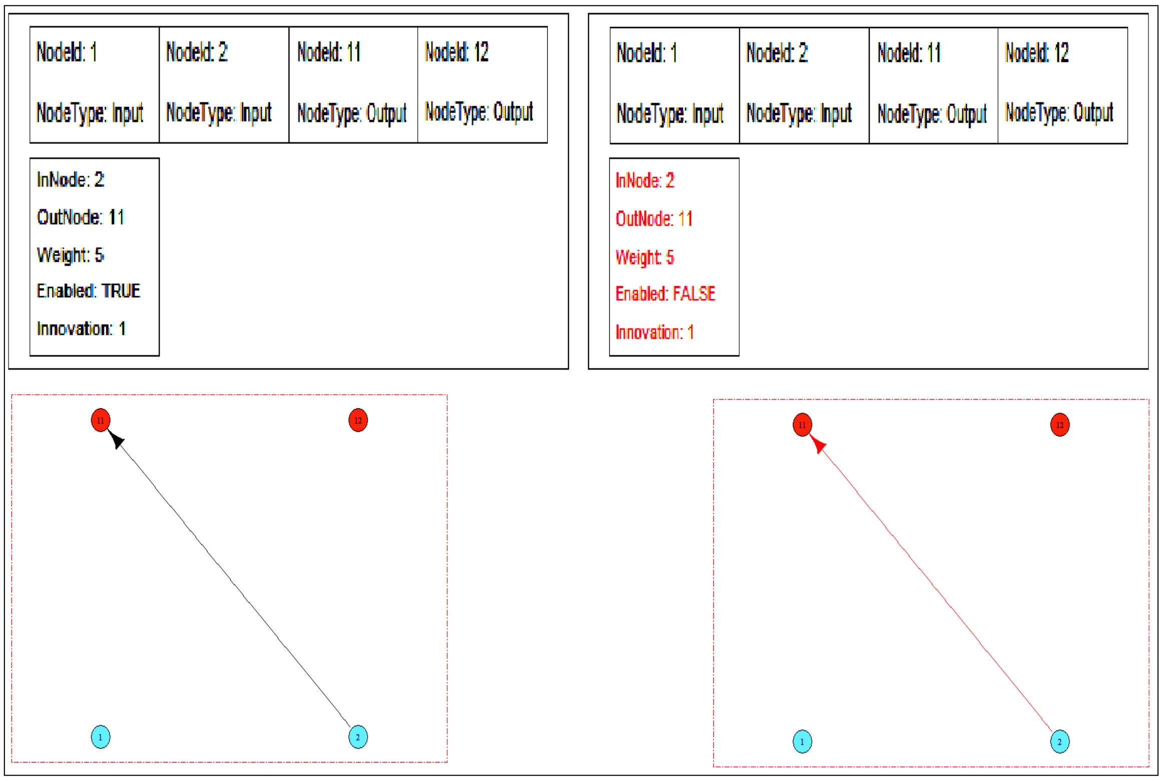
NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: TRUE Innovation: 1

NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: FALSE Innovation: 1	InNode: 2 OutNode: 3 Weight: 1 Enabled: TRUE Innovation: 2	InNode: 3 OutNode: 11 Weight: 5 Enabled: TRUE Innovation: 3

FIGURE 4.4: Node Mutate by NEAT



NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: TRUE Innovation: 1

NodeId: 1 NodeType: Input	NodeId: 2 NodeType: Input	NodeId: 11 NodeType: Output	NodeId: 12 NodeType: Output

InNode: 2 OutNode: 11 Weight: 5 Enabled: FALSE Innovation: 1

FIGURE 4.5: Enable/Disable mutate by NEAT

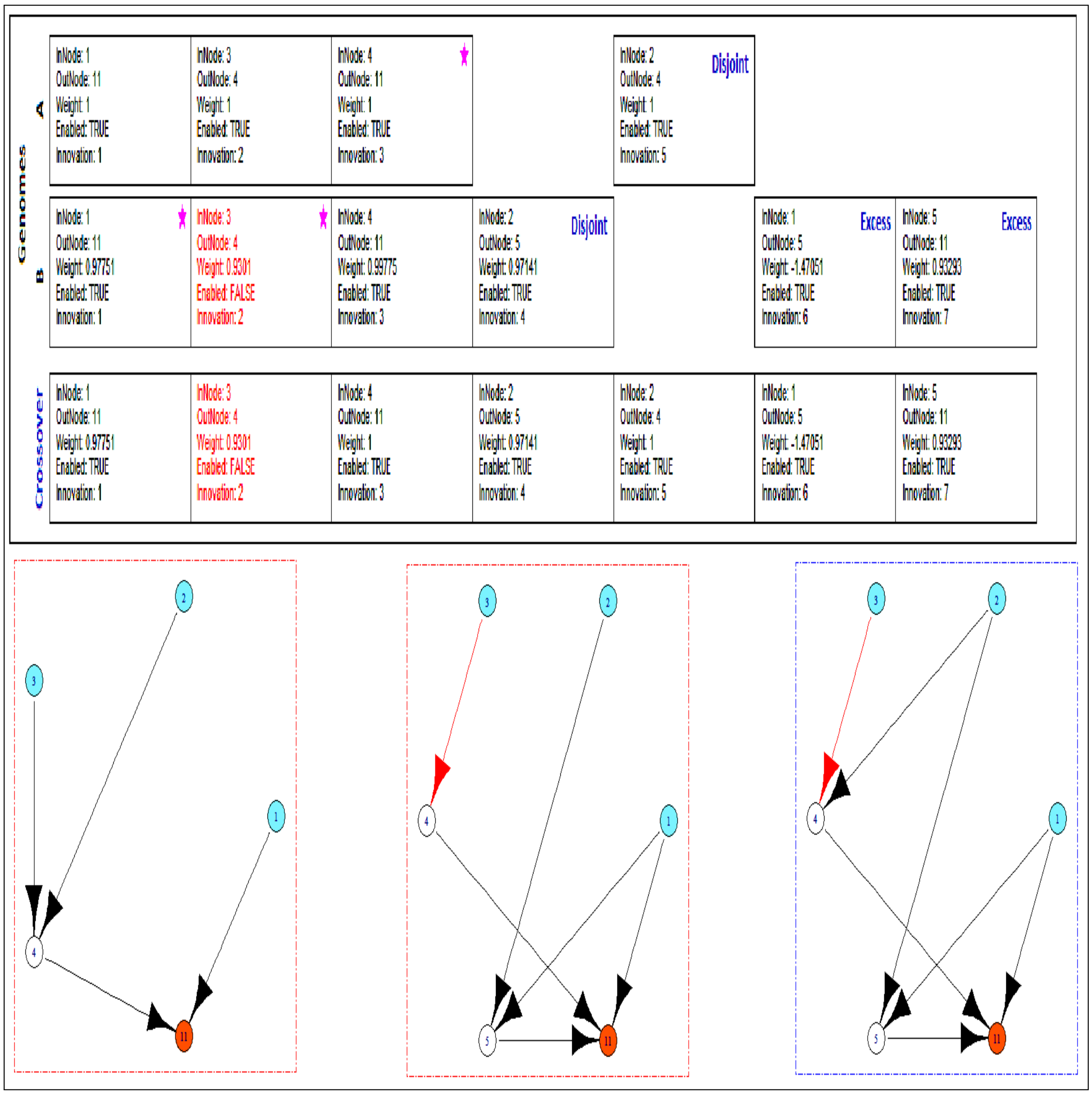

Genomes
A
InNode: 1 OutNode: 11 Weight: 1 Enabled: TRUE Innovation: 1
InNode: 3 OutNode: 4 Weight: 1 Enabled: TRUE Innovation: 2
InNode: 4 OutNode: 11 Weight: 1 Enabled: TRUE Innovation: 3
InNode: 2 OutNode: 4 Weight: 1 Enabled: TRUE Innovation: 5
Disjoint
B
InNode: 1 OutNode: 11 Weight: 0.97751 Enabled: TRUE Innovation: 1
InNode: 3 OutNode: 4 Weight: 0.9301 Enabled: FALSE Innovation: 2
InNode: 4 OutNode: 11 Weight: 0.99775 Enabled: TRUE Innovation: 3
InNode: 2 OutNode: 5 Weight: 0.97141 Enabled: TRUE Innovation: 4
Disjoint
InNode: 1 OutNode: 5 Weight: -1.47051 Enabled: TRUE Innovation: 6
Excess
InNode: 5 OutNode: 11 Weight: 0.93293 Enabled: TRUE Innovation: 7
Excess
Crossover
InNode: 1 OutNode: 11 Weight: 0.97751 Enabled: TRUE Innovation: 1
InNode: 3 OutNode: 4 Weight: 0.9301 Enabled: FALSE Innovation: 2
InNode: 4 OutNode: 11 Weight: 1 Enabled: TRUE Innovation: 3
InNode: 2 OutNode: 5 Weight: 0.97141 Enabled: TRUE Innovation: 4
InNode: 2 OutNode: 4 Weight: 1 Enabled: TRUE Innovation: 5
InNode: 1 OutNode: 5 Weight: -1.47051 Enabled: TRUE Innovation: 6
InNode: 5 OutNode: 11 Weight: 0.93293 Enabled: TRUE Innovation: 7
1
2
3
4
5
11

FIGURE 4.6: Crossover between Genome in NEAT

Crossover

Crossover or mating takes place between the two parent genomes. Both the parent genome use to lined up using the innovation number, and a particular innovation of the parent genome is copied to the child genome which is most fit, if both parents are fit equally then the innovation is randomly selected from either of the parent genome and if the innovation is only present in one of the parent genome means it is disjoint then it simply copied to the child genome.

4.2.3 Steps involved in NEAT

- Pool of the genome is created with k random genomes, where k is the initial population of the genomes.
- Calculate the fitness of the genome apply the training dat or the simulation.
- Assign each genome to the species.
- Produce the offspring species by doing the crossover and mutation.
- Repeat till the desired fitness is not achieved or number of species are not generated.

4.3 Analysis of NEAT & Backpropagation Algorithm

The backpropagation algorithm is the one of the learning algorithm for the neural network, therefore we can only consider it in the case of the speed of learning because the network topology is selected in advance and weights are calculated with minimised error. For the backpropagation algorithm, the gradient at each layer of the neurons is calculated and then it error is propagate to the whole network and the weights at each edges of the neural network are adjusted.

The complexity for the backpropagation training algorithm and weight adjustment is $O(2^n)$ where n is the number of edges in the neural network. While the complexity of the model generation with the weight by NEAT is $O(n^3\ f)$ where n is the number of nodes in final network and f is the number of iteration done to produce the structure of the neural network.

Chapter 5

Results and Evaluation

We have used the different classification techniques and use the Android application permission and APIs as the features for classification model. We have collected the 440 sample of the malware and the benign application, used the Androguard [56] and Mosf [40] to extract the permission and API from the android application package namely .apk file and created a dataset for the classification model such as *Decision tree*, *Support Vector Machine*, *Neural Networks* and *NEAT*.

5.1 Implementaion

The android application package .apk file are processed with Androguard to extract the features Android permission and APIs and collected the permission to create a dataset and collective have the 330 features. We have use the Rapid Miner tools [57], the python Scikit-learn [27] and NEAT-python to perform the classification on the feature dataset of the permissions and APIs. Now we will discuss results and evaluation

5.1.1 Results

We had applied the*Decision tree*, *Support Vector Machine*, *Neural Networks* and *NEAT* classification methods, we had used the dataset as for the training and testing is done 10 cross fold. We discuss the result of each techniques one by one and them compare them with the NEAT techniques

Decision Tree
Using Gini Index

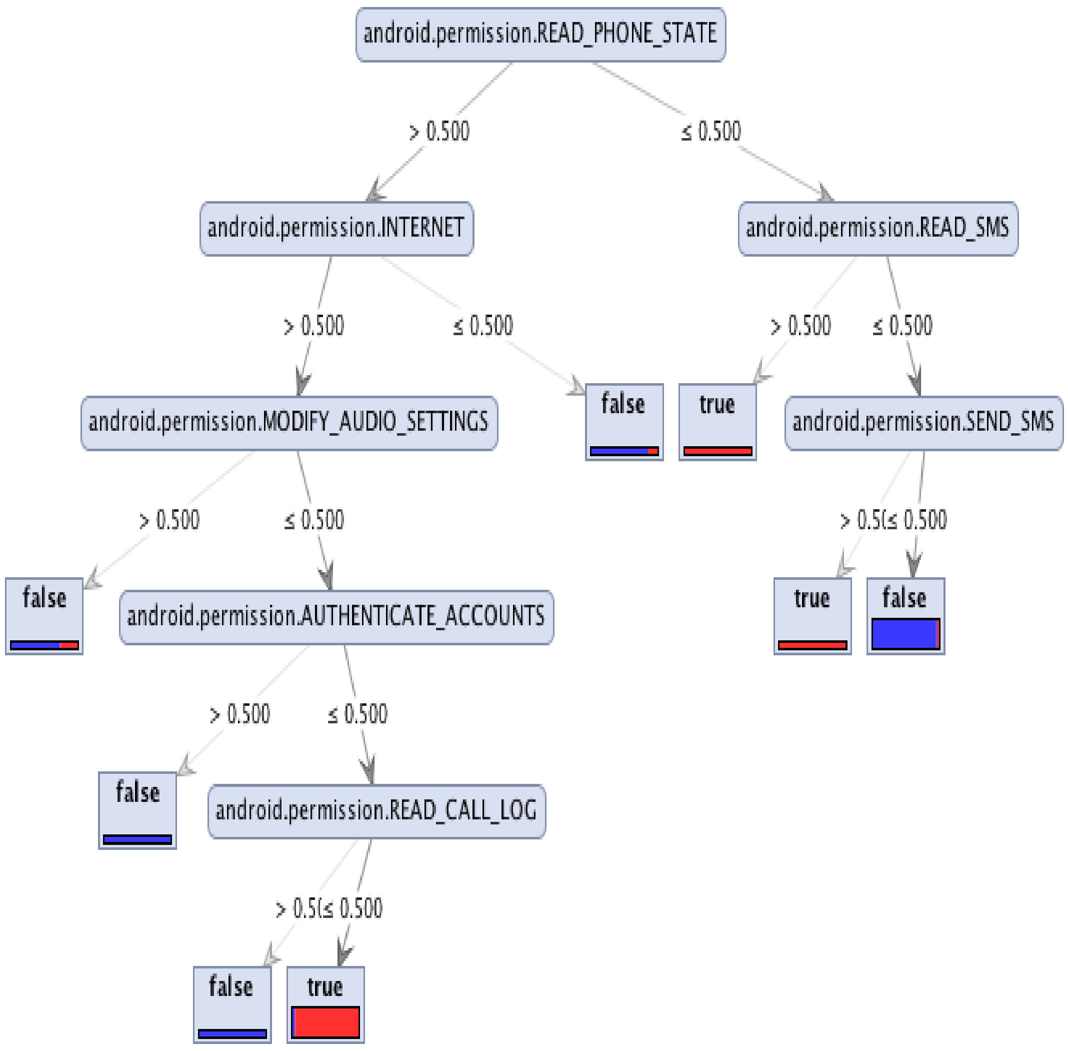

android.permission.READ_PHONE_STATE
> 0.500
≤ 0.500
android.permission.INTERNET
android.permission.READ_SMS
> 0.500
≤ 0.500
> 0.500
≤ 0.500
android.permission.MODIFY_AUDIO_SETTINGS
false
true
android.permission.SEND_SMS
> 0.500
≤ 0.500
> 0.5(≤ 0.500
false
android.permission.AUTHENTICATE_ACCOUNTS
true
false
> 0.500
≤ 0.500
false
android.permission.READ_CALL_LOG
> 0.5(≤ 0.500
false
true

FIGURE 5.1: Decision tree with gini index

TABLE 5.1: Output using Decision Tree with the Gini Index

Itr no.	Accuracy	Precision	Recall	F1-score
1	89.0	88.000	89.759	88.888
2	98.0	100.000	95.918	97.916
3	92.0	96.078	89.090	92.452
4	90.0	100.000	81.481	89.795
5	90.0	89.130	89.130	89.130
6	93.0	100.000	86.274	92.631
7	86.0	88.000	84.615	86.274

Using Information Gain

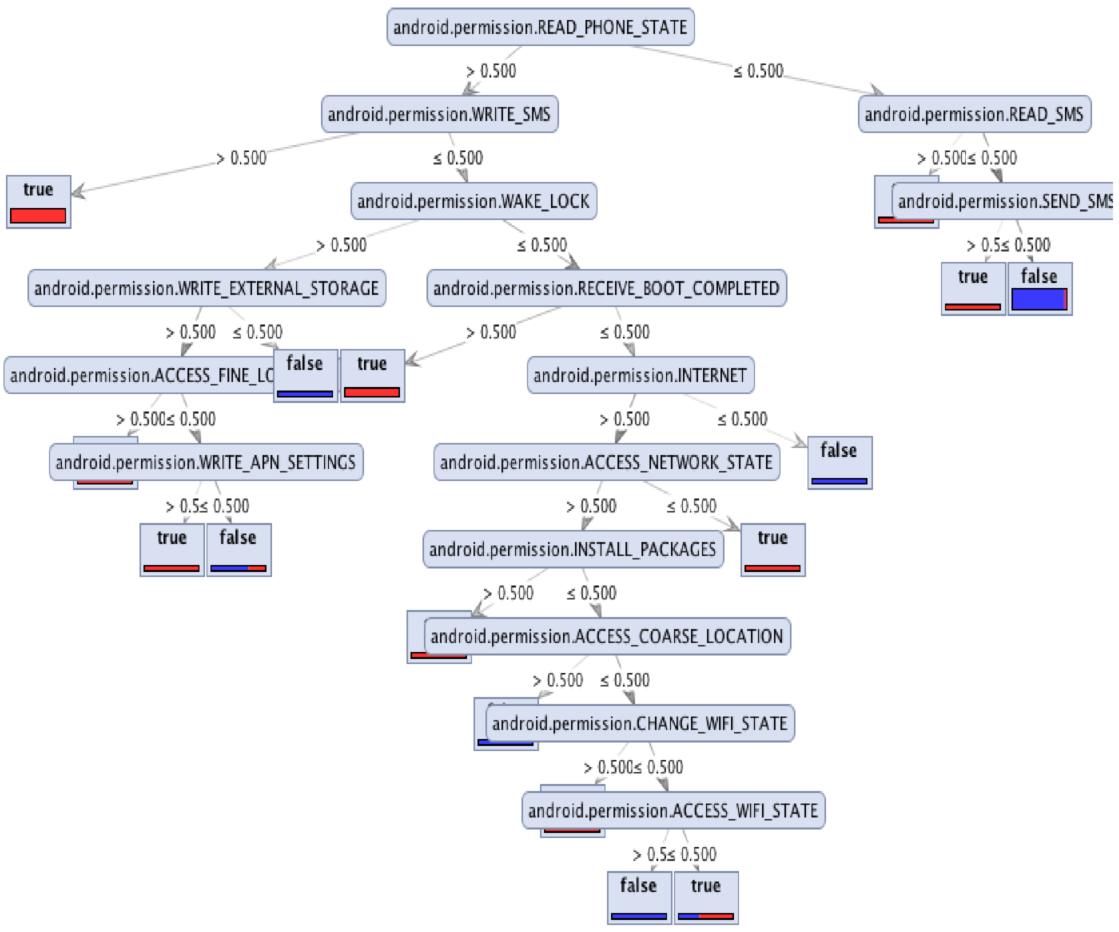

android.permission.READ_PHONE_STATE
> 0.500
≤ 0.500
android.permission.WRITE_SMS
android.permission.READ_SMS
true
android.permission.WAKE_LOCK
android.permission.SEND_SMS
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.ACCESS_FINE_LC
false
true
android.permission.INTERNET
android.permission.WRITE_APN_SETTINGS
android.permission.ACCESS_NETWORK_STATE
android.permission.INSTALL_PACKAGES
android.permission.ACCESS_COARSE_LOCATION
android.permission.CHANGE_WIFI_STATE
android.permission.ACCESS_WIFI_STATE

FIGURE 5.2: Decision tree with information gain

TABLE 5.2: Output using Decision Tree with the Information Gain

Itr no.	Accuracy	Precision	Recall	F1-score
1	88.0	87.755	87.755	87.755
2	96.0	97.872	93.877	95.833
3	92.0	97.959	87.272	92.307
4	91.0	100.000	83.333	90.909
5	89.0	90.697	84.782	87.640
6	92.0	95.744	88.235	91.836
7	88.0	92.478	82.692	87.755

Support Vector Machine

Using Linear Kernel

Table 5.3: Output using SVM using Linear Kernel

Itr no.	Accuracy	Precision	Recall	F1-score
1	96.0	90.153	96.153	96.153
2	92.0	94.000	90.384	92.156
3	97.0	98.000	96.078	97.029
4	95.0	95.740	93.750	94.736
5	89.0	88.461	90.196	89.230
6	93.0	89.361	95.454	92.307
7	95.0	96.226	94.444	95.327

Using Gaussian Kernel

Table 5.4: Output using SVM using Gaussian Kernel

Itr no.	Accuracy	Precision	Recall	F1-score
1	92.0	94.000	90.384	92.156
2	88.0	85.714	92.303	88.888
3	91.0	87.500	96.078	91.588
4	94.0	93.750	93.750	93.750
5	86.0	83.636	90.196	86.792
6	87.0	79.245	95.545	86.597
7	95.0	94.545	96.296	95.412

Using Sigmoid Kernel

Table 5.5: Output using SVM using Sigmoid Kernel

Itr no.	Accuracy	Precision	Recall	F1-score
1	93.0	97.872	88.461	92.929
2	87.0	93.333	80.769	86.597
3	91.0	93.750	88.235	90.909
4	93.0	97.674	87.500	92.307
5	91.0	95.652	86.274	90.721
6	90.0	88.636	88.636	88.636
7	96.0	98.076	94.444	96.226

Neural Network

Table 5.6: Output using Neural Networks

Itr no.	Accuracy	Precision	Recall	F1-score
1	94.0	92.452	96.078	94.230
2	96.0	93.877	97.872	95.833
3	90.0	92.452	89.090	90.740
4	92.0	95.918	88.679	92.156
5	92.0	94.339	90.909	92.592
6	92.0	91.836	91.836	91.836
7	92.0	100.000	96.551	98.245

Neural Network with NEAT

Table 5.7: Output using NEAT

Itr no.	Accuracy	Precision	Recall	F1-score
1	93.0	92.307	94.117	93.203
2	94.0	91.071	95.918	94.444
3	95.0	95.652	93.6170	94.623
4	95.0	98.039	92.592	95.238
5	95.0	98.254	100.000	99.115
6	94.0	93.023	93.023	93.023
7	93.0	95.454	89.361	92.307

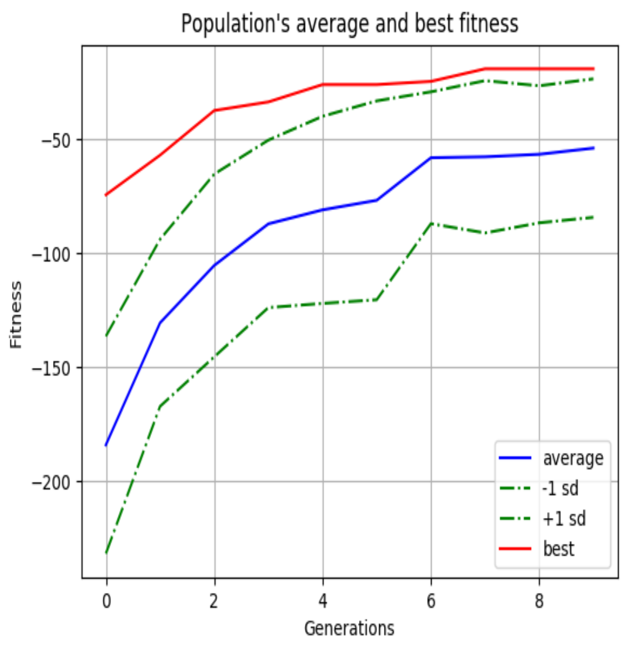


Figure 5.3: Neat with the average and best fit generation

5.2 Evaluation

We have compared the classification results of the various techniques with NEAT, we have measured *Accuracy*,*Precision*, *Recall* and *F1-score* of the each classification with the NEAT. Table 5.8 show the average score of the each techniques we have applied in the series of experiment performed and found that the NEAT had the maximum of 94.142 % of accuracy from all the other one is best from rest of techniques.

The ROC curve was used shown in figure 5.8 to measure the score for the different techniques, we find that the Support Vector Machine was having the curve very close to the NEAT outstanding from the Decision tree and Neural Network, but it was not able to get the same closeness to the 1 like as of the NEAT.

We have also plotted the graph to measure the accuracy rate for each of the techniques and find that the NEAT had the accuracy curve which is consistent and is more frequently touching the maximum value of 100% in figure 5.9, thus have the maximum probability of getting the prediction with maximum accuracy.

Table 5.8: Average classification score of different techniques

	Decision Tree	SVM	Neural Networks	NEAT
Accuracy	92.571	93.857	92.571	94.142
Precision	94.458	94.410	93.134	94.827
Recall	88.043	93.002	93.779	94.383
F1-Score	91.012	93.660	93.861	94.567

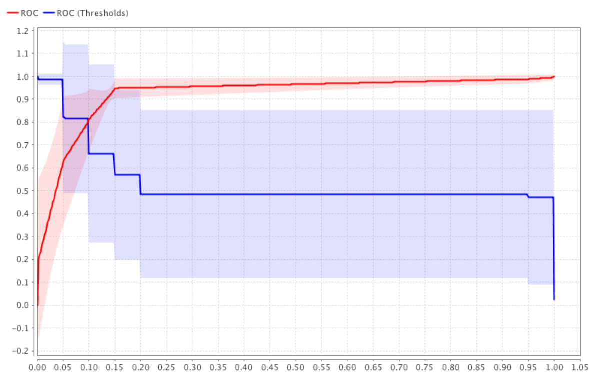


FIGURE 5.4: ROC of Decision Tree

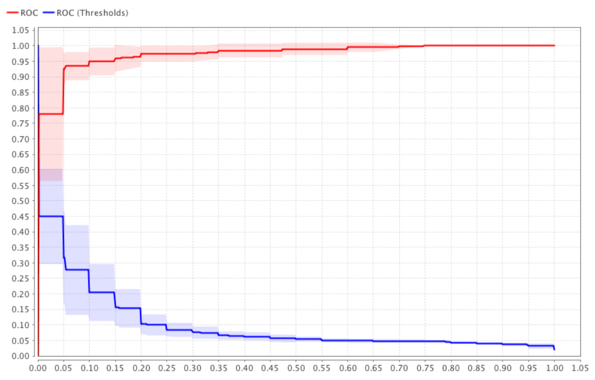


FIGURE 5.5: ROC of SVM

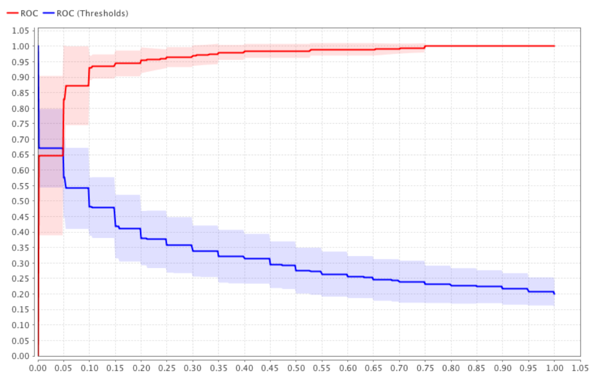


FIGURE 5.6: ROC of Neural Networks

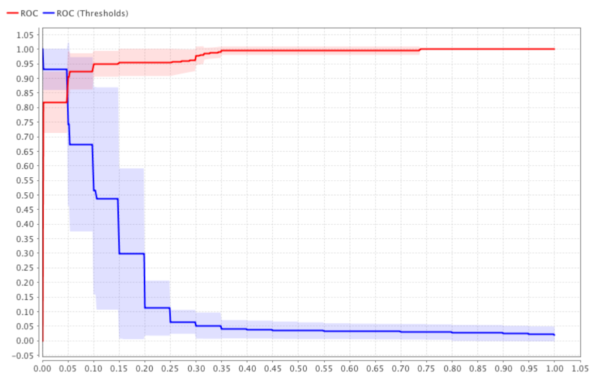


FIGURE 5.7: ROC of NEAT

FIGURE 5.8: ROC of Different Techniques

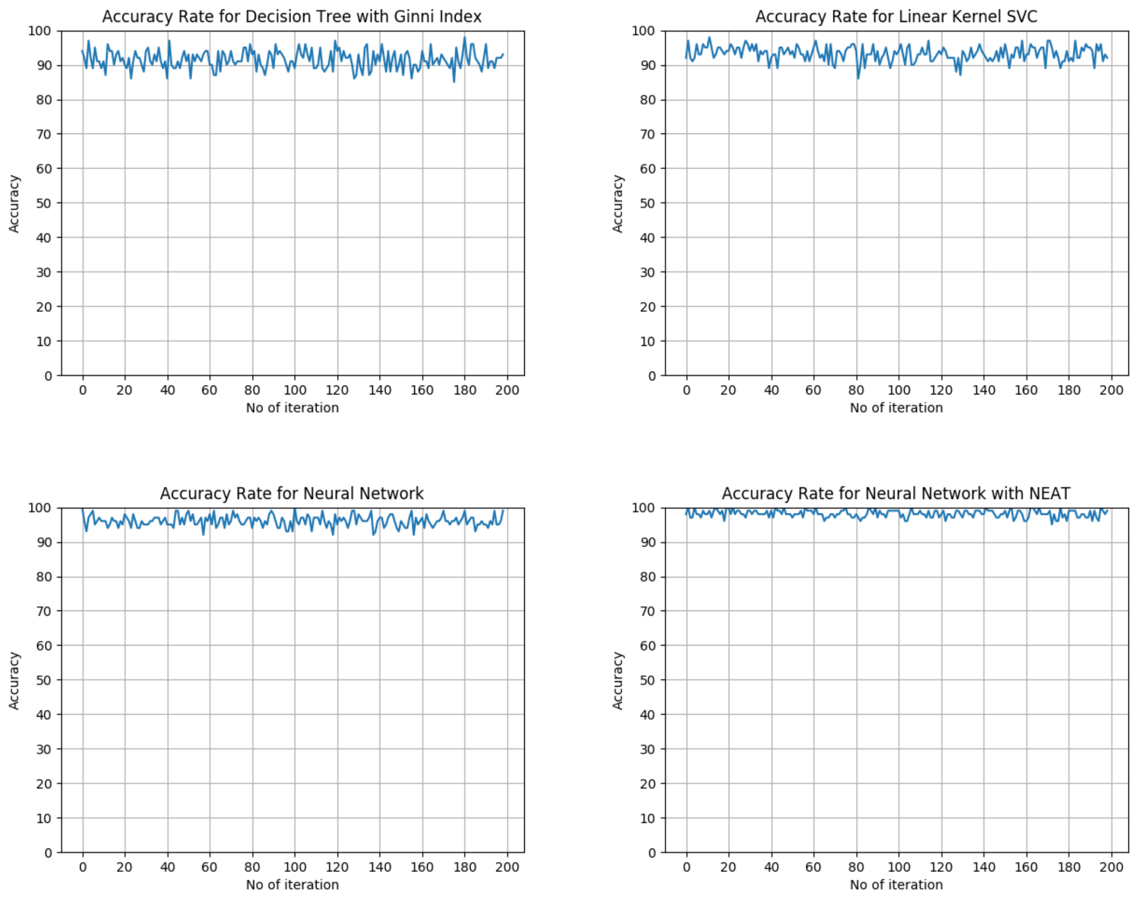


FIGURE 5.9: Accuracy rate Different Techniques

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Malware of any platform either on regular desktop computer or mobile platform like android, iOS etc., are very difficult to detect in the real world scenario, the standard signature based techniques are not that much effective and become useless when the new zero day attack comes in. Although the dynamic analysis are better than the signature based one but they requires the cloud service or the other virtual environment to execute the malware , but there is meagreness of service which could do automated dynamic analysis of malware. Also the malware are able to detect the virtual environment and change their behaviour. In this work we propose a schema for the detection of android malware by using the API and Permission of the android applications and used the far more advance technique for machine learning than the standard techniques called NEAT, which is a type of reinforcement learning which has a high detection(classification) rate and produce a optimised structure which provide the faster detection due to faster processing. Experiment on the real world demonstrate the good performance of the NEAT with respect to the neural networks, Support vector machines and Decision trees using the android permission and API as the features.

6.2 Future Work

In future we will like to extend our work by adding and finding the new feature from the application of the android as well can use more advance machine learning techniques. For the features we can use the hexdump image of the apk, jar file in the apk, database file if present in the apk, process dump of the application in various time interval during execution, and then represent them as the image of NxN matrix. We can use this image as the feature for the machine learning. For the machine learning technique we can use the Deep Learning technique by applying the convolution networks on those images and other processing and perform the training, with this techniques we can use the computational power of Graphic Processing Unit along with Central Processing Unit to increase the rate of learning, and then we can port this model to the android device which can detect the malware.

Bibliography

[1] Wikipedia, “Google Play — Wikipedia, the free encyclopedia.” `http://en.wikipedia.org/w/index.php?title=Google%20Play&oldid=785033684`, 2017.

[2] Androguard, “androguard - github.com/androguard/androguard.”

[3] iBotpeaches, “Apk tool: A tool for reverse engineering android apk files.”

[4] Wikipedia, “Android application package — Wikipedia, the free encyclopedia.” `http://en.wikipedia.org/w/index.php?title=Android%20application%20package&oldid=782491961`, 2017.

[5] “Dataset malware/beningn permissions android — kaggle.”

[6] “Over 1 billion android-based smart phones to ship in 2017 -canalys.com.”

[7] “Operating system market share - netmarketshare.com.”

[8] alcatel lucent, “Kindsight security labs malware report,” tech. rep., Kindsight security labs, Q3 2013.

[9] E. Kalige, “A case study of eurograbber: How 36 million euros was stolen via malware,” tech. rep., Head of Security Operation Center, Versafe, 2012.

[10] H. Lockheimer, “Android and security..” http://googlemobile.blogspot.nl/2012/02/android-and-security.html, Feb 2012.

[11] X. Jiang, “An evaluation of the application (”app”) verification service in android 4.2.” Department of Computer Science, NC State University.

[12] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, “An android application sandbox system for suspicious software detection.,” in *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, pp. 55–62, IEEE, 2010.

[13] A. Reina, A. Fattori, and L. Cavallaro, “A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors,” *EuroSec, April*, 2013.

[14] V. Rastogi, Y. Chen, and W. Enck, “Appsplayground: Automatic security analysis of smartphone applications,” in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY ’13, (New York, NY, USA), pp. 209–220, ACM, 2013.

[15] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, “Mobile-sandbox: Having a deeper look into android applications,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC ’13, (New York, NY, USA), pp. 1808–1815, ACM, 2013.

[16] L. K. Yan and H. Yin, “Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis,” in *Proceedings of the 21st USENIX Conference on Security Symposium*, Security’12, (Berkeley, CA, USA), pp. 29–29, USENIX Association, 2012.

[17] A. A. A. Samra and O. A. Ghanem, “Analysis of clustering technique in android malware detection,” in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 729–733, July 2013.

[18] Z. Salehi, M. Ghiasi, and A. Sami, “A miner for malware detection based on api function calls and their arguments,” in *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012)*, pp. 563–568, May 2012.

[19] J. Sahs and L. Khan, “A machine learning approach to android malware detection,” in *2012 European Intelligence and Security Informatics Conference*, pp. 141–147, Aug 2012.

[20] J. Platt, B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” tech. rep., November 1999.

[21] N. Peiravian and X. Zhu, “Machine learning for android malware detection using permission and api calls,” in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 300–305, Nov 2013.

[22] “Number of available android applications - appbrain.”

[23] M. Christodorescu and S. Jha., "Static analysis of executables to detect malicious patterns," tech. rep., DTIC Document, 2006.

[24] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421–430, Dec 2007.

[25] R. Rojas, *Neural Networks: A Systematic Introduction.* New York, NY, USA: Springer-Verlag New York, Inc., 1996.

[26] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, June 2002.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] "Platform architecture - developer.android.com."

[29] "Dalvik virtual machine - www.dalvikvm.com."

[30] J. A. Bergstra and A. Ponse, "Register-machine based processes," *J. ACM*, vol. 48, pp. 1207–1241, Nov. 2001.

[31] Wikipedia, "Stack machine — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Stack%20machine&oldid=780705727`, 2017.

[32] A. Gerber and C. Craig, *Learn Android Studio: Build Android Apps Quickly and Effectively.* Berkely, CA, USA: Apress, 1st ed., 2015.

[33] CarrierIQ, "Carrieriq: Know your customer experience."

[34] F-secure, "Threat description: Trojan:android/droidkungfu.c."

[35] B. Dixon, Y. Jiang, A. Jaiantilal, and S. Mishra, "Location based power analysis to detect malicious code in smartphones," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, (New York, NY, USA), pp. 27–32, ACM, 2011.

[36] K. Savage, "Android.jsmshider is a trojan horse that opens a back door on android devices.."

[37] "Droiddream : Malware become nightmare for the android platform."

[38] M. Zheng, M. Sun, and J. C. Lui, "Droidray: A security evaluation system for customized android firmwares," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, (New York, NY, USA), pp. 471–482, ACM, 2014.

[39] Y. Zhou and X. Jiang, "Android malware genome project."

[40] Mobsf, "Mobile security framework is an intelligent, all-in-one open source mobile application (android/ios/windows) automated pen-testing framework capable of performing static, dynamic analysis and web api testing.."

[41] J. Freke, "smali/baksmali is an assembler/disassembler for the dex format used by dalvik, android's java vm implementation.."

[42] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[43] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, Sept. 1995.

[44] W. S. McCulloch and W. Pitts, "Neurocomputing: Foundations of research," ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27, Cambridge, MA, USA: MIT Press, 1988.

[45] B. tak Zhang and H. Muhlenbein, "Evolving optimal neural networks using genetic algorithms with occam's razor," *Complex Systems*, vol. 7, pp. 199–220, 1993.

[46] J. H. Holland, "Genetic algorithms.," *Scholarpedia*, vol. 7, no. 12, p. 1482, 2012.

[47] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 54–65, Jan 1994.

[48] D. Dasgupta and D. R. McGregor, "Designing application-specific neural networks using the structured genetic algorithm," in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 87–96, Jun 1992.

[49] B. Fullmer and R. Miikkulainen, "Using marker-based genetic encoding of neural networks to evolve finite-state behaviour," in *In Proceedings of the first European Conference on Artificial Life (ECAL-91*, pp. 255–262, MIT Press, 1991.

[50] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Proceedings of the 1st Annual Conference on Genetic Programming*, (Cambridge, MA, USA), pp. 81–89, MIT Press, 1996.

[51] J.-H. Kim and C.-H. Lee, "Evolutionary ordered neural network and its application to robot manipulator control," in *Proceedings of the 1996 IEEE IECON. 22nd International Conference on Industrial Electronics, Control, and Instrumentation*, vol. 2, pp. 876–880 vol.2, Aug 1996.

[52] M. Mandischer, *Representation and Evolution of Neural Networks*, pp. 643–649. Vienna: Springer Vienna, 1993.

[53] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 39–53, Jan 1994.

[54] X. Yao and Y. Liu, "Towards designing artificial neural networks by evolution," *Appl. Math. Comput.*, vol. 91, pp. 83–90, Apr. 1998.

[55] J. a. C. F. Pujol and R. Poli, "Evolving the topology and the weights of neural networks using a dual representation," *Applied Intelligence*, vol. 8, pp. 73–84, Jan. 1998.

[56] A. Desnos., "Androguard: tool to reverse engineer apk."

[57] I. RapidMiner, "Rapidminer studio:real data science, fast and simple.."